OpenDaylight Documentation Documentation

Release Nitrogen

OpenDaylight Project

Mar 05, 2018

Contents

This handbook provides details on various aspects of OpenDaylight from the user guides to the developer guides and tries to act as a single point of contact for all documentation related articles in OpenDaylight. If you would like to contribute to the Handbook please refer to the *Documentation Guide*.

CHAPTER 1

Content for OpenDaylight Users

The following content is intended for people who would like to deploy, use, or just learn more about OpenDaylight.

1.1 Release Notes

1.1.1 Target Environment

For Execution

The OpenDaylight Karaf container, OSGi bundles, and Java class files are portable and should run on any Java 7- or Java 8-compliant JVM to run. Certain projects and certain features of some projects may have additional requirements. Those are noted in the project-specific release notes.

Projects and features which have known additional requirements are:

- TCP-MD5 requires 64-bit Linux
- TSDR has extended requirements for external databases
- Persistence has extended requirements for external databases
- SFC requires addition features for certain configurations
- SXP depends on TCP-MD5 on thus requires 64-bit Linux
- SNBI has requirements for Linux and Docker
- OpFlex requires Linux
- DLUX requires a modern web browser to view the UI
- AAA when using federation has additional requirements for external tools
- VTN has components which require Linux

For Development

OpenDaylight is written primarily in Java project and primarily uses Maven as a build tool Consequently the two main requirements to develop projects within OpenDaylight are:

- A Java 8-compliant JDK
- Maven 3.1.1 or later

Applications and tools built on top of OpenDaylight using it's REST APIs should have no special requirements beyond whatever is needed to run the application or tool and make the REST calls.

In some places, OpenDaylight makes use of the Xtend language. While Maven will download the appropriate tools to build this, additional plugins may be required for IDE support.

The projects with additional requirements for execution typically have similar or more extensive additional requirements for development. See the project-specific release notes for details.

1.1.2 Known Issues and Limitations

Other than as noted in project-specific release notes, we know of the following limitations:

- Migration from prior OpenDaylight releases to Carbon has not been extensively tested. The per-project release notes include migration and compatibility information when it is known.
- There are scales beyond which the controller has been unreliable when collecting flow statistics from OpenFlow switches. In tests, these issues became apparent when managing thousands of OpenFlow switches, however this may vary depending on deployment and use cases.

1.1.3 Security Limitations

All OpenDaylight Security Advisories can be found on the Security Advisories wiki page.

The following Security Advisory is of special note to OpenDaylight Nitrogen users:

- CVE-2017-1000406

1.1.4 Project-specific Release Notes

AAA

Major Features

For each top-level feature, identify the name, url, description, etc. User-facing features are used directly by end users.

odl-aaa-shiro

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=aaa.git;a=blob_plain;f=features/shiro/features-aaa-shiro/src/main/features/features.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** ODL Shiro-based AAA implementation
- **Top Level:** Yes
- **User Facing:** Yes

- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/aaa/job/aaa-csit-1node-authn-all-nitrogen/

odl-aaa-authn

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=aaa.git;a=blob;f=features/authn/features-aaa/src/main/features/features.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** Same as odl-aaa-shiro
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/aaa/job/aaa-csit-1node-authn-all-nitrogen/

odl-aaa-cert

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=aaa.git;a=blob;f=features/authn/features-aaa/src/main/features/features.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** MD-SAL based encrypted certificate management
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/aaa/job/aaa-csit-1node-authn-all-nitrogen/

odl-aaa-cli

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=aaa.git;a=blob;f=features/authn/features-aaa/src/main/features/features.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** Basic karaf CLI commands for interacting with AAA
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/aaa/job/aaa-csit-1node-authn-all-nitrogen/

Documentation

Please provide the URL to each document at docs.opendaylight.org. If the document is under review, provide a link to the change in Gerrit.

- **User Guide(s):**
  - *Authentication, Authorization and Accounting (AAA) Services*
- **Developer Guide(s):**

- *Authentication, Authorization and Accounting (AAA) Services*

Security Considerations

- Do you have any external interfaces other than RESTCONF?

  No.

- Other security issues?

  N/A.

Quality Assurance

- Link to Sonar Report (54% code coverage)
- Link to CSIT Jobs

Migration

- Bug 7793: shiro.ini is no longer exposed in ODL Nitrogen.

shiro.ini is no longer exposed in ODL Nitrogen. A more robust mechanism is provided to configure AAA in ODL Nitrogen based on the clustered-app-config framework. A migration utility is provided and may be run by invoking the following:

```
python bin/upgrade/convert-shiro-ini-to-rest-payload <filename>
```

An XML payload is output to stdout, which can be used as a PUT payload to the aaa-app-config REST endpoint to maintain configuration from a previous version. An alternative is to write the resulting payload to the initial application config:

```
python bin/upgrade/convert-shiro-ini-to-rest-payload <filename> > etc/opendaylight/
↪datastore/initial/config/aaa-app-config.xml
```

For Example:

```
python bin/upgrade/convert-shiro-ini-to-rest-payload etc/shiro.ini > etc/opendaylight/
↪datastore/initial/config/aaa-app-config.xml
```

Compatibility

- Is this release compatible with the previous release?

  Yes.

- Any API changes?

  No.

- Any configuration changes?

  Some CLI commands were modified for security and ease of use purposes. Nothing else.

Bugs Fixed

- 6772 When it is known some features have not activated fully, do not return 401
- 8717 deprecate the existing mdsal AAA datastore impl
- 8572 remove SecureBlockingQueue which is unused
- 8724 clean AAA features

Known Issues

- List key known issues with workarounds
- 5838 token authentication fails intermittently
- Link to Open Bugs

End-of-life

- N/A

Standards

- LDAP, JDBC, ActiveDirectory (less tested)

Release Mechanics

- Link to release plan
- Describe any major shifts in release schedule from the release plan

  None.

Application-Layer Traffic Optimization (ALTO)

odl-alto-release

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=alto.git;a=blob;f=alto-release-features/features-alto/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** This is a summary feature containing the default functionalities provided by ALTO project.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/alto/job/alto-csit-1node-setup-all-nitrogen/

Documentation

- **User Guide(s):**
  - *ALTO User Guide*
- **Developer Guide(s):**
  - *ALTO Developer Guide*

Security Considerations

Besides RESTCONF, ALTO also uses customized Jetty interfaces because YANG model is not compatible with formats specified in RFC 7285.

The customized interfaces use port 8080 and are NOT protected by the AAA project. All resources exposed by customized interfaces are read-only.

Quality Assurance

- Link to Sonar Report 31.7%
- Link to CSIT Jobs
- The tests are using the OpenDaylight CSIT infrastructure.
  - How extensive was it? Not very extensive since the tests are customized to test certain functionalities.
  - What should be expected to work? The core modules (northbound and resourcepool) and also some basic components (simple-ird)
  - What has not be tested as much? Some basic components (simple-ecs and spce) and extended components (multicost, incremental update and RSA service).

Migration

Migration with data from Boron isn't supported.

Compatibility

This release is not compatible with the previous release from the developer's point of view because we have changed the namespaces for most YANG models, which involves both API changes and configuration changes (blueprint configuration files).

Java projects using the ALTO classes generated by yangtools MUST change the packages for the classes because of the namespace migration. The incompatibility can be fixed using regex replacement.

Projects using RESTCONF or the customized ALTO service do not need to migrate.

Since ALTO is migrating services to Blueprint, services depending on ALTO may also need to migrate to Blueprint instead of using CONFIG subsystem.

Bugs Fixed

- Fixed Bugs

Known Issues

Parallel query for simple-ecs service can conduct failure.

- Bug 8826

End-of-life

- Nothing deprecated, EOL.

Standards

- ALTO protocols are not compatible with YANG model
- Message types for RFC 7285 have been implemented
- ALTO project provides several basic services in RFC 7285
- Work-in-progress Internet drafts for path-vector, multi-cost, incremental updates and RSA service are also scheduled but not fully implemented.

Release Mechanics

- Link to release plan
- Major shifts:
  - Unable to finish the extensions (path-vector and RSA service)

BGP LS PCEP

Major Features

odl-bgpcep-bgp

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=bgpcep.git;a=blob;f=features/bgp/features-bgp/src/main/features/features.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** OpenDaylight Border Gateway Protocol (BGP) plugin.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:** https://logs.opendaylight.org/releng/vex-yul-odl-jenkins-1/bgpcep-csit-1node-userfeatures-all-nitrogen

odl-bgpcep-bmp

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=bgpcep.git;a=blob;f=features/bmp/features-bmp/src/main/features/features.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** OpenDaylight BGP Monitoring Protocol (BMP) plugin.

- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:** https://logs.opendaylight.org/releng/vex-yul-odl-jenkins-1/bgpcep-csit-1node-userfeatures-all-nitrogen

odl-bgpcep-pcep

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=bgpcep.git;a=blob;f=features/pcep/features-pcep/src/main/features/features.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** OpenDaylight Path Computation Element Configuration Protocol (PCEP) plugin.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:** https://logs.opendaylight.org/releng/vex-yul-odl-jenkins-1/bgpcep-csit-1node-userfeatures-all-nitrogen

Documentation

- **User Guide(s):**
  - *BGP User Guide*
  - *BGP Monitoring Protocol User Guide*
  - *PCEP User Guide*
- **Developer Guide(s):**
  - *BGP Developer Guide*
  - *BGP Monitoring Protocol Developer Guide*
  - *PCEP Developer Guide*

Security Considerations

None Known - all protocol implements the TCP Authentication Option (TCP MD5)

Quality Assurance

- Link to Sonar Report (80%)
- Link to CSIT Jobs
- User features test
- PCEP performance and scale tests
- BGP Application peer performance and scale tests
- BGP performance and scale test

- BGP clustering

  The BGP extensions were tested manually with vendor's BGP router implementation or other software implementations (exaBGP, bagpipeBGP). Also, they are covered by the unit tests and automated system tests.

- New BGP Openconfig statistics feature requires more testing.

Migration

BGP:

Protocol Configuration

First we get old configuration

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols`

Method: `GET`

where *example-bmp-monitor* old bmp monitor id

Then we insert it

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols`

Method: `POST`

Topology Configuration

First we get old configuration

URL: `/restconf/config/network-topology:network-topology`

Method: `GET`

Then we insert it

URL: `/restconf/config/network-topology:network-topology`

Method: `POST`

BMP:

First we get old configuration

URL: `/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules/config:module/odl-bmp-impl-cfg:bmp-monitor-impl/example-bmp-monitor`

Method: `GET`

example-bmp-monitor old bmp monitor id

Then we insert it

URL: `/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules/config:module/odl-bmp-impl-cfg:bmp-monitor-impl/example-bmp-monitor`

Method: `PUT`

PCEP:

There are no additional steps needed for migration to this release.

Compatibility

- Is this release compatible with the previous release? Yes
- Any API changes?
- Any configuration changes? BGP OpenConfig configuration should be used instead of previous BGP CSS configuration.

Bugs Fixed

- List of bugs fixed since the previous release

Known Issues

- BUG-6562 Support add-path in base BGP NLRI

End-of-life

- None

Standards

- RFC4271 - A Border Gateway Protocol 4 (BGP-4)
- RFC4760 - Multiprotocol Extensions for BGP-4
- RFC1997 - BGP Communities Attribute
- RFC4360 - BGP Extended Communities Attribute
- RFC4486 - Subcodes for BGP Cease Notification Message
- RFC5004 - Avoid BGP Best Path Transitions from One External to Another
- RFC7752 - North-Bound Distribution of Link-State and TE Information using BGP
- RFC5440 - Path Computation Element (PCE) Communication Protocol (PCEP)
- RFC5541 - Encoding of Objective Functions in the Path Computation Element Communication Protocol (PCEP)
- RFC5455 - Diffserv-Aware Class-Type Object for the Path Computation Element Communication Protocol
- RFC5492 - Capabilities Advertisement with BGP-4

- RFC5521 - Extensions to the Path Computation Element Communication Protocol (PCEP) for Route Exclusions
- RFC5557 - Path Computation Element Communication Protocol (PCEP) Requirements and Protocol Extensions in Support of Global Concurrent Optimization
- RFC5575 - Flow Specification
- RFC5886 - A Set of Monitoring Tools for Path Computation Element (PCE)-Based Architecture
- RFC6286 - Autonomous-System-Wide Unique BGP Identifier for BGP-4
- RFC6793 - BGP Support for Four-Octet Autonomous System (AS) Number Space
- RFC7311 - The Accumulated IGP Metric Attribute for BGP
- RFC7674 - Clarification of the Flowspec Redirect Extended Community
- RFC5668 - 4-Octet AS Specific BGP Extended Community
- RFC3107 - Carrying Label Information in BGP-4
- RFC4364 - BGP/MPLS IP Virtual Private Networks (VPNs)
- RFC7432 - BGP MPLS-Based Ethernet VPN
- RFC7911 - Advertisement of Multiple Paths in BGP
- RFC2918 - Route Refresh Capability for BGP-4
- draft-ietf-bess-evpn-overlay - A Network Virtualization Overlay Solution using EVPN
- draft-ietf-bess-evpn-vpws-02 - VPWS support in EVPN
- draft-ietf-pce-pceps - Secure Transport for PCEP
- draft-gredler-idr-bgp-ls-segment-routing-ext-03 - BGP Link-State extensions for Segment Routing
- draft-ietf-idr-bgpls-segment-routing-epe-05 - Segment Routing Egress Peer Engineering BGP-LS Extensions
- draft-ietf-idr-flow-spec-v6-06 - Dissemination of Flow Specification Rules for IPv6
- draft-ietf-idr-flowspec-redirect-ip-01 - BGP Flow-Spec Redirect to IP Action
- Stateful extensions to the Path Computation Element Protocol, December 2013
  - draft-ietf-pce-stateful-pce-07 - PCEP Extensions for Stateful PCE
  - draft-ietf-pce-pce-initiated-lsp-00 - PCEP Extensions for PCE-initiated LSP Setup in a Stateful PCE Model
- Segment routing extension to the Path Computation Element Protocol, October 2014
  - draft-ietf-pce-segment-routing-01 - PCEP Extension for segment routing
  - draft-ietf-pce-lsp-setup-type-01 - PCEP Extension for path setup type
  - draft-ietf-pce-stateful-sync-optimizations-03 - Optimizations of Label Switched Path State Synchronization Procedures for a Stateful PCE
  - draft-sivabalan-pce-binding-label-sid-01 - Carrying Binding Label/Segment-ID in PCE-based Networks
  - RFC7854 - BGP Monitoring Protocol

Release Mechanics

- Link to release plan

Bit Indexed Explicit Replication (BIER)

odl-bier-all

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=bier.git;a=blob;f=features/features-bier/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** This is a summary feature containing the default functionalities provided by BIER project.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/bier/job/bier-csit-1node-basic-all-nitrogen/

Documentation

- **User Guide(s):**
  - *BIER User Guide*
- **Developer Guide(s):**
  - *BIER Developer Guide*

Security Considerations

- Do you have any external interfaces other than RESTCONF?
  - BIER project needs to get topology information via BGP-LS and BIER configuration via NETCONF.
- Other security issues?
  - The required security issues are provided in the RESTCONF, NETCONF and BGP-LS projects.

Quality Assurance

- Link to Sonar Report 76.7%
- Link to CSIT Jobs
- Testing methodology. How extensive was it? What should be expected to work? What has not been tested as much?
- There are unit tests and integration test available under folder "test" and system test in CSIT but the NETCONF interface is not tested and will be completed in next release.

Migration

<<<<<<< HEAD * Is it possible to migrate from the previous release? If so, how? ======= * Is it possible migrate from the previous release? If so, how? >>>>>>> Update bier release-notes for nitrogen

- Migration with data from Carbon to Nitrogen is not supported.

Compatibility

- Is this release compatible with the previous release? Yes.
- Any API changes? Yes. Some BIER-TE APIs have been added and listed as following.

bier/bierman/api/src/main/yang/bier-te-config-api.yang configure-te-node configure-te-label delete-te-babel delete-te-bsl delete-te-si delete-te-bp * Any configuration changes? Yes. BGP-LS should be used instead of OpenFlow to get topology information.

Bugs Fixed

- None.

Known Issues

- None.

End-of-life

- None.

Standards

- Multicast using Bit Index Explicit Replication
- YANG Data Model for BIER Protocol

Release Mechanics

- Link to release plan
- No major changes.

Cardinal

Major Features

odl-cardinal

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=cardinal.git;a=blob;f=features/odl-cardinal/pom.xml
- **Feature Description:** This feature installs the odl-cardinal application which provides OpenDaylight health statistics, Karaf and Bundle statistics, Openflow/NETCONF specific statistics to a NMS server via SNMP protocol. And it also provides REST service to expose these statistics.
- **Top Level:** Yes
- **User Facing:** Yes

- **Experimental:** Yes
- **CSIT Test:** NA

odl-cardinal-api

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=cardinal.git;a=blob;f=features/odl-cardinal-api/pom.xml
- **Feature Description:** This feature contains the dependencies to use MDSAL features in CARDINAL.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** NA

odl-cardinal-rest

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=cardinal.git;a=blob;f=features/odl-cardinal-rest/pom.xml
- **Feature Description:** Implements a South Bound Rest interface to send configuration to REST-capable switches.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** NA

odl-cardinal-ui

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=cardinal.git;a=blob;f=features/odl-cardinal-ui/pom.xml
- **Feature Description:** This feature is the CARDINAL User Interface.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** NA

Documentation

- **User Guide(s):**
  - *Cardinal: OpenDaylight Monitoring as a Service*
- **Developer Guide(s):**
  - *Cardinal: OpenDaylight Monitoring as a Service*

Security Considerations

- SNMP agent runs on port 161,2001,2003
  - Current support is for SNMPv2c (no encryption or authentication)
- Are all interfaces exposed using RESTCONF?
  - Cardinal supports two interfaces - SNMP and RESTCONF
  - Cardinal REST APIs are RESTCONF (authentication) enabled
  - Cardinal SNMP support is through SNMP Agent (SNMPv2c as mentioned above)
  - *Link to all RESTCONF API*

Quality Assurance

- Link to Sonar Report (25.8%)
- Link to CSIT Jobs
- All modules have been unit tested. Integration tests have been performed for all major features. System tests have been performed on most major features.

Migration

- Is it possible to migrate from the previous release? If so, how?

  Yes. Migration to this release involves migrating features to Karaf 4; see the wiki for details.

Compatibility

- Is this release compatible with the previous release?

  No.
- Any API changes?

  All Karaf 3 features have been removed in favour of (compatible) Karaf 4 features

Bugs Fixed

None.

Known Issues

No known issues.

End-of-life

- N/A.

Standards

- MIB OIDS were compiled for generating java classes using 3rd party library Open-DMK(mib-gen)

Release Mechanics

- ODL CARDINAL Nitrogen release plan
- No major shifts in the release schedule from the release plan

Controller

Major Features

odl-mdsal-broker

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=controller.git;a=blob;f=features/mdsal/odl-mdsal-broker/pom.xml
- **Feature Description:** Core MD-SAL implementations.
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/controller/job/controller-csit-verify-3node-clustering/

Documentation

- **User Guide(s):**
  - *User Guide*
- **Developer Guide(s):**
  - *Controller*

Security Considerations

- Do you have any external interfaces other than RESTCONF?
  - Yes, akka uses port 2550 and by default communicates with unencrypted, unauthenticated messages. Securing akka communication isn't described here, but those concerned should look at the "Configuring SSL/TLS for Akka Remoting" section at http://doc.akka.io/docs/akka//2.4.17/scala/remoting.html.
- Other security issues?
  - No

Quality Assurance

- Link to Sonar Report (60%)
- Link to CSIT Jobs

Migration

- Is it possible to migrate from the previous release? If so, how?

  Yes, no specific steps needed unless prior updates to config subsystem modules were made via the controller-config yang-ext mount in which case the etc/opendaylight/current/controller.currentconfig.xml file must be manually edited to remove the following elements corresponding to config yang modules that were removed:

  - Remove the <data-broker> element from the <module> element with <name> binding-broker-impl
  - Remove the <module> element with <name> inmemory-binding-data-broker
  - Remove the <service> element with <name> binding-data-broker
  - Remove <capability>urn:opendaylight:params:xml:ns:yang:controller:threadpool?module=threadpool&revision=2013-04-09</capability> from <required-capabilities>

  Since the config subsystem is deprecated, it is recommended to migrate any custom configuration additions and/or changes contained in controller.currentconfig.xml and remove the file.

Compatibility

- Is this release compatible with the previous release?
  - Yes
- Any API changes?
  - No
- Any configuration changes?
  - No

Bugs Fixed

- List of bugs fixed since the previous release
  - Bugs Fixed

Known Issues

- List key known issues with workarounds
  - None
- Link to Open Bugs

End-of-life

- List of features/APIs which are EOLed, deprecated, and/or removed in this release
  - The XSQL component packaged in odl-mdsal-xsql has been removed.
  - The DataProviderService and DataBrokerService APIs and the corresponding implementations that were previously deprecated after the Hydrogen release have been removed.
  - The following config subsystem yang modules have been removed:
    - threadpool
    - threadpool-impl-fixed
    - threadpool-impl-flexible
    - threadpool-impl-scheduled
    - threadpool-impl
  - The config subsystem is officially deprecated in this release with removal planned in 2 releases (Flourine). All projects still using the config subsystem must be converted to use Blueprint.

Standards

- List of standards implemented and to what extent
  - None

Release Mechanics

- Link to release plan

Data Export/Import

Major Features

odl-daexim-all

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=daexim.git;a=blob;f=features/odl-daexim-all/src/main/feature/feature.xml
- **Feature Description:** This is a wrapper feature which includes all the sub features provided by daexim project.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/daexim/job/daexim-csit-1node-basic-only-nitrogen/

Documentation

- **User Guide(s):**
  - *Data Export/Import User Guide*
- **Developer Guide(s):**
  - *Data Export/Import Developer Guide*

Security Considerations

- Do you have any external interfaces other than RESTCONF?
  - No
- Other security issues?
  - N/A

Quality Assurance

- *Link to Sonar Report <https://sonar.opendaylight.org/overview?id=71877>* Code coverage is 79.3%.
- There are extensive unit-tests in the code.

Migration

- Is it possible to migrate from the previous release? If so, how?
  - This is the first release of the project. However, migration should work across all releases.

Compatibility

- Is this release compatible with the previous release? Yes
- Any API changes? No.
- Any configuration changes? No.

Bugs Fixed

- List of bugs fixed since the previous release
  - First release of project

Known Issues

https://bugs.opendaylight.org/buglist.cgi?quicksearch=daexim

End-of-life

- List of features/APIs which are EOLed, deprecated, and/or removed in this release
  - None

Standards

- List of standrads implemented and to what extent
  - None

Release Mechanics

- *Link to release plan <https://wiki.opendaylight.org/view/Daexim:Nitrogen:Release_Plan>*
- Describe any major shifts in release schedule from the release plan
  - None

Integration/Distribution

Major Features

odl-integration-all

- **Gitweb URL:** https://git.opendaylight.org/gerrit/gitweb?p=integration/distribution.git;a=blob;f=features/singles/odl-integration-all/pom.xml;h=c3df09e8828ff16299d96f82e78b1901de1a60ca;hb=refs/heads/stable/nitrogen
- **Description:** An aggregate feature grouping all user-facing ODL features which can be installed together without Karaf becoming unusable or without port conflicts.
- **Top Level:** Yes.
- **User Facing:** Yes, but not intended for production use (only for testing purposes).
- **Experimental:** No.
- **CSIT Test:** https://jenkins.opendaylight.org/releng/job/distribution-deploy-nitrogen

odl-integration-compatible-with-all

- **Gitweb URL:** https://git.opendaylight.org/gerrit/gitweb?p=integration/distribution.git;a=blob;f=features/singles/odl-integration-compatible-with-all/pom.xml;h=5ddd52a15cdc658ed18f4647469666b8c849cf4c;hb=refs/heads/stable/nitrogen
- **Description:** An aggregate feature grouping all user-facing ODL features which are not pro-active and which (as a group) should be compatible with most other ODL features.
- **Top Level:** Yes.
- **User Facing:** Yes, but not intended for production use (only for testing purposes).
- **Experimental:** No.

- **CSIT Test:** https://jenkins.opendaylight.org/releng/job/distribution-csit-1node-userfeatures-all-nitrogen

odl-distribution-version

- **Gitweb URL:** https://git.opendaylight.org/gerrit/gitweb?p=integration/distribution.git;a=blob;f=features/singles/odl-distribution-version/pom.xml;h=b504cbeb6889379492d33322f1c5cfa488a207a4;hb=refs/heads/stable/nitrogen
- **Description:** Allows NETCONF/RESTCONF users to determine the version of ODL they are communicating with.
- **Top Level:** Yes.
- **User Facing:** Yes.
- **Experimental:** No.
- **CSIT Test:** https://jenkins.opendaylight.org/releng/job/distribution-csit-1node-userfeatures-all-nitrogen

Karaf 4 distribution archive

- **Gitweb URL:** https://git.opendaylight.org/gerrit/gitweb?p=integration/distribution.git;a=blob;f=karaf/pom.xml;h=082fd09a2467e02e4303f8a5ce0bd42b48ad0267;hb=refs/heads/stable/nitrogen
- **Description:** Zip or tar.gz; when extracted, a self-consistent ODL installation is created.
- **Top Level:** Yes.
- **User Facing:** Yes.
- **Experimental:** No.
- **CSIT Test:** https://jenkins.opendaylight.org/releng/job/distribution-deploy-nitrogen

Documentation

- **Getting Started Guide**
  - *Clustering scripts*
  - Distribution version
- **User Guide:**
  - Distribution version
- **Developer Guide**
  - Test features
  - Distribution version

Security Considerations

- Karaf 4 exposes ssh console on port 8101. The security basically basically the same as in upstream Karaf of corresponding versions, except library version overrides implemented in odlparent:karaf-parent.

  See *Securing the Karaf container*

There is *Bug 9044 <https://bugs.opendaylight.org/show_bug.cgi?id=9044>* which limits accessibility of karaf console over SSH after restart.

- Sonar Report (0%)
  - Only 42 lines of java code.
- Test report page
- No additional manual testing was needed.

Migration

- Version feature works exactly the same as in Carbon. After migration the versions are set to the new default, configurable in runtime or via configfile. The Carbon configfile would work, but users are strongly advised to consider reporting the migrated versions.
- No upgrade path for other major features.

Compatibility

- Multiple API changes, as Nitrogen is Karaf 4 while Carbon was Karaf 3.
- Even odl-distribution-version depends on different version of Config Subsystem.
- Only cluster configuration scripts remained compatible.

Bugs Fixed

None since Carbon SR1 release.

Known Issues

- ODLPARENT-110

** Successive feature installation from karaf4 console causes bundles refresh.

* **Workaround:

  - Use –no-auto-refresh option in the karaf feature install command.

```
feature:install --no-auto-refresh odl-netconf-topology
```

  - List all the features you need in the karaf config boot file.
  - Install all features at once in console, for example:

```
feature:install odl-restconf odl-netconf-mdsal odl-mdsal-apidocs odl-
↪clustering-test-app odl-netconf-topology
```

- ODLPARENT-113

** The ssh-dss method is used by Karaf SSH console, but no longer supported by clients such as OpenSSH.

* **Workaround:

  - Use the bin/client script, which uses karaf:karaf as the default credentials.

- Use this ssh option:

```
ssh -oHostKeyAlgorithms=+ssh-dss -p 8101 karaf@localhost
```

** After restart, Karaf is unable to re-use the generated host.key file.

* **Workaround:** Delete the etc/host.key file before starting Karaf again.

- ODLPARENT-115

** Karaf is slow to start processing features after start.

* **Workaround: Use a wait loop to continue only when OpenDaylight starts restonding correctly** to requests to the desired northbound interface. If no OpenDaylight feature is installed, use bin/client in the wait loop until Karaf SSH console starts responding.

End-of-life

- All APIs and functionalities related to Karaf 3 were removed. Only Karaf 4 (or higher) will be supported from now on.

Standards

No standard implemented directly (see upstream projects).

Release Mechanics

- Release plan
- Major shifts in release schedule
  - Upstream OpenDaylight projects were slow to controbute their Karaf 4 features and debug them when needed.
  - While project deliverables were relatively on time, wiki updates were late, and documentation and milestone report were extremely late.

Dlux

Major Features

odl-dlux-core

- **Feature URL:** https://git.opendaylight.org/gerrit/#/c/61762/1/features/odl-dlux-core/pom.xml
- **Feature Description:** Core DLUX functionality
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** N/A

Documentation

- **Developer Guide(s):**
  - Dlux Getting Started

Security Considerations

- There are no security issues found.

Quality Assurance

- Link to Sonar Report
- GUI is tested mostly manually, CSITs are on the way.

Migration

- All applications are moved from Dlux project to DluxApps. Only odl-dlux-core feature remains.

Compatibility

- Release is compatible with previous.

Bugs Fixed

https://bugs.opendaylight.org/buglist.cgi?bug_status=__closed__&content=&no_redirect=1&order=Importance&product=dlux&query_format=specific

Known Issues

- Link to Open Bugs

End-of-life

- N/A

Standards

- List of standards implemented and to what extent
  - N/A

Release Mechanics

- Link to release plan

DluxApps

Major Features

odl-dluxapps-nodes

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=dluxapps.git;a=blob;f=features/odl-dluxapps-nodes/pom.xml;h=672a6a317ccfe4b51c8fddd25e3e285b3018581e;hb=3eedd3072f269d652d0ddb664a0b8bf20cf81e6e
- **Feature Description:** Application displays list of nodes in openflow (flow:1) topology.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** N/A

odl-dluxapps-topology

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=dluxapps.git;a=blob;f=features/odl-dluxapps-topology/pom.xml;h=5796edc2869cc2cf98b92b7c7bc4813848659bf7;hb=3eedd3072f269d652d0ddb664a0b8bf20cf81e6e
- **Feature Description:** Basic topology application. Displays nodes and links from openflow (flow:1) topology.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** N/A

odl-dluxapps-yangman

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=dluxapps.git;a=blob;f=features/odl-dluxapps-yangman/pom.xml;h=2a743ba9667bf60d144a8cb527a41f5323cd5a29;hb=3eedd3072f269d652d0ddb664a0b8bf20cf81e6e
- **Feature Description:** GUI for data manipulation in controller. Generates forms based on loaded Yang models. User can interact with controller without knowledge of Yang models, test them, etc. Replacement of YangUI app.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/dluxapps/job/dluxapps-csit-1node-yangman-only-carbon/

odl-dluxapps-yangui

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=dluxapps.git;a=blob;f=features/odl-dluxapps-yangui/pom.xml;h=fe56512415f35f145cdd79258125334661823 99c;hb=3eedd3072f269d652d0ddb664a0b8bf20cf81e6e
- **Feature Description:** Previous version of YangUI. Will be removed in next release.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** N/A

odl-dluxapps-yangvisualizer

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=dluxapps.git;a=blob;f=features/odl-dluxapps-yangvisualizer/pom.xml;h=daf31e33454403f8d8f5cc8c3bce4a3938cc4a35;hb=3eedd3072f269d652d0ddb664a0b8bf20cf81e6e
- **Feature Description:** Topology-like visualization of Yang models loaded in controller.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** N/A

Documentation

- **Developer Guide(s):**
  - DluxApps Developer Guide

Security Considerations

- There are no security issues found.

Quality Assurance

- Link to Sonar Report
- Link to CSIT Jobs
- GUI is tested mostly manually, CSITs are on the way.

Migration

- All applications are moved from Dlux project to DluxApps. Also feature names changed, so instead odl-dlux-* use odl-dluxapps-*. Everything else works same.

Compatibility

- Release is compatible with previous.
- API changes are in feature names - odl-dlux-* changes to odl-dluxapps-*

Bugs Fixed

https://bugs.opendaylight.org/buglist.cgi?bug_status=__closed__&content=&no_redirect=1&order=Importance&product=dluxapps&query_format=specific

Known Issues

- Link to Open Bugs

End-of-life

- odl-dluxapps-yangui - deprecated

Standards

- List of standrads implemented and to what extent
  - N/A

Release Mechanics

- Link to release plan
- UT coverage is not increased
- Yang Visualized refactor and redesign is not started

eman

Major Features

odl-eman

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=eman.git;a=blob;f=features/features-eman/src/main/features/features.xml;hb=stable/carbon
- **Feature Description:** This provides a Northbound API to the eman Information Model
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** none

Documentation

- **User Guide(s):**
  - *eman User Guide*
- **Developer Guide(s):**
  - *eman Developer Guide*

Security Considerations

- Do you have any external interfaces other than RESTCONF?

  While eman does not expose other external interfaces, it does rely upon the external interfaces exposed by the SNMP plugin.

- Other security issues?

  None

Quality Assurance

- Link to Sonar Report -
- Link to CSIT Jobs - No CSIT jobs for this experimental release
- Other manual testing and QA information - Manual testing via RESTCONF and DLUX
- Testing methodology. How extensive was it? What should be expected to work? What hasn't been tested as much?

  Testing has been manual interaction via DLUX using an SNMP simulator as described in *eman User Guide*.

Migration

- Is it possible to migrate from the previous release? If so, how?

  Yes, no changes to API

Compatibility

Yes, compatible with other features and the previous release

Bugs Fixed

- List of bugs fixed since the previous release

  None

Known Issues

- List key known issues with workarounds

  No known issues

- Link to Open Bugs

  No open bugs

End-of-life

- List of features/APIs which are EOLed, deprecated, and/or removed in this release

  None

Standards

- List of standards implemented and to what extent
  - IETF Energy Management (eman) standards. Only powerMeasurement table is currently implemented.

Release Mechanics

- Link to release plan
- Describe any major shifts in release schedule from the release plan

  None

FaaS - Fabric As A Service

Major Features

odl-faas-all

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=faas.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** This is a top level wrapper feature which includes all the sub features faas offers.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** N/A

Documentation

- **User Guide(s):**
  - *Fabric As A Service*

- **Developer Guide(s):**
  - *Fabric As A Service*

Security Considerations

- Do you have any external interfaces other than RESTCONF?
  - No
- Other security issues?
  - N/A

Quality Assurance

- Link to Sonar Report (N/A)
- There are unit tests and integration test scripts available under folder "demo" in the faas repo, these scripts can be manually invoked for testing. These tests only depend on mininet and ovs which can easily be installed on one VM.

Migration

- Is it possible to migrate from the previous release? If so, how?
  - No

Compatibility

- Is this release compatible with the previous release? Yes
- Any API changes? No.
- Any configuration changes? No.

Bugs Fixed

- List of bugs fixed since the previous release
  - None

Known Issues

https://bugs.opendaylight.org/buglist.cgi?quicksearch=faas

End-of-life

- List of features/APIs which are EOLed, deprecated, and/or removed in this release
  - None

Standards

- List of standrads implemented and to what extent
  - None

Release Mechanics

- Link to release plan
- Describe any major shifts in release schedule from the release plan
  - None

Groupbasedpolicy (GBP)

Major Features

- GBP UI - Groupbasedpoilicy User Interface
- Neutron Provider - maps neutron configuration to GBP service model
- FaaS Renderer - maps GBP service model to the common abstraction logical network models of the Fabric As A Service
- IOS-XE Renderer - maps GBP service model to IOS-XE based devices
- IOvisor Renderer - maps GBP service model to agents of the IOVisor Linux Foundation project
- Netconf Renderer - maps GBP service model to NETCONF based network elements
- OpenFlow Overlay Renderer - enable network virtualization behavior using OpenFlow
- SXP Distribution Service - enables SGT Exchange Protocol
- VPP Renderer - enable network virtualization behavior for VPP devices

odl-groupbasedpolicy-ofoverlay

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=groupbasedpolicy.git;a=blob;f=features/odl-groupbasedpolicy-ofoverlay/src/main/feature/features.xml;h=cd8aa7f1c4a08cc4d197135674d29806f71a886e;hb=refs/heads/stable/nitrogen
- **Feature Description:** Feature can be added to the base to enable a Network Virtualization behavior using OpenFlow
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/groupbasedpolicy/job/groupbasedpolicy-csit-1node-3-node-only-nitrogen/
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/groupbasedpolicy/job/groupbasedpolicy-csit-1node-6node-only-nitrogen/

odl-groupbasedpolicy-iovisor

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=groupbasedpolicy.git;a=blob;f=features/odl-groupbasedpolicy-iovisor/src/main/feature/features.xml;h=9c3df4b13a08a90d6e9fb0d32adc1eea7520d4af;hb=refs/heads/stable/nitrogen
- **Feature Description:** This renderer maps GBP service model to agents of the IOVisor Linux Foundation project
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** N/A

odl-groupbasedpolicy-neutronmapper

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=groupbasedpolicy.git;a=blob;f=features/odl-groupbasedpolicy-neutronmapper/src/main/feature/features.xml;h=072eb849b39c4399863241818495ad460fb41663;hb=refs/heads/stable/nitrogen
- **Feature Description:** This renderer maps Neutron northbound configuration to GBP service model
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/groupbasedpolicy/job/groupbasedpolicy-csit-1node-openstack-liberty-openstack-nitrogen/

odl-groupbasedpolicy-neutron-and-ofoverlay

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=groupbasedpolicy.git;a=blob;f=features/odl-groupbasedpolicy-neutron-and-ofoverlay/src/main/feature/features.xml;h=57c0b759454d00aa97a18e82b31168b37b74908d;hb=refs/heads/stable/nitrogen
- **Feature Description:** Neutron and OpenFlow Overlay
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/groupbasedpolicy/job/groupbasedpolicy-csit-1node-3-node-only-nitrogen/
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/groupbasedpolicy/job/groupbasedpolicy-csit-1node-6node-only-nitrogen/
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/groupbasedpolicy/job/groupbasedpolicy-csit-1node-openstack-liberty-openstack-nitrogen/

odl-groupbasedpolicy-vpp

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=groupbasedpolicy.git;a=blob;f=features/odl-groupbasedpolicy-vpp/src/main/feature/features.xml;h=05c6a72e95aa9f51c98f466da77569ffc4d9d012;hb=refs/heads/stable/nitrogen
- **Feature Description:** This renderer maps GBP service model to VPP devices
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** N/A

odl-groupbasedpolicy-neutron-vpp-mapper

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=groupbasedpolicy.git;a=blob;f=features/odl-groupbasedpolicy-neutron-vpp-mapper/src/main/feature/features.xml;h=394dd02b54093f4c8767889c3935cb1c4a18c45a;hb=refs/heads/stable/nitrogen
- **Feature Description:** Neutron Northbound services for VPP renderer
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/groupbasedpolicy/job/groupbasedpolicy-csit-1node-openstack-liberty-openstack-nitrogen/

odl-groupbasedpolicy-ui

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=groupbasedpolicy.git;a=tree;f=features/odl-groupbasedpolicy-ui;h=af30b7c9fc6d20de755d071b2d2e3da556d7b4a5;hb=refs/heads/stable/nitrogen
- **Feature Description:** Groupbasedpolicy User Interface
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** N/A

odl-groupbasedpolicy-ip-sgt-distribution-service

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=groupbasedpolicy.git;a=blob;f=features/odl-groupbasedpolicy-ip-sgt-distribution-service/src/main/feature/features.xml;h=f421db3463d86751dde6a161466db309bc7e33a7;hb=refs/heads/stable/nitrogen
- **Feature Description:** SXP Distribution Service
- **Top Level:** Yes
- **User Facing:** Yes

- **Experimental:** Yes
- **CSIT Test:** N/A

odl-groupbasedpolicy-ios-xe

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=groupbasedpolicy.git;a=blob;f=features/odl-groupbasedpolicy-ios-xe/src/main/feature/features.xml;h=b2498a4da528d8f43da84778516ba0677a0fbafe;hb=refs/heads/stable/nitrogen
- **Feature Description:** This renderer maps GBP service model to IOS-XE devices
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** N/A

odl-groupbasedpolicy-sxp-ep-provider

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=groupbasedpolicy.git;a=blob;f=features/odl-groupbasedpolicy-sxp-ep-provider/src/main/feature/features.xml;h=4b3aa65f93776134d75e7c76305ca23300043f98;hb=refs/heads/stable/nitrogen
- **Feature Description:** SXP integration: Endpoint provider
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** N/A

odl-groupbasedpolicy-sxp-ise-adapter

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=groupbasedpolicy.git;a=blob;f=features/odl-groupbasedpolicy-sxp-ise-adapter/src/main/feature/features.xml;h=14559f62741cee2809f92c43a27eb517a5fbef79;hb=refs/heads/stable/nitrogen
- **Feature Description:** SXP integration: ISE adapter
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** N/A

Documentation

- **Installation Guide(s):**
  - Groupbasedpolicy Installation Guide
- **User Guide(s):**

- *Group Based Policy User Guide*

Security Considerations

- No other external interfaces than RESTCONF
- No known security issues

Quality Assurance

Sonar report (64.2%)

Groupbasedpolicy CSIT:

- https://jenkins.opendaylight.org/releng/view/groupbasedpolicy/job/groupbasedpolicy-csit-1node-3-node-all-nitrogen/
- https://jenkins.opendaylight.org/releng/view/groupbasedpolicy/job/groupbasedpolicy-csit-1node-6node-all-nitrogen/
- https://jenkins.opendaylight.org/releng/view/groupbasedpolicy/job/groupbasedpolicy-csit-1node-openstack-newton-openstack-ni
- https://jenkins.opendaylight.org/releng/view/groupbasedpolicy/job/groupbasedpolicy-csit-3node-clustering-all-nitrogen/

Other manual testing and QA information

- GBP devstack demo
- GBP-SFC demo
- VPP demo

Guides about how to run demo can be found on GBP wiki under Demo

Migration

Migration from previous releases is not supported.

Compatibility

- Is this release compatible with the previous release?

  Yes
- Any API changes?

  N/A
- Any configuration changes?

  N/A

Bugs Fixed

- Fixed Bugs

Known Issues

- List key known issues with workarounds

  N/A

- Open Bugs

End-of-life

- List of features/APIs which are EOLed, deprecated, and/or removed in this release

  N/A

Standards

- List of standards implemented and to what extent

  N/A

Release Mechanics

- Release plan
- Describe any major shifts in release schedule from the release plan

  N/A

Genius (Generic Network Interface, Utilities & Services)

Genius project provides Generic Network Interfaces, Utilities & Services. Any ODL application can use these to achieve interference-free co-existence with other applications using Genius. OpendayLight Carbon Genius provides following modules –

- **Interface (logical port) Manager** allows bindings/registration of multiple services to logical ports/interfaces
- **Overlay Tunnel Manager** creates and maintains overlay tunnels between configured tunnel endpoints
- **Aliveness Monitor** provides tunnel/nexthop aliveness monitoring services
- **ID Manager** generates cluster-wide persistent unique integer IDs
- **MD-SAL Utils** provides common generic APIs for interaction with MD-SAL
- **Resource Manager** provides a resource sharing framework for applications sharing common resources e.g. table-ids, group-ids etc.
- **FCAPS Application** generates various alarms and counters for the different genius modules
- **FCAPS Framework** module collectively fetches all data generated by fcaps application. Any underlying infrastructure can subscribe for its events to have a generic overview of the various alarms and counters

Major Features

- **Features URL:** https://git.opendaylight.org/gerrit/gitweb?p=genius.git;a=blob_plain;f=features/genius-features/pom.xml

odl-genius

- **Feature Description:** This feature provides all functionalities provided by genius modules, including interface manager, tunnel manager, resource manager and ID manager and MDSAL Utils. It includes Genius APIs and implementation.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Tests:** * https://jenkins.opendaylight.org/releng/view/genius/job/genius-csit-1node-upstream-all-nitrogen/ * https://jenkins.opendaylight.org/releng/view/genius/job/genius-csit-3node-upstream-all-nitrogen/

odl-genius-rest

- **Feature Description:** This feature includes RESTCONF with 'odl-genius' feature.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Tests:** * https://jenkins.opendaylight.org/releng/view/genius/job/genius-csit-1node-upstream-all-nitrogen/ * https://jenkins.opendaylight.org/releng/view/genius/job/genius-csit-3node-upstream-all-nitrogen/

odl-genius-api

- **Feature Description:** This feature includes API for all the functionalities provided by Genius.
- **Top Level:** No
- **User Facing:** No
- **Experimental:** No
- **CSIT Tests:** * https://jenkins.opendaylight.org/releng/view/genius/job/genius-csit-1node-upstream-all-nitrogen/ * https://jenkins.opendaylight.org/releng/view/genius/job/genius-csit-3node-upstream-all-nitrogen/

odl-genius-fcaps-application

- **Feature Description:** includes genius FCAPS application.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Tests:** None

odl-genius-fcaps-framework

- **Feature Description:** includes genius FCAPS Framework.
- **Top Level:** Yes

- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Tests:** None

New capabilities and enhancements added in Nitrogen

Planned new capability added

- *Service Recovery Framework*

Enhancements added to project

1. Migration to Karaf4
2. Bug fixes

Documentation

- **Installation Guide(s):**
  - N/A
- **User Guide(s):**
  - *User Guide*
- **Developer Guide(s):**
  - *Developer Guide*

Security Considerations

- Do you have any external interfaces other than RESTCONF?
  - No
- Other security issues?
  - N/A

Quality Assurance

- Sonar Report
- CSIT Jobs
- Netvirt CSIT for Genius patches
- Netvirt Cluster CSIT for Genius patches

> **Note:** Genius is used extensively in NetVirt, so NetVirt's CSIT also provides confidence in genius.

- Other manual testing and QA information

  - N/A
- Testing methodology. How extensive was it? What should be expected to work? What hasn't been tested as much?
  - fcaps_framework and fcaps_application features hasn't been tested much.

Migration

- Is it possible to migrate from the previous release? If so, how?
  - No. OpenDaylight doesn't support migration natively for applications that use datastore.

Compatibility

- Is this release compatible with the previous release?
  - Functionality is fully backwards compatible.
- Any API changes?
  - New API added for *service-recovery </submodules/genius/docs/specs/service-recovery>* feature
- Any configuration changes?
  - No

Bugs Fixed

- List of bugs fixed since the previous release
  - Fixed BUGS

Known Issues

- List key known issues with workarounds
  - None
- Open Bugs

End-of-life

- List of features/APIs which are EOLed, deprecated, and/or removed in this release
  - N/A

Standards

- List of standards implemented and to what extent
  - N/A

Release Mechanics

- Release plan
- Describe any major shifts in release schedule from the release plan
  - N/A

Infrautils

Major Features

odl-infrautils-all

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=infrautils.git;a=blob;f=common/features/odl-infrautils-all/pom.xml;hb=stable/nitrogen
- **Feature Description:** This feature exposes all infrautils framework features
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** Yes
- **CSIT Test:** covered by Netvirt and Genius CSITs * https://jenkins.opendaylight.org/releng/view/netvirt-csit/job/netvirt-csit-1node-openstack-ocata-upstream-stateful-nitrogen/ * https://jenkins.opendaylight.org/releng/view/genius/job/genius-csit-1node-upstream-all-nitrogen/

Documentation

- **User Guide(s):**
  - Infrautils provides low-level technical framework utilities and therefore has no user guide.
- **Developer Guide(s):**
  - *Developer Guide*

Security Considerations

- No external interfaces

Quality Assurance

- Link to Sonar Report
- Project infrautils provides low-level technical framework utilities and therefore no CSIT automated system testing is available. However the same gets covered by the CSIT of users of infrautils (eg : Genius, Netvirt)

Migration

- No additional migration steps needed

Compatibility

- Is this release compatible with the previous release?
  - Functionality is fully backwards compatible.
- Any API changes?
  - New APIs added for system ready
  - New APIs added for jobcoordinator
- Any configuration changes?
  - No

Bugs Fixed

- List of bugs fixed since the previous release:

Known Issues

- There are no currently known issues

End-of-life

- This section is N/A

Standards

- This section is N/A

Release Mechanics

- Link to release plan

L2Switch

odl-l2switch-switch

- **Feature URL:** https://github.com/opendaylight/l2switch/blob/stable/carbon/features/features-l2switch/src/main/features/features.xml
- **Feature Description:** Provides a basic L2 Switch abstraction over multiple switches using OpenFlow
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/l2switch/job/l2switch-csit-1node-switch-all-carbon/

  - https://jenkins.opendaylight.org/releng/view/l2switch/job/l2switch-integration-carbon/
  - https://jenkins.opendaylight.org/releng/view/l2switch/job/l2switch-merge-carbon/
  - https://jenkins.opendaylight.org/releng/view/l2switch/job/l2switch-sonar/
  - https://jenkins.opendaylight.org/releng/view/l2switch/job/l2switch-validate-autorelease-carbon/
  - https://jenkins.opendaylight.org/releng/view/l2switch/job/l2switch-clm-carbon/
  - https://jenkins.opendaylight.org/releng/view/l2switch/job/l2switch-csit-1node-periodic-host-scalability-daily-only-carbon/
  - https://jenkins.opendaylight.org/releng/view/l2switch/job/l2switch-csit-1node-scalability-all-carbon/
  - https://jenkins.opendaylight.org/releng/view/l2switch/job/l2switch-csit-1node-scalability-only-carbon/
  - https://jenkins.opendaylight.org/releng/view/l2switch/job/l2switch-csit-1node-switch-all-carbon/
  - https://jenkins.opendaylight.org/releng/view/l2switch/job/l2switch-csit-1node-switch-only-carbon/
  - https://jenkins.opendaylight.org/releng/view/l2switch/job/l2switch-distribution-check-carbon/

Documentation

- **User Guide(s):**
  - *L2 Switch User Guide*
- **Developer Guide(s):**
  - *L2Switch Developer Guide*

Security Considerations

- Are there any known security issues?

  None.

Quality Assurance

- Link to Sonar Report
- Link to CSIT Jobs
- The tests are using the OpenDaylight CSIT infrastructure.
  - How extensive was it? Extensive, covers functionality, scalability tests.
  - What should be expected to work? The core modules like Address tracker, Packet handler, Host tracker, loop removal, simple mininet ping.
  - What has not be tested as much? Basic scalablity tests exists today, extensive scalability could be performed.

Migration

Migration with data from Boron to Carbon is not supported.

Compatibility

This release is compatible with the previous release.

Since l2switch is migrating services to Blueprint, services depending on l2switch may also need to migrate to Blueprint instead of using CONFIG subsystem.

Bugs Fixed

No bug is fixed in this release.

Known Issues

- Bug 6654

l2switch does not work well when mininet is stopped/started with no delay.

End-of-life

No Changes

Standards

None.

Release Mechanics

- Link to release plan
- No major changes.

LISP Flow Mapping

Major Features

odl-lispflowmapping-msmr

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=lispflowmapping.git;a=blob;f=features/features-lispflowmapping/src/main/features/features.xml
- **Feature Description:** This is the core feature that provides the Mapping Services and includes the LISP southbound plugin feature as well.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/lispflowmapping/job/lispflowmapping-csit-1node-msmr-all-carbon/

odl-lispflowmapping-neutron

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=lispflowmapping.git;a=blob;f=features/features-lispflowmapping/src/main/features/features.xml
- **Feature Description:** This feature provides neutron integration.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes

odl-lispflowmapping-ui

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=lispflowmapping.git;a=blob;f=features/features-lispflowmapping/src/main/features/features.xml
- **Feature Description:** This feature provides a GUI to access the Mapping Service data.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes

Documentation

- **User Guide(s):** *LISP Flow Mapping User Guide*

Security Considerations

- Do you have any external interfaces other than RESTCONF?
- Yes, the southbound plugin
  - If so, how are they secure? * LISP southbound plugin follows LISP RFC6833 security guidelines.
  - What port numbers do they use? * Port used: 4342
- Other security issues? * None

Quality Assurance

- Link to Sonar Report (68%)
- Link to CSIT Jobs
- All modules have been unit tested. Integration tests have been performed for all major features. System tests have been performed on most major features.
- Registering and retrieval of basic mappings have been tested more thoroughly. More complicated mapping policies have gone through less testing.

Migration

- Is it possible to migrate from the previous release? If so, how?
  - LISP Flow Mapping service will auto-populate the datastructures from existing MD-SAL data upon service start if the data has already been migrated separately. No automated way for transfering the data is provided in this release.

Compatibility

- Is this release compatible with the previous release?
  - Yes
- Any API changes?
  - No
- Any configuration changes?
  - No

Bugs Fixed

- List of bugs fixed since the previous release:
- 6536 MappingSystem#getWidestNegativePrefix(Eid) returns incorrect results
- 6754 Merging of negative prefixes is incorrect
- 6759 Subscribers from both Northbound and Southbound origin are stored in SimpleMapCache
- 6634 SMR scheduler task tracking unreliable
- 6782 RNegative mapping subscriptions on SB take into account NB mappings too
- 6925 Newly added mapping is removed when it has the same EID prefix as the old one
- 7018 Negative mapping in SB masking overlapping more specific positive added later to NB
- 7035 Expired mapping removed from map-cache, not removed from MD-SAL
- 6361 DAO: Empty sub-tables are not removed
- 7293 Integration tests: multi-site doesn't send SMR-invoked Map-Request on SMR
- 7586 Authentication cannot be properly disabled
- 7789 Upon negative mapping removal, subscribers should be notified to delete the negative

Known Issues

- Clustering is still an experimental feature and may have some issues particularly related to operational datastore consistency.
- Link to Open Bugs

End-of-life

- None

Standards

- The LISP implementation module and southbound plugin conforms to the IETF RFC6830 and RFC6833 , with the following exceptions:
  - In Map-Request message, M bit(Map-Reply Record exist in the MapRequest) is processed but any mapping data at the bottom of a Map-Request are discarded.
  - LISP LCAFs are limited to only up to one level of recursion, as described in the IETF LISP YANG draft.
  - No standards exist for the LISP Mapping System northbound API as of this date.

Release Mechanics

- Link to release plan
  - No major shifts from the release plan.

MD-SAL

Major Features

odl-mdsal-binding

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=mdsal.git;a=blob;f=common/features/odl-mdsal-binding/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** MDSAL Binding layer, representing mapping of YANG modeled data to respective Java Objects
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/mdsal/job/mdsal-csit-1node-periodic-bindingv1-only-nitrogen/

odl-mdsal-binding2

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=mdsal.git;a=blob;f=common/features/odl-mdsal-binding2/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** MDSAL Binding v2 layer, representing mapping of YANG modeled data to respective Java Objects
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** Yes

Documentation

- **Developer Guide(s):**
  - *MDSAL Developer guide*
  - MDSAL Binding v2 guide

Security Considerations

- MDSAL libraries are designed to be embedded and not to be a stand-alone application so security concerns need to be addressed by the application using this library.

Quality Assurance

- Link to Sonar Report (61.0% line coverage)
- Link to CSIT Jobs

Migration

- no additional steps needed for migration

Compatibility

- Release is compatible with the previous one
- No configuration changes

Bugs Fixed

- Link of fixed bugs Additionally, all issues fixed in Carbon SR1 and SR2 release trains have been integrated.

Known Issues

- List key known issues with workarounds
- Link to Open Bugs

End-of-life

- Users are advised that any API element marked as @deprecated may be removed in the next release. Users are advised to follow JavaDoc pointers and migrate to replacement APIs.
- Next release is likely to make incompatible API changes due to changes in the next yangtools release.

Standards

- relies on processing according to RFC 6020 and RFC 7950.

Release Mechanics

- Link to release plan

NEtwork MOdeling(NEMO)

Major Features

- odl nemo rest provides an abstracted intent model whose target is to enable network users/applications to describe their intent in an intuitive way without caring about the underlying physical network.
- nemo engine is the core module of NEMO project, which releases the mapping from intent to physical network. It includes two import process: intent-virtual network(VN) and virtual network-physical network(PN).
- openflow renderer is a sourthbound render to translate the mapping result of VN-PN to flow table in devices supporting for openflow protocol.
- cli render is also a sourthbound render to translate the mapping result of VN-PNinto forwarding table in devices supporting for traditional protocol.
- nemo engine ui is reponsible for showing the status of physical network, intent, generated virtual network and mapping result of VN-PN, which facilitate users to understand better the intent handling process if they want to.

NEMO Engine UI

- **Feature Name:** odl-nemo-engine-ui
- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=nemo.git;a=blob;f=nemo-features/odl-nemo-engine-ui/pom.xml;
- **Feature Description:** DSL based for the abstraction of network models and conclusion of operation patterns.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/nemo/job/nemo-csit-1node-engine-all-nitrogen/

Documentation

- **User Guide(s):**
  - *NEtwork MOdeling (NEMO)*
- **Developer Guide(s):**
  - *NEtwork MOdeling (NEMO)*

Security Considerations

- There are no security issues found.

Quality Assurance

- Link to Sonar Report 42.8%
- Link to CSIT Jobs
- Manual Tests
- External System Test is done manually, since the sandbox environment could not satisfy NEMO's requirements.

Migration

- Nothing beyond general OpenDaylight migration requirements.

Compatibility

- Nothing beyond general OpenDaylight compatibility constraints.

Bugs Fixed

- Bug Report

Known Issues

- For using openflow-renderer, requiring special switch to construct physical network. The install guide is in https://github.com/zhangmroy?tab=repositories. Other virtual switch, such as, ovs, will be support in future OpenDaylight version.
- For using cli-renderer, the physical network should be constructed with HuaWei's device: NE40E. More devices will be considered in the future OpenDaylight versions.

End-of-life

- Nothing deprecated, EOL.

Standards

- N/A

Release Mechanics

- NEMO Release Plan
- Project was on schedule

NETCONF

Major Features

For each top-level feature, identify the name, url, description, etc. User-facing features are used directly by end users.

odl-netconf-topology

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=netconf.git;a=blob;f=features/netconf-connector/odl-netconf-topology/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** NETCONF southbound plugin single-node, configuration through mdsal
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/netconf/job/netconf-csit-1node-userfeatures-all-nitrogen/

odl-netconf-clustered-topology

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=netconf.git;a=blob;f=features/netconf-connector/odl-netconf-clustered-topology/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** NETCONF southbound plugin clustered, configuration through mdsal
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/netconf/job/netconf-csit-3node-clustering-all-nitrogen/

odl-netconf-console

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=netconf.git;a=blob;f=features/netconf-connector/odl-netconf-console/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** NETCONF southbound configuration with karaf cli
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes

odl-netconf-mdsal

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=netconf.git;a=blob;f=features/netconf/odl-netconf-mdsal/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** NETCONF server for mdsal
- **Top Level:** Yes

- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/netconf/job/netconf-csit-1node-userfeatures-all-nitrogen/

odl-restconf

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=netconf.git;a=blob;f=features/restconf/odl-restconf/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** Restconf
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:** Tested by any suite that uses Restconf

odl-mdsal-apidocs

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=netconf.git;a=blob;f=features/restconf/odl-mdsal-apidocs/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** MDSal - apidocs
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No

odl-yanglib

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=netconf.git;a=blob;f=features/yanglib/odl-yanglib/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** Yanglib server
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No

odl-netconf-callhome-ssh

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=netconf.git;a=blob;f=features/netconf-connector/odl-netconf-callhome-ssh/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** Netconf call home
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No

- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/netconf/job/netconf-csit-1node-callhome-all-nitrogen/

Documentation

Please provide the URL to each document at docs.opendaylight.org. If the document is under review, provide a link to the change in Gerrit.

- **User Guide(s):**
  - *NETCONF User Guide*
- **Developer Guide(s):**
  - *NETCONF Developer Guide*

Security Considerations

- Do you have any external interfaces other than RESTCONF?

  Yes, we have md-sal and css netconf servers. Also server for netconf call-home.

  - If so, how are they secure?

    NETCONF over SSH

  - What port numbers do they use?

    Please see https://wiki.opendaylight.org/view/Ports. Netconf call-home uses TCP port 6666

- Other security issues?

  None that we are aware of

Quality Assurance

- Link to Sonar Report Test coverage percent: 63.3%
- Link to CSIT Jobs

Migration

- Is it possible to migrate from the previous release? If so, how?

  Yes.

  Several config subsystem yang modules were removed therefore if prior updates to config subsystem modules were made via the controller-config loopback mountpoint then the etc/opendaylight/current/controller.currentconfig.xml file must be manually edited to remove elements corresponding to the removed config yang modules. These include the elements from the following XML files under etc/opendaylight/karaf:

  - 10-rest-connector.xml
  - 10-restconf-service.xml

  The mechanism of spawning netconf connectors via the controller-config loopback mountpoint was removed so any previously configured connectors must be migrated to the config datastore and the elements removed from the controller.currentconfig.xml file.

The netconf yanglib config subsystem module was removed so any configuration change that was made via the controller-config loopback mountpoint must be migrated to the config datastore (see the Compatibility section) and must also be removed from the controller.currentconfig.xml file.

Since the config subsystem is deprecated, it is recommended to migrate any custom configuration additions and/or changes contained in controller.currentconfig.xml and remove the file.

Compatibility

- Is this release compatible with the previous release?

  Yes

- Any API changes?

  No

- Any configuration changes?
  - The restconf northbound feature is now started via blueprint instead of the config subsystem. The corresponding config yang file, opendaylight-rest-connector.yang, and the 10-rest-connector.xml file installed under etc/opendaylight/karaf have been removed. The restconf configuration attributes (specifically websocket-port) are now specified via the etc/org.opendaylight.restconf.cfg file.
  - The JSONRestconfService API is no longer advertised via the config subsystem and the corresponding config yang file, sal-restconf-service.yang, and the 10-restconf-service.xml file installed under etc/opendaylight/karaf have been removed. The JSONRestconfService must now be obtained directly from the OSGi service registry (preferably via blueprint).
  - The netconf yanglib feature is now now started via blueprint instead of the config subsystem and is configured using the yanglib:yanglib-config container defined in yanglib.yang via the config datastore.

Bugs Fixed

- List of bugs fixed since the previous release

  https://bugs.opendaylight.org/buglist.cgi?bug_status=RESOLVED&bug_status=VERIFIED&chfield=resolution&chfieldfrom=2017-07-12&chfieldto=Now&chfieldvalue=FIXED&list_id=78801&product=netconf&query_format=advanced&resolution=FIXED

Known Issues

- List key known issues with workarounds

  None

- Link to Open Bugs

End-of-life

- List of features/APIs which are EOLed, deprecated, and/or removed in this release

  The mechanism of spawning netconf connectors via the config subsystem's controller-config loopback mountpoint was deprecated in the previous release and has been removed.

Standards

- RFC 6241 - Network Configuration Protocol (NETCONF)
- RFC 6470 - Base Notifications partly supported, netconf-config-change unsupported
- draft-ietf-yang-library-06
- draft-bierman-netconf-restconf-04
- RFC 8040 - RESTCONF protocol

Release Mechanics

- Link to release plan
- Describe any major shifts in release schedule from the release plan

  No shifts

NetVirt

Major Features

Feature Name

- **Feature Name:** odl-netvirt-openstack
- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=netvirt.git;a=blob;f=vpnservice/features/odl-netvirt-openstack/pom.xml;hb=HEAD
- **Feature Description:** This feature provides a network virtualization solution.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/netvirt-csit/job/netvirt-csit-1node-openstack-ocata-upstream-stateful-ca

Documentation

- **User Guide(s):**
  - *NetVirt User Guide*
  - *OpenStack with NetVirt*
- **Developer Guide(s):**
  - *NetVirt Developer Guide*
- **Contributor Guide(s):**
  - *NetVirt Contributor Guide*

Security Considerations

No known issues.

Quality Assurance

- Sonar Report
- All CSIT Jobs
- Main test jobs

Migration

Nothing beyond general migration requirements.

Compatibility

Nothing beyond general compatibility requirements.

Bugs Fixed

- Closed Bugs

Known Issues

- Open Bugs

End-of-life

N/A

Standards

N/A

Release Mechanics

- Release Plan
- Project was on schedule

Neutron Northbound

Major Features

- YANG model for OpenStack Neutron integration
- REST API for OpenStack Neutron integration which stores necessary information into Neutron YANG model
- Logger to log activity on Neutron YANG models
- helper library to support for OpenStack service providers

odl-neutron-service

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=neutron.git;a=blob;f=features/production/odl-neutron-service/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** This is a top level feature to load Neutron northbound functionality.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:** no CSIT tests as test weiver had been requested. OpenStack CI results can be found from https://review.openstack.org/#/q/project:openstack/networking-odl

odl-neutron-northbound-api

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=neutron.git;a=blob;f=features/production/odl-neutron-northbound-api/pom.xml;hhb=refs/heads/stable/nitrogen
- **Feature Description:** This feature provides REST API for OpenStack Neutron
- **Top Level:** No
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:** no CSIT tests as test weiver had been requested. OpenStack CI results can be found from https://review.openstack.org/#/q/project:openstack/networking-odl

odl-neutron-spi

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=neutron.git;a=blob;f=features/production/odl-neutron-spi/pom.xml;hb=stable/nitrogen
- **Feature Description:** SPI for Neutron northbound feature
- **Top Level:** No
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:** no CSIT tests as test weiver had been requested. OpenStack CI results can be found from https://review.openstack.org/#/q/project:openstack/networking-odl

odl-neutron-transcriber

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=neutron.git;a=blob;f=features/production/odl-neutron-transcriber/pom.xml;hb=stable/nitrogen
- **Feature Description:** Data converter from/to REST API to/from MD-SAL YANG model
- **Top Level:** No
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:** no CSIT tests as test weiver had been requested. OpenStack CI results can be found from https://review.openstack.org/#/q/project:openstack/networking-odl

odl-neutron-logger

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=neutron.git;a=blob;f=features/production/odl-neutron-logger/pom.xml;hb=stable/nitrogen
- **Feature Description:** Logger on activity on Neutron YANG models
- **Top Level:** No
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:** no CSIT tests as test weiver had been requested. OpenStack CI results can be found from https://review.openstack.org/#/q/project:openstack/networking-odl

odl-neutron-hostconfig-ovs

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=neutron.git;a=blob;f=features/production/odl-neutron-hostconfig-ovs/pom.xml;hb=stable/nitrogen
- **Feature Description:** Helper library to support hostconfig for OpenStack service provider with Open vSwitch
- **Top Level:** No
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:** no CSIT tests as test weiver had been requested. OpenStack CI results can be found from https://review.openstack.org/#/q/project:openstack/networking-odl

odl-neutron-hostconfig-vpp

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=neutron.git;a=blob;f=features/production/odl-neutron-hostconfig-vpp/pom.xml;hb=stable/nitrogen
- **Feature Description:** Helper library to support hostconfig for OpenStack service provider with VPP
- **Top Level:** No
- **User Facing:** No
- **Experimental:** No

- **CSIT Test:** no CSIT tests as test weiver had been requested. OpenStack CI results can be found from https://review.openstack.org/#/q/project:openstack/networking-odl

Documentation

- **User Guide(s):**
  - *Neutron Service User Guide* is a guide for cloud admin who deploys OpenStack with OpenDaylight.
- **Developer Guide(s):**
  - *Neutron Northbound* is a guide for those who develops new Neutron Northbound API which OpenStack Neutron talks to.
  - *Neutron Service Developer Guide* is a guide for those who develops new OpenStack Service Provider like netvirt, group-based-policy.

Security Considerations

- Do you have any external interfaces other than RESTCONF?

  Yes. REST API for OpenStack Neutron.
  - If so, how are they secure? It's authenticated by AAA.
  - What port numbers do they use? 8080 and 8181 by default. 8087 is also used by networking-odl/devstack.
- Other security issues?

  None.

Quality Assurance

- Link to Sonar Report (78.2%)
- Link to CSIT Jobs N/A
- Other manual testing and QA information
  - OpenStack CI results can be found from https://review.openstack.org/#/q/project:openstack/networking-odl
  - failure rate of OpenStack CI http://grafana.openstack.org/dashboard/db/networking-odl-failure-rate
  - Other OpenDaylight projects which provides OpenStack Service (e.g. netvirt, group-based-policy and vtn etc..) have their own system tests which also exercise Neutron Norhtbound. Which give coverage.
- Testing methodology. How extensive was it? What should be expected to work? What hasn't been tested as much?
  - Unit test: coverage 24.9%
  - Integration test: coverage 75.8%
  - OpenStack CI

Migration

- Is it possible to migrate from the previous release? If so, how?

  No as incompatble change was introduced.

Compatibility

- Is this release compatible with the previous release?

  Yes.

- Any API changes?

  No.

- Any configuration changes?

  No.

Bugs Fixed

- List of bugs fixed since the previous release
  - Link to Bugs fixed

Known Issues

- List key known issues with workarounds

  None

- Link to Open Bugs

End-of-life

- List of features/APIs which are EOLed, deprecated, and/or removed in this release

  N/A

Standards

- List of standrads implemented and to what extent

  OpenStack Neutron API ODL Neutron Northbound REST API is based on OpenStack Neutron API and OpenStack Neutron implementation. So the two REST APIs are similar inherently, but different if necessary for technical reason. The goal of ODL Neutron Northbound project is to help OpenStack ODL driver for OpenStack Neutron (networking-odl) and ODL OpenStack Service Provider(netvirt, group-based-policy, and vtn etc...). Not re-implement OpenStack Neutron API.

Release Mechanics

- Link to release plan
- Describe any major shifts in release schedule from the release plan
  - Postponed YANG model change to drop tenant-id, make status operational to Nitrogen cycle
  - update supported QoS rules
  - TAPaaS

NIC

Major Features

odl-nic-core-mdsal

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=nic.git;a=blob_plain;f=features/odl-nic-core-mdsal/pom.xml;hb=stable/nitrogen
- **Feature Description:** This feature contains the dependencies to use MDSAL features on NIC
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/nic/job/nic-csit-1node-basic-all-nitrogen/

odl-nic-intent-common

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=nic.git;a=blob_plain;f=features/odl-nic-intent-common/pom.xml;hb=stable/nitrogen
- **Feature Description:** This feature contains the lifecycle management for Intents, also is used to join two major features 'intent-statemachine' and 'intent-listeners'. This feature enable NIC to work with different renderers at the same time.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/nic/job/nic-csit-1node-basic-all-nitrogen/

Documentation

- **User Guide(s):**
  - *Network Intent Composition (NIC) User Guide*
- **Developer Guide(s):**
  - *Network Intent Composition (NIC) Developer Guide*

Additional information can be found at the NIC wiki page.

Security Considerations

- Do you have any external interfaces other than RESTCONF?

  No

- Other security issues?

  N/A

Quality Assurance

- Link to Sonar Report (33.2% code coverage)
- Link to CSIT Jobs
- Other manual testing and QA information
- Testing methodology. How extensive was it? What should be expected to work? What has not been tested as much?

  There are a guide to evaluate manual tests using NIC on our wiki page

Migration

- Is it possible to migrate from the previous release? If so, how?

  Migration with user configuration and state is not supported.

Compatibility

- Is this release compatible with the previous release?

  Yes

- Any API changes?

  No

- Any configuration changes?

  No

Bugs Fixed

- List of bugs fixed since the previous release

Known Issues

- List key known issues with workarounds

  For Nitrogen release, NIC contains multiple renderers, where each renderer can be used at the same time without uninstalling NIC and restarting ODL.

End-of-life

- List of features/APIs which are EOLed, deprecated, and/or removed in this or last release
- odl-nic-renderer-nemo
- odl-nic-renderer-vtn
- odl-nic-core-hazelcast

Standards

- List of standards implemented and to what extent

  N/A

Release Mechanics

- Link to release plan
- Describe any major shifts in release schedule from the release plan

  Capability to use multiple Renderers at the same time. Capability to support new Intent definitions. Integration with Intent State Machine. Integration with BGPCEP project in order to advertise routes using Intents. Apply OpenFlow rules using OpenFlowPlugin Meters Integration with Genius project in order to manage the Meter IDs.

OCP-plugin

Major Features

odl-ocpplugin-southbound

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=ocpplugin.git;a=blob;f=features/features-ocpplugin/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** Handling of OCP v4.1.1 request/response messages
- **Top Level:** No
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/ocpplugin/job/ocpplugin-csit-1node-get-all-nitrogen

odl-ocpplugin-app-ocp-service

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=ocpplugin.git;a=blob;f=features/features-ocpplugin/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** User facing interface and rrh-agent registration and lifecycle management
- **Top Level:** Yes
- **User Facing:** Yes

- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/ocpplugin/job/ocpplugin-csit-1node-get-all-nitrogen

Documentation

- **User Guide(s):**
  - *OCP Plugin User Guide*
- **Developer Guide(s):**
  - *OCP Plugin Developer Guide*

Security Considerations

- Do you have any external interfaces other than RESTCONF?
  - There is no futher secure description on the OCP 4.1.1 spec, it's out of our design scope, so there is no extenal security interface other than RESTCONF.
- Other security issues?
  - No other security issue

Quality Assurance

- Link to Sonar Report (60.8%)
- Link to CSIT Jobs
- Other manual testing and QA information
  - More detail testing, https://wiki.opendaylight.org/view/OCP_Plugin:Nitrogen_System_Test_Report

Migration

- Is it possible to migrate from the previous release? If so, how?
  - Yes, there is no config change and there is not a need to migrate data in the datastore.

Compatibility

- Is this release compatible with the previous release?
  - Release is compatible with previous.
- Any API changes?
  - N/A
- Any configuration changes?
  - N/A

Bugs Fixed

- List of bugs fixed since the previous release
  - Fixed BUGS

Known Issues

- List key known issues with workarounds
  - N/A

End-of-life

- List of features/APIs which are EOLed, deprecated, and/or removed in this release
  - Migrated from using the CSS to blueprint, deprecated CSS

Standards

- List of standards implemented and to what extent
  - OCP(ORI [Open Radio Interface] C&M [Control and Management]) v4.1.1
  - The ocpplugin poeject extended connection establishment and state machines used on both ends of the connection.

Release Mechanics

- Link to release plan
- Describe any major shifts in release schedule from the release plan
  - N/A

ODL Parent

Major Features

There are no user-visible features.

Documentation

- **Developer Guide(s):**
  - *ODL Parent Developer Guide*

Security Considerations

- Do you have any external interfaces other than RESTCONF?

  No.

- Other security issues?

  No.

Quality Assurance

- Link to Sonar Report (6.4% coverage)
- There are no CSIT jobs, ODL Parent has a system test waiver
- ODL Parent is tested by all builds, and manually tested by running the basic Karaf container and verifying the scripts we modify (`client` in particular).
- We verify the following:
  - `start` starts the Karaf container. (in a working state, capable of installing features)
  - `client` can connect to a running Karaf container.
  - `stop` stops a running Karaf container.

Migration

- Is it possible to migrate from the previous release? If so, how?

  Yes. Migration to this release involves migrating features to Karaf 4; see the wiki for details.

Compatibility

- Is this release compatible with the previous release?

  No.

- Any API changes?

  All Karaf 3 features have been removed in favour of (compatible) Karaf 4 features.

- Any configuration changes?

  No. ODL Parent has no user-visible configuration.

Bugs Fixed

- 4219: Milestone: Upgrade karaf to 4.0.1 or later
- 6278: karaf-parent belongs in odlparent
- 6523: org.osgi.service.blueprint.container.ComponentDefinitionException Caused by: java.lang.RuntimeException at org.objectweb.asm.MethodVisitor.visitParameter
- 6652: Migrate to karaf-maven-plugin features generation
- 6709: Migrate dependent projects off opendaylight-karaf-empty

- 6730: Upgrade shiro to current compatible release
- 7446: Milestone: upgrade to guava-22
- **‘7813: karaf: do not package spring<https://bugs.opendaylight.org/show_bug.cgi?id=7813>‘_**

Known Issues

- Link to Open Bugs

End-of-life

- N/A.

Standards

- N/A.

Release Mechanics

- Link to release plan

OF-CONFIG

Major Features

odl-of-config-all

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=of-config.git;a=blob;f=features/features-of-config/src/main/features/features.xml;h=86615e2f22ebc8f21b403185d3a600aa2a463588;hb=HEAD
- **Feature Description:** This is a top level wrapper feature which includes all the sub features of-config offers.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** N/A

Documentation

- **User Guide(s):**
  - *OF-CONFIG User Guide*
- **Developer Guide(s):**
  - *OF-CONFIG Developer Guide*

Security Considerations

- Do you have any external interfaces other than RESTCONF?
  - No. This project indirectly uses the NETCONF project to connect to devices.
- Other security issues?
  - N/A

Quality Assurance

- Link to Sonar Report (71.4% code coverage)
- Other manual testing and QA information
- Testing methodology. How extensive was it? What should be expected to work? What has not been tested as much?
- External System Test is almost done manually. The test has coverd all external APIs of OF-CONFIG and all supplementary options based on OF-CONFIG 1.2.

Migration

- Is it possible to migrate from the previous release? If so, how?

Yes, there are no additional steps for migration.

Compatibility

- Is this release compatible with the previous release? Yes.
- Any API changes? No.
- Any configuration changes? No.

Bugs Fixed

None.

Known Issues

- No known issues.

End-of-life

- List of features/APIs which are EOLed, deprecated, and/or removed in this release

  N/A

Standards

- List of standrads implemented and to what extent
- OF-CONFIG 1.2

Release Mechanics

- Link to release plan
- Describe any major shifts in release schedule from the release plan

  N/A

OpenFlowPlugin Project

Major Features

odl-openflowjava-protocol

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;a=blob;f=openflowjava/features-openflowjava-aggregator/odl-openflowjava-protocol/pom.xml
- **Feature Description:** OpenFlow protocol implementation
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-all-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-frs-only-nitroge
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-3node-clustering-only-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-bulkomatic-perf-daily
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-3node-periodic-bulkomatic-clustering
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-gate-scale-stats-colle
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-scale-stats-collection
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-sw-scalability-daily-
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-link-scalability-daily

odl-openflowplugin-app-config-pusher

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;a=blob;f=features-aggregator/odl-openflowplugin-app-config-pusher/pom.xml
- **Feature Description:** Pushes node configuration changes to OpenFlow device
- **Top Level:** Yes

- **User Facing:** No
- **Experimental:** No
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-all-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-frs-only-nitroge
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-3node-clustering-only-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-bulkomatic-perf-daily
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-3node-periodic-bulkomatic-clustering
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-gate-scale-stats-colle
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-scale-stats-collection
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-sw-scalability-daily-
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-link-scalability-daily

odl-openflowplugin-app-forwardingrules-manager

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;a=blob;f=features-aggregator/odl-openflowplugin-app-forwardingrules-manager/pom.xml
- **Feature Description:** Sends changes in config datastore to OpenFlow device incrementally. forwardingrules-manager can be replaced with forwardingrules-sync and vice versa.
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-all-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-gate-scale-stats-colle

odl-openflowplugin-app-forwardingrules-sync

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;a=blob;f=features-aggregator/odl-openflowplugin-app-forwardingrules-sync/pom.xml
- **Feature Description:** Sends changes in config datastore to OpenFlow devices taking previous state in account and doing diffs between previous and new state. forwardingrules-sync can be replaced with forwardingrules-manager and vice versa.
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** Yes
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-frs-only-nitroge
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-scale-stats-collection

odl-openflowplugin-app-table-miss-enforcer

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;a=blob;f=features-aggregator/odl-openflowplugin-app-table-miss-enforcer/pom.xml
- **Feature Description:** Sends table miss flows to OpenFlow device when it connects
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-sw-scalability-daily-
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-link-scalability-daily-

odl-openflowplugin-app-topology

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;a=blob;f=features-aggregator/odl-openflowplugin-app-topology/pom.xml
- **Feature Description:** Discovers topology of connected OpenFlow devices
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-all-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-frs-only-nitroge
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-3node-clustering-only-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-bulkomatic-perf-daily
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-3node-periodic-bulkomatic-clustering
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-gate-scale-stats-colle
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-scale-stats-collection-
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-sw-scalability-daily-
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-link-scalability-daily-

odl-openflowplugin-nxm-extensions

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;a=blob;f=extension/features-extension-aggregator/odl-openflowplugin-nxm-extensions/pom.xml
- **Feature Description:** Support for OpenFlow Nicira Extensions
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No

- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/netvirt/job/netvirt-csit-1node-openstack-newton-upstream-stateful-snat-conntr

odl-openflowplugin-onf-extensions

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;a=blob;f=extension/features-extension-aggregator/odl-openflowplugin-onf-extensions/pom.xml
- **Feature Description:** Support for Open Networking Foundation Extensions
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** No

odl-openflowplugin-flow-services

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;a=blob;f=features-aggregator/odl-openflowplugin-flow-services/pom.xml
- **Feature Description:** Wrapper feature for standard applications
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-all-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-frs-only-nitroge
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-3node-clustering-only-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-bulkomatic-perf-daily
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-3node-periodic-bulkomatic-clustering
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-gate-scale-stats-colle
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-scale-stats-collection
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-sw-scalability-daily-
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-link-scalability-daily

odl-openflowplugin-flow-services-rest

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;a=blob;f=features-aggregator/odl-openflowplugin-flow-services-rest/pom.xml
- **Feature Description:** Wrapper + REST interface
- **Top Level:** Yes
- **User Facing:** Yes

- **Experimental:** No
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-all-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-frs-only-nitroge
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-3node-clustering-only-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-bulkomatic-perf-daily
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-3node-periodic-bulkomatic-clustering
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-gate-scale-stats-colle
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-scale-stats-collection
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-sw-scalability-daily-
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-link-scalability-daily

odl-openflowplugin-flow-services-ui

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;a=blob;f=features-aggregator/odl-openflowplugin-flow-services-ui/pom.xml
- **Feature Description:** Wrapper + REST interface + UI
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-all-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-frs-only-nitroge
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-3node-clustering-only-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-bulkomatic-perf-daily
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-3node-periodic-bulkomatic-clustering
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-gate-scale-stats-colle
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-scale-stats-collection
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-sw-scalability-daily-
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-link-scalability-daily

odl-openflowplugin-nsf-model

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;a=blob;f=features-aggregator/odl-openflowplugin-nsf-model/pom.xml
- **Feature Description:** OpenFlowPlugin YANG models
- **Top Level:** Yes
- **User Facing:** No

- **Experimental:** No
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-all-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-frs-only-nitroge
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-3node-clustering-only-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-bulkomatic-perf-daily
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-3node-periodic-bulkomatic-clustering
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-gate-scale-stats-colle
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-scale-stats-collection-
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-sw-scalability-daily-
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-link-scalability-daily-

odl-openflowplugin-southbound

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;a=blob;f=features-aggregator/odl-openflowplugin-southbound/pom.xml
- **Feature Description:** Southbound API implementation
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-all-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-flow-services-frs-only-nitroge
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-3node-clustering-only-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-bulkomatic-perf-daily
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-3node-periodic-bulkomatic-clustering
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-gate-scale-stats-colle
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-scale-stats-collection-
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-sw-scalability-daily-
  - https://jenkins.opendaylight.org/releng/view/openflowplugin/job/openflowplugin-csit-1node-periodic-link-scalability-daily-

Documentation

- **User Guide(s):**
  - *OpenFlow Plugin Project User Guide*
- **Developer Guide(s):**
  - *OpenFlow Plugin Project Developer Guide*

Security Considerations

- Do you have any external interfaces other than RESTCONF? Yes, OpenFlow devices
- Other security issues?
  - Insecure OpenFlowPlugin <–> OpenFlow device connections
  - Topology spoofing: non authenticated LLDP packets to detect links between switches which makes it vulnerable to a number of attacks, one of which is topology spoofing The problem is that all controllers we have tested set chassisSubtype value to the MAC address of the local port of the switch, which makes it easy for an adversary to spoof that switch since controllers use that MAC address as a unique identifier of the switch. By intercepting clear LLDP packets containing MAC addresses, a malicious switch can spoof other switches to falsify the controller's topology graph.
  - DoS: an adversary switch could generate LLDP flood resulting in bringing down the openflow network
  - DoS attack when the switch rejects to receive packets from the controller

Quality Assurance

- Link to Sonar Report (73.8)
- Link to CSIT Jobs

Migration

- Is it possible to migrate from the previous release? If so, how?

  Yes, OpenFlowJava was merged into OpenFlowPlugin, so if project was relying on **org.opendaylight.openflowjava** dependency, it was moved to **org.opendaylight.openflowplugin.openflowjava**. Feature **odl-openflowjava-protocol** is same as before.

Compatibility

- Is this release compatible with the previous release? Yes
- Any API changes? Yes, *mastershipChangeServiceManager <https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;a=*
  api/src/main/java/org/opendaylight/openflowplugin/api/openflow/mastership/MastershipChangeServiceManager.java>
  and *reconcilliationManager <https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;a=blob;f=applications/reconciliat*
  framework/src/main/java/org/opendaylight/openflowplugin/applications/reconciliation/ReconciliationManager.java>
  blueprint services was added
- Any configuration changes? Yes, **bundle-based-reconcilliation** flag was added to *configuration file <https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;a=blob;f=openflowplugin-blueprint-config/src/main/resources/initial/openflowplugin.cfg#l33>*

Bugs Fixed

- List of bugs fixed since the previous release

  https://bugs.opendaylight.org/buglist.cgi?chfieldfrom=2017-05-25&chfieldto=2017-09-12&list_id=78767&product=openflowplugin&query_format=advanced&resolution=FIXED

Known Issues

- List key known issues with workarounds: None
- Link to Open Bugs

End-of-life

- List of features/APIs which are EOLed, deprecated, and/or removed in this release: None

Standards

OpenFlow versions:

- OpenFlow1.3.2
- OpenFlow1.0.0

Release Mechanics

- Link to release plan

OpFlex

Major Features

OpFlex Agent

OpFlex Agent provides support for local enforcement of group-based policy model synced using the OpFlex protocol using an Open vSwitch-based bridge. Supported renderer currently works with Cisco ACI.

libopflex

libopflex provides an implementation of the OpFlex protocol along with an in-memory managed object database for managing OpFlex data.

genie

Genie provides a modeling language and code generator for producing data models that work with libopflex. Genie also contains the group-based policy model that is used by the OpFlex Agent.

Documentation

Please provide the URL to each document at docs.opendaylight.org. If the document is under review, provide a link to the change in Gerrit.

- **Installation Guide(s):**
  - *OpFlex agent-ovs Install Guide*

- **User Guide(s):**
  - *OpFlex agent-ovs User Guide*
- **Developer Guide(s):**
  - *OpFlex libopflex Developer Guide*
  - *OpFlex genie Developer Guide*
  - *OpFlex agent-ovs Developer Guide*

Security Considerations

- Do you have any external interfaces other than RESTCONF?
  - No.
- Other security issues?
  - None.

Quality Assurance

- OpFlex projects are tested with extensive unit testing as well as Cisco-internal automated testing with ACI.
- Unit tests run as part of regular build

Migration

- Simply install and restart daemons.

Compatibility

OpFlex GBP model and configuration files remain backward compatible.

Changes since previous release

- Advertise external services on their interface with ARP/ND packets when the interface comes up.
- Always allow ARP and ND packets without contracts
- Improved robustness of agent shutdown and OpenFlow socket reconnections
- Clean up endpoint-related OpenFlow rules when EPG is removed with endpoint remaining

Known Issues

- None

End-of-life

- None

Standards

- OpFlex protocol (reference implementation)

OVSDB Project

Major Features

odl-ovsdb-southbound-api

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=ovsdb.git;a=blob;f=southbound/southbound-features/odl-ovsdb-southbound-api/pom.xml;h=7baad461a78e7dd311516ec03b7dbf7c9a0679aa;hb=refs/heads/stable/nitrogen
- **Feature Description:** This feature provides the YANG models for northbound users to configure the OVSDB device. These YANG models are designed based on the OVSDB schema. This feature does not provide the implementation of YANG models. If user/developer prefer to write their own implementation they can use this feature to load the YANG models in the controller.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/ovsdb/job/ovsdb-csit-1node-upstream-southbound-all-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/ovsdb/job/ovsdb-csit-3node-upstream-clustering-only-nitrogen/

odl-ovsdb-southbound-impl

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=ovsdb.git;a=blob;f=southbound/southbound-features/odl-ovsdb-southbound-impl/pom.xml;h=261a85eacef24c1985a11f60d018816b1f880b10;hb=refs/heads/stable/nitrogen
- **Feature Description:** This feature is the main feature of the OVSDB Southbound plugin. This plugin handle the OVS device that supports the OVSDB schema and uses the OVSDB protocol. This feature provides the implementation of the defined YANG models. Developers developing the in-controller application and want to leverage OVSDB for device configuration can add dependency on this feature and it will load all the required modules.
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/ovsdb/job/ovsdb-csit-1node-upstream-southbound-all-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/ovsdb/job/ovsdb-csit-3node-upstream-clustering-only-nitrogen/

odl-ovsdb-southbound-impl-rest

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=ovsdb.git;a=blob;f=southbound/southbound-features/odl-ovsdb-southbound-impl-rest/pom.xml;h=6a14e3f90fceba595695d69cdab2571e1a306999;hb=refs/heads/stable/nitrogen
- **Feature Description:** This feature is the wrapper feature that installs the odl-ovsdb-southbound-api & odl-ovsdb-southbound-impl feature with other required features for restconf access to provide a functional OVSDB southbound plugin. Users, who want to develop application that manages the OVSDB supported devices but want to runs the application outside of the OpenDaylight controller must install this feature.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/ovsdb/job/ovsdb-csit-1node-upstream-southbound-all-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/ovsdb/job/ovsdb-csit-3node-upstream-clustering-only-nitrogen/

odl-ovsdb-hwvtepsouthbound-api

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=ovsdb.git;a=blob;f=hwvtepsouthbound/hwvtepsouthbound-features/odl-ovsdb-hwvtepsouthbound-api/pom.xml;h=e08f4233a6025da2d84dc1d87b6fb220a187e070;hb=refs/heads/stable/nitrogen
- **Feature Description:** This feature provides the YANG models for northbound users to configure the device that supports OVSDB Hardware vTEP schema. These YANG models are designed based on the OVSDB Hardware vTEP schema. This feature does not provide the implementation of YANG models. If user/developer prefer to write their own implementation of the defined YANG model, they can use this feature to install the YANG models in the controller.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** Minimal set of CSIT test is already in place. More work is in progress and will have better functional coverage in Oxygen release.
  - https://jenkins.opendaylight.org/releng/view/Patch-Test/job/ovsdb-patch-test-l2gw-nitrogen/

odl-ovsdb-hwvtepsouthbound

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=ovsdb.git;a=blob;f=hwvtepsouthbound/hwvtepsouthbound-features/odl-ovsdb-hwvtepsouthbound/pom.xml;h=3bb0d9f0093d83d0a82b3b8edffc0acfc93ee93c;hb=refs/heads/stable/nitrogen
- **Feature Description:** This feature is the main feature of the OVSDB Hardware vTep Southbound plugin. This plugin handle the OVS device that supports the OVSDB Hardware vTEP schema and uses the OVSDB protocol. This feature provides the implementation of the defined YANG models. Developers developing the in-controller application and want to leverage OVSDB Hardware vTEP plugin for device configuration can add dependency on this feature and it will load all the required modules.
- **Top Level:** Yes

- **User Facing:** No
- **Experimental:** Yes
- **CSIT Test:** Minimal set of CSIT test is already in place. More work is in progress and will have better functional coverage in Oxygen release.
  - https://jenkins.opendaylight.org/releng/view/Patch-Test/job/ovsdb-patch-test-l2gw-nitrogen/

odl-ovsdb-hwvtepsouthbound-rest

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=ovsdb.git;a=blob;f=hwvtepsouthbound/hwvtepsouthbound-features/odl-ovsdb-hwvtepsouthbound-rest/pom.xml;h=8691103618cbe430994657016229b23c9b372d9d;hb=refs/heads/stable/nitrogen
- **Feature Description:** This feature is the wrapper feature that installs the odl-ovsdb-hwvtepsouthbound-api & odl-ovsdb-hwvtepsouthbound feature with other required features for restconf access to provide a functional OVSDB Hardware vTEP plugin. Users, who want to develop application that manages the hardware vTEP supported devices but want to runs the application outside of the OpenDaylight controller must install this feature.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** Minimal set of CSIT test is already in place. More work is in progress and will have better functional coverage in Oxygen release.
  - https://jenkins.opendaylight.org/releng/view/Patch-Test/job/ovsdb-patch-test-l2gw-nitrogen/

odl-ovsdb-library

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=ovsdb.git;a=blob;f=library/features/odl-ovsdb-library/pom.xml;h=58002499237ac290071a89ca5e0b9c9297974400;hb=refs/heads/stable/nitrogen
- **Feature Description:** Encode/decoder library for OVSDB and Hardware vTEP schema.
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/ovsdb/job/ovsdb-csit-1node-upstream-southbound-all-nitrogen/
  - https://jenkins.opendaylight.org/releng/view/ovsdb/job/ovsdb-csit-3node-upstream-clustering-only-nitrogen/

Documentation

- **User Guide(s):**
  - *OVSDB User Guide*
- **Developer Guide(s):**
  - *OVSDB Developer Guide*

Security Considerations

- Do you have any external interfaces other than RESTCONF? Yes, Southbound Connection to OVSDB/Hardware vTEP devices.
- Other security issues?

  Plugin's connection to device is by default unsecured. User need to explicitly enable the TLS support through ovsdb library configuration file. User can refer to the wiki page here for the instructions.

Quality Assurance

- Link to Sonar Report (57%)
- Link to CSIT Jobs
-
- OVSDB southbound plugin is extensively tested through Unit Tests, IT test and system tests. OVSDB southbound plugin is tested in both single node setup as well as three node cluster setup. Hardware vTEP plugin is currently tested through (1) Unit testing (2) CSIT Tests (3) NetVirt project L2 Gateway features CSIT tests and (4) Manual Testing. (3) https://jenkins.opendaylight.org/releng/view/netvirt/job/netvirt-csit-hwvtep-1node-openstack-newton-nodl-v2-upstream-stateful-nitrogen/

Migration

- Is it possible to migrate from the previous release? If so, how? Yes. User facing features and interfaces are not changed, only enhancements are done.

Compatibility

- Is this release compatible with the previous release? Yes
- Any API changes? No changes in the YANG models from previous release.
- Any configuration changes? No

Bugs Fixed

- List of bugs fixed since the previous release https://bugs.opendaylight.org/buglist.cgi?chfieldfrom=2017-05-25&chfieldto=2017-09-10&list_id=78767&product=ovsdb&query_format=advanced&resolution=FIXED

Known Issues

- List key known issues with workarounds None
- Link to Open Bugs

End-of-life

- List of features/APIs which are EOLed, deprecated, and/or removed in thisrelease

  None

Standards

- Open vSwitch Database Management Protocol
- OVSDB Schema
- Hardware vTep Schema

Release Mechanics

- Link to release plan

PacketCable

Major Features

odl-packetcable-policy-server

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=packetcable.git;a=blob;f=features-packetcable-policy/features4-packetcable-policy/pom.xml;h=0945b9287711a1ce9a7bd6cc0b457607a3cd6248;hb=refs/heads/stable/nitrogen
- **Feature Description:** Plugin that provides a PCMM model implementation based on CMTS structure and COPS protocol. It implements *RFC 2748 <https://tools.ietf.org/html/rfc2748>*.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/packetcable/job/packetcable-csit-1node-pcmm-all-nitrogen/

Documentation

- **User Guide(s):**
  - *PacketCable User Guide*
- **Developer Guide(s):**
  - *PacketCable Developer Guide*

Security Considerations

- Do you have any external interfaces other than RESTCONF? No.
- The PacketCable project interfaces to southbound devices using the COPS protocol. Securing communication on this interface is outslide the scope of this project.

Quality Assurance

- Link to Sonar Report (Test coverage percent - 53.41%)
- Link to CSIT Job:
- https://jenkins.opendaylight.org/releng/view/packetcable/job/packetcable-csit-1node-pcmm-all-nitrogen/
- Other manual testing and QA information - The CSIT job runs the PacketCable plugin in a simple cable access controller emulation environment. The code is manually tested during the development cycle with an actual (controller).
- Testing methodology. There is substantial unit testing executed in the project build process; CSIT testing is executed in an "emulated" cable access network environment. All product APIs are validated during the development cycle. CSIT testing would benefit from an upgrade to cover some of the post-Carbon feature additions.

Migration

- Is it possible to migrate from the previous release? Yes Migration from PacketCable Carbon version to the Nitrogen version is accomplished by replacement of the PacketCable plugin components.
- Any data stored in COPS models will need to be manually replicated.
- All previous API calls will work with the new release.

Compatibility

- Is this release compatible with the previous release? Yes
- Any API changes? No
- Any configuration changes? No

Bugs Fixed

- List of Bugzilla bugs fixed since the previous release NONE
- The only functional change for the Nitrogen release of Packetcable is the upgrade from Karaf3 to Karaf4.

  Known Issues

- There are no known issues with the Carbon release of PacketCable

End-of-life

- No PacketCable features or APIs are EOLed, deprecated, or removed in this release

Standards

- The Packetcable plug-in implements a subset of the provisioning operations defined in these specifications.
- CableLabs "PacketCable 1.5 Specification: MTA Device Provisioning" PKT-SP-PROV1.5-I04-090624
- COPS protocol RFC 2748 <https://tools.ietf.org/html/rfc2748>

Release Mechanics

- Link to Packetcable Nitrogen release plan: <https://wiki.opendaylight.org/view/PacketCablePCMM:Release_Plan_Nitrogen>
- There were no major shifts in release schedule from the release plan

Service Function Chaining

Major Features

odl-sfc-netconf

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** Provides functionality to communicate with netconf capable Service Functions.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

odl-sfc-scf-openflow

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** SFC stand-alone openflow classifier.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

odl-sfc-scf-vpp

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** SFC stand-alone vpp classifier.
- **Top Level:** Yes

- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

odl-sfc-openflow-renderer

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** Renderer functionality for OpenFlow capable switches.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

odl-sfclisp

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** Programs LISP capable switches.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

odl-sfc-sb-rest

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** Implements a South Bound Rest interface to send configuration to REST-capable switches.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

odl-sfc-ui

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** This feature is the SFC User Interface.

- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

odl-sfc-vnfm-tacker

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** Tacker VNF Manager interface.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

odl-sfc-ios-xe-renderer

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** Renderer functionality for IO XE switches that use netconf.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

odl-sfc-vpp-renderer

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** Renderer functionality for fd.io VPP (Vector Packet Processor) switches that use netconf.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

odl-sfc-pot

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** This feature implements a Proof of Transit for the Service Functions.

- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

These features are consumed by the User facing features above

odl-sfc-genius

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** This feature implements the Genius utilities created by SFC project.
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

odl-sfc-model

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** This feature defines and implements the SFC data model as specified here https://datatracker.ietf.org/doc/rfc7665/
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

odl-sfc-pot-netconf-renderer

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** This feature implements the Netconf rendering for the Proof of Transit for the Service Functions.
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

odl-sfc-provider

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** This feature provides an easy-to-use interface to the sfc-model.
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

odl-sfc-provider-rest

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** This feature provides no functionality, and just installs the necessary features for SFC restconf.
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

odl-sfc-ovs

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** This feature provides functionality for SFC to communicate with OVSDB for SFF configuration.
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

odl-sfc-test-consumer

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sfc.git;a=blob;f=features/src/main/features/features.xml
- **Feature Description:** This feature is used for testing only.
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sfc/job/sfc-csit-3node-clustering-all-carbon

Documentation

- **User Guide(s):**
  - *Service Function Chaining*
- **Developer Guide(s):**
  - *Service Function Chaining*

Security Considerations

None.

Quality Assurance

- Link to Sonar Report (57.1%)
- Link to CSIT Jobs
- All modules have been unit tested. Integration tests have been performed for all major features. System tests have been performed on most major features.

Migration

- Is it possible to migrate from the previous release? If so, how?

No changes were made to the SFC data model in this release, so no migration from the previous release is needed.

Compatibility

This release of SFC is completely compatible with the previous release.

Bugs Fixed

List of bugs fixed since the previous release

- 3712 Setting an SF on an SFP hop with an SF type different than the corresponding SFC hop should fail
- 7554 Update GUI after deprecating nsh-aware in SF and other changes in model
- 7555 SfcRenderingException for logicalSFF when SFs share a compute node
- 7629 Karaf 4 migration: provide Karaf 4 sfc features

Known Issues

SFC needs changes in OVS to include the Network Service Headers (NSH) Chaining encapsulation feature. This patch has been ongoing for quite a while, but has finally been officially merged in OVS 2.8. ODL will be updated to use this new version of OVS in the Oxygen release. Until then, SFC will use a branched version of OVS based on 2.6.1, called the "Yi Yang Patch", located here. Previous versions of this OVS patch only supported VXLAN-GPE + NSH encapsulation, but this version supports both ETH + NSH and VXLAN-GPE + ETH + NSH.

- Link to Open Bugs

End-of-life

- None

Standards

- List of standards implemented and to what extent
- IETF SFC RFC
- IETF NSH Only NSH Metadata type 1 is implemented.
- OpenFlow v1.3

Release Mechanics

- ODL SFC Carbon release plan
- No major shifts in the release schedule from the release plan

SNMP Plug-in

Major Features

odl-snmp-plugin

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=snmp.git;a=blob;f=features/features-snmp/src/main/features/features.xml;hb=stable/carbon
- **Feature Description:** Provides NB API to SB SNMP interface
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:**
  - https://jenkins.opendaylight.org/releng/view/snmp/job/snmp-csit-1node-basic-all-nitrogen/

Documentation

- **Getting Started:**
  - SNMP Plugin:Getting Started
- **User Guide:**
  - *SNMP Plugin User Guide*
- **SNMP Simulator:**
  - SNMP simulator guide

Security Considerations

- Do you have any external interfaces other than RESTCONF?

  Yes, this plugin provides SNMP endpoints for talking to southbound devices.

- Other security issues?

  Securing communication to devices (or not) over SNMP is outside the scope ofthis project and left to users.

Quality Assurance

- Link to Sonar Report (3.5% code coverage)
- Link to CSIT Jobs:
  - https://jenkins.opendaylight.org/releng/view/snmp/job/snmp-csit-1node-basic-all-nitrogen/
- Other manual testing and QA information: None

Migration

- Is it possible to migrate from the previous release? If so, how?

  It is possible to seamlessly migrate consumers to this iteration of the plug-in as there has been no functional change to features. Migration of state data is not defined.

Compatibility

Compatible with the previous release, no functional change to features

Bugs Fixed

- List of bugs fixed since the previous release

  None - no functional change to features

Known Issues

- List key known issues with workarounds

  No known issues

- Link to Open Bugs

End-of-life

- List of features/APIs which are EOLed, deprecated, and/or removed in this release

  None

Standards

- List of standards implemented and to what extent
  - SNMP

Release Mechanics

- Link to release plan
- Describe any major shifts in release schedule from the release plan

  None

SNMP4SDN

Major Features

odl-snmp4sdn-snmp4sdn

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=snmp4sdn.git;a=blob;f=features/odl-snmp4sdn-snmp4sdn/pom.xml;h=eece899425487cf81e81e3d87bff78a2f3d2797c;hb=HEAD
- **Feature Description:** This feature will install all bundles required for SNMP4SDN Plugin
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** NA

Documentation

- **User Guide:**
  - *SNMP4SDN User Guide*
- **Developer Guide(s):**
  - *SNMP4SDN Developer Guide*

Security Considerations

- The interface or configurable resource exposed to users includes RESTCONF API and the switch list file. Switch list file, which is a plain-text file, contains security information such as SNMP community.
- SNMP4SDN Plugin configures switches via SNMP protocol, and listens to SNMP listen port for link-up/down trap. SNMP v2c is used.

Quality Assurance

- Link to Sonar Report (Test coverage percent NA)
- Link to CSIT Jobs
- Other manual testing and QA information
- For each function of SNMP4SDN Plugin, use REST API to confirm it's availability and correctness. Existing functions includes flow configuration (such as VLAN and forwarding table) and topology discovery.

Migration

- Is it possible to migrate from the previous release? If so, how?

  No

Compatibility

- Is this release compatible with the previous release?

  Yes
- Any API changes?

  No
- Any configuration changes?

  No

Bugs Fixed

- None (no bugs reported since the previous release)

Known Issues

- List key known issues with workarounds

  None
- Link to Open Bugs

End-of-life

- List of features/APIs which are EOLed, deprecated, and/or removed in this release

  None

Standards

- List of standards implemented and to what extent

  None (no standards implemented, and use a third-party library to configure switches in standard SNMP protocol)

Release Mechanics

- Link to release plan
- No changes in this release

Scalable-Group Tag eXchange Protocol (SXP)

Major Features

odl-sxp-api

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sxp.git;a=blob;f=features/odl-sxp-api/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** This feature provides models based on RFC.
- **Top Level:** No
- **User Facing:** No
- **Experimental:** Yes
- **CSIT Test:** N/A

odl-sxp-core

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sxp.git;a=blob;f=features/odl-sxp-core/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** This feature performs tasks for managing SXP devices and provides the implementation of RFC.
- **Top Level:** No
- **User Facing:** No
- **Experimental:** Yes
- **CSIT Test:** N/A

odl-sxp-controller

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sxp.git;a=blob;f=features/odl-sxp-controller/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** This feature performs tasks regarding managing SXP devices via RESTCONF.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sxp/job/sxp-csit-1node-basic-all-nitrogen/
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sxp/job/sxp-csit-1node-filtering-all-nitrogen/
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sxp/job/sxp-csit-1node-topology-all-nitrogen/

- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sxp/job/sxp-csit-3node-periodic-clustering-all-nitrogen/

odl-sxp-robot

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sxp.git;a=blob;f=features/odl-sxp-robot/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** This is a sample feature used in CSIT testing.
- **Top Level:** No
- **User Facing:** No
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sxp/job/sxp-csit-1node-periodic-performance-all-nitrogen/

odl-sxp-routing

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=sxp.git;a=blob;f=features/odl-sxp-routing/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** This feature that performs managing of SXP devices in cluster environment.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/sxp/job/sxp-csit-3node-periodic-routing-all-nitrogen/

Documentation

- **Installation Guide(s):**
  - Installation Guide
- **User Guide(s):**
  - *SXP User Guide*
- **Developer Guide(s):**
  - *SXP Developer Guide*

Security Considerations

- Do you have any external interfaces other than RESTCONF?
  - Yes on port 64999 based on SXP RFC secured by TCP-MD5, optionally also with SSL.
- Other security issues?
  - TCP-MD5 security option is now deprecated, and in future will replaced by TCP-AO

Quality Assurance

- Link to Sonar Report (80%)
- Link to CSIT Jobs
  - CSIT Job basic
  - CSIT Job filtering
  - CSIT Job topology
  - CSIT Job clustering
  - CSIT Job performance
  - CSIT Job routing
- Other manual testing and QA information
  - N/A
- Testing methodology. How extensive was it? What should be expected to work? What hasn't been tested as much?
  - CSIT Test document 1
  - CSIT Test document 2
  - CSIT Test document 3

Migration

- Is it possible to migrate from the previous release? If so, how?
  - Yes, no data models were changed that would break the migration.

Compatibility

- Is this release compatible with the previous release?
  - Functionality is fully backwards compatible.
- Any API changes?
  - N/A
- Any configuration changes?
  - feature odl-sxp-route was renamed to odl-sxp-routing

Bugs Fixed

- List of bugs fixed since the previous release
  - Fixed BUGS

Known Issues

- List key known issues with workarounds
  - N/A
- Open Bugs

End-of-life

- List of features/APIs which are EOLed, deprecated, and/or removed in this release
  - N/A

Standards

- List of standards implemented and to what extent
  - SXP Fully implemented

Release Mechanics

- Release plan
- Describe any major shifts in release schedule from the release plan
  - N/A

Topology Processing Framework

Major Features

odl-topoprocessing-framework

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=topoprocessing.git;a=blob;f=features/odl-topoprocessing-framework/pom.xml;h=c1c7b89ddb42af81efbeb5ae444e3179b0a14533;hb=refs/heads/stable/nitrogen
- **Feature Description:** Topology processing core
- **Top Level:** No
- **User Facing:** No
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/topoprocessing/job/topoprocessing-csit-1node-topology-operations-all-nitrogen/

odl-topoprocessing-mlmt

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=topoprocessing.git;a=blob;f=features/odl-topoprocessing-mlmt/pom.xml;h=9fe3d505825f0f06dfcb166708b629d06855ec72;hb=refs/heads/stable/nitrogen
- **Feature Description:** Multi-Layer and Multi-Technology (MLMT) module
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/topoprocessing/job/topoprocessing-csit-1node-topology-operations-all-nitrogen/

odl-topoprocessing-network-topology

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=topoprocessing.git;a=blob;f=features/odl-topoprocessing-network-topology/pom.xml;h=0de34de8dd99de3ac4b0c0bc5908a1de24a8f7ea;hb=refs/heads/stable/nitrogen
- **Feature Description:** Support for network-topology model
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/topoprocessing/job/topoprocessing-csit-1node-topology-operations-all-nitrogen/

odl-topoprocessing-inventory

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=topoprocessing.git;a=blob;f=features/odl-topoprocessing-inventory/pom.xml;h=ef8b9c3b1cffc72bf871fd7168799ab797e05e5d;hb=refs/heads/stable/nitrogen
- **Feature Description:** Support for inventory model
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/topoprocessing/job/topoprocessing-csit-1node-topology-operations-all-nitrogen/

odl-topoprocessing-l2rs

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=topoprocessing.git;a=blob;f=features/odl-topoprocessing-i2rs/pom.xml;h=f6b747cadfebc92d6df58e84ed894ffd390d6768;hb=refs/heads/stable/nitrogen
- **Feature Description:** Support for i2rs model

- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/topoprocessing/job/topoprocessing-csit-1node-topology-operations-all-nitrogen/

odl-topoprocessing-inventory-rendering

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=topoprocessing.git;a=blob;f=features/odl-topoprocessing-inventory-rendering/pom.xml;h=cf278e2429f7ae048eac134a7c7b7f7095d4ba24;hb=refs/heads/stable/nitrogen
- **Feature Description:** Rendering demo
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/topoprocessing/job/topoprocessing-csit-1node-topology-operations-all-nitrogen/

Documentation

- **Developer Guide(s):**
  - *Topology Processing Framework Developer Guide*
  - Wiki

Security Considerations

- No external interfaces other then restconf
- No known security issues

Quality Assurance

- Link to Sonar Report (80%)
- Link to CSIT Jobs

Migration

- Not supported

Compatibility

- Compatible

Bugs Fixed

- Link to fixed bugs

Known Issues

- Leafs which are children (direct or indirect) of list can't be used as target fields
- Aggregation of termination points in case of more mappings works only if all underlay topologies are from the same model and only if that model is Network Topology or I2RS
- Aggregation of termination points in combination with aggregation of nodes doesn't work with inventory model
- Aggregation of termination points in combination with aggregation of nodes in case of more mappings works only if aggregation of termination points is specified on each underlay topology and only if model of all underlay topologies is the same
- Filtration of termination points in case of more filters works only if all underlay topologies are from the same model
- Maximum of one correlation per correlation item (aggregation may not work correctly in case of more correlation with the same correlation item)
- Link aggregation works only if user specify also link computation

MLMT limitations

- The mlmt module provides YANG models as based on: * network-topology YANG model version 2013-10-21 * TED YANG model version 2013-10-21
- The mlmt module works with underlay topologies based on: * network-topology YANG model version 2013-10-21 * isis-topology YANG model version 2013-10-21
- The mlmt module does not support underlay topologies based on ospf-topology YANG model 2013-10-21.

End-of-life

- Network Topology model is not supported for overlay topologies

Release Mechanics

- Link to release plan

Table Type Patterns

Major Features

odl-ttp-model

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=ttp.git;a=blob;f=features/features-ttp/src/main/features/features.xml;hb=stable/nitrogen
- **Feature Description:** Provides a YANG model for describing ONF TTP 1.0 Table Type Patterns (TTPs) in JSON as well as a database of TTPs and an augmentation adding supported and active TTPs on OpenFlow nodes.

- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** None.

TTP CLI Tools

- **Feature URL:** The Nitrogen executable jar can be found here: https://nexus.opendaylight.org/content/repositories/public/org/opendaylight/ttp/parser/0.5.0-Nitrogen/parser-0.5.0-Nitrogen-jar-with-dependencies.jar
- **Feature Description:** Provides stand-alone command line tools to validate and interact with TTPs in XML or JSON.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** None.

Documentation

Please provide the URL to each document at docs.opendaylight.org. If the document is under review, provide a link to the change in Gerrit.

- **User Guide(s):**
  - *TTP CLI Tools User Guide*
- **Developer Guide(s):**
  - *TTP CLI Tools Developer Guide*
  - *TTP Model Developer Guide*

Security Considerations

- Do you have any external interfaces other than RESTCONF?
  - No.
- Other security issues?
  - None.

Quality Assurance

- Link to Sonar Report (43.3% Test Coverage)
- No CSIT testing.
- There was minimal manual testing in Nitrogen, but also there were no changes beyond keeping up-to-date with changes in upstream projects. Unit tests cover the basics of the model.

Migration

- Is it possible to migrate from the previous release? If so, how?

  While it should be possible to export all TTP-related information by doing RESTCONF GETs and then import it by doing RESTCONF PUTs after the fact, this has not been tested and isn't officially supported.

Compatibility

- Is this release compatible with the previous release?

  Yes. There have been no code changes except to tolerate changes in upstream projects.

- Any API changes?

  No. No changes in models or APIs.

- Any configuration changes?

  No. The TTP project has no configuration.

Bugs Fixed

None fixed.

Known Issues

The TTP YANG model does not match the ONF TTP JSON precisely. Exact details are documented in the TTP model YANG file.

Open Bugs

End-of-life

None.

Standards

ONF TTP 1.0

Release Mechanics

- Nitrogen Table Type Patterns Release Plan
  - This was purely a maintenance release, so no changes were planned or happened.

Unimgr

Major Features

odl-unimgr

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=unimgr.git;a=blob;f=features/odl-unimgr/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** MEF Presto core infrastructure.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes

odl-unimgr-netvirt

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=unimgr.git;a=blob;f=features/odl-unimgr-netvirt/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** MEF Legato implementation using netvirt.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes

odl-unimgr-cisco-xr-driver

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=unimgr.git;a=blob;f=features/odl-unimgr-cisco-xr-driver/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** MEF presto implementation with cisco xr
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes

odl-unimgr-ovs-driver

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=unimgr.git;a=blob;f=features/odl-unimgr-ovs-driver/pom.xml;hb=refs/heads/stable/nitrogen
- **Feature Description:** MEF presto implementation with ovsdb
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes

Documentation

- **User Guide(s):**
  - *User Network Interface Manager Plug-in (Unimgr) User Guide*
- **Developer Guide(s):**
  - *User Network Interface Manager Plug-in (Unimgr) Developer Guide*

Security Considerations

No known security issues

Quality Assurance

- Link to Sonar Report (50.7% code coverage)
- Link to CSIT Jobs
- Tested Manually all main features.

  We have added unit and integration tests for Presto layer. Presto and Legato APIs were tested manually.

Migration

- Is it possible to migrate from the previous release? No, Current release is backward incompatible.

Compatibility

- Is this release compatible with the previous release?
- Any API changes?
- Any configuration changes?

Presto API replaces with completely new Presto NRP model

Bugs Fixed

- Only Bugs related to current release have been fixed

Known Issues

- No known issues

End-of-life

none

Standards

- MEF PRESTO API
- MEF LEGATO API

Release Mechanics

- Link to release plan

Unified Secure Channel

Major Features

- USC Agent provides proxy and agent functionality on top of all standard protocols supported by the device. It initiates call-home with the controller, maintains live connections with with the controller, acts as a de-muxer/muxer for packets with the USC header, and authenticates the controller.
- USC Plugin is responsible for communication between the controller and the USC agent . It responds to call-home with the controller, maintains live connections with the devices, acts as a muxer/demuxer for packets with the USC header, and provides support for TLS/DTLS.
- USC Manager handles configurations, high availability, security, monitoring, and clustering support for USC.
- USC UI is responsible for displaying a graphical user interface representing the state of USC in the OpenDaylight DLUX UI.

USC Channel UI

- **Feature Name:** odl-usc-channel-ui
- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=usc.git;a=blob;f=usc-features/odl-usc-channel-ui/pom.xml;
- **Feature Description:** Responsible for communication between the controller and the USC agent . It responds to call-home with the controller, maintains live connections with the devices, acts as muxer/demuxer for packets with the USC header, and provides support for TLS/DTLS.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/usc/job/usc-csit-1node-tcp-all-nitrogen/

Documentation

Please provide the URL to each document at docs.opendaylight.org. If the document is under review, provide a link to the change in Gerrit.

- **User Guide(s):**
  - *Unified Secure Channel*
- **Developer Guide(s):**

  - *Unified Secure Channel*

Security Considerations

- USC uses TLS and DTLS to secure the channels. Asymmetric authentication handshake when establishing the channels. Mutual authentication achieved with certificates configured in usc.properties for both the controller and the device.

Quality Assurance

- Link to Sonar Report
- Link to CSIT Jobs
- Link to Additional Details
- Code is covered by unit and integration tests
- System Tests are performed by CSIT jobs using java test agent.

Migration

- Nothing beyond general OpenDaylight migration requirements.

Compatibility

- Nothing beyond general OpenDaylight compatibility constraints.

Bugs Fixed

- Bug Report

Known Issues

- 3402 USC features has configuration issues with 3-node cluster environment.

End-of-life

- Nothing deprecated, EOL.

Standards

- N/A

Release Mechanics

- USC Release Plan
- Project was on schedule

Honeycomb Virtual Bridge Domain

Major Features

odl-vbd

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=honeycomb/vbd.git;a=blob;f=features/odl-vbd/src/main/feature/feature.xml;h=37a666153982e4efa38a37ca0b971be5d5cbdcd6;hb=refs/heads/stable/nitrogen
- **Feature Description:** This feature provides models to configure Virtual Bridge Domains on VPP.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** N/A

odl-vbd-ui

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=honeycomb/vbd.git;a=tree;f=features/odl-vbd-ui;h=a6172d7fb3d2c1930b0a87213b7043f58a711f60;hb=refs/heads/stable/nitrogen
- **Feature Description:** This feature provides the GUI for VBD.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** Yes
- **CSIT Test:** N/A

Documentation

- Wiki
- VBD API

Security Considerations

- N/A

Quality Assurance

- Sonar Report (0% - no coverage results available)
- VBD project is tested within FastDataStacks (FDS) testing suite, where severeal automated tests are performed. More information about FDS testing can be found here: FDS testing and test results are available here: FDS test results
- FDS automated tests perform series of functests where the whole stack is beeing tested (Openstack/ODL (GBP,VBD)/HC/VPP).

Migration

- Please use VPP 17.04 stable.

Compatibility

- Not compatible with previous VPP 17.01 or older stable versions.

Bugs Fixed

- N/A

Known Issues

- Due to yang updates for keeping VPP Rendering compatible with the latest stable for VPP, 17.04, we are not going to be compatible with previous stable VPPs. This, on the other hand, comes with a lot of augmentations of features.

End-of-life

- N/A

Standards

- N/A

Release Mechanics

- Release plan
- no major shifts from official release plan

VTN

Major Features

odl-vtn-manager-rest

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=vtn.git;a=blob;f=manager/features/odl-vtn-manager-rest/pom.xml;h=c130771e9dbc0d77dddf9b81a65d1a0c9aab936c;hb=refs/heads/stable/nitrogen#l24
- **Feature Description:** This is the feature that allows users to use the VTN virtualization, by creating the various components as needed for the network.
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/vtn/job/vtn-csit-1node-manager-all-nitrogen/

odl-vtn-manager-neutron

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=vtn.git;a=blob;f=manager/features/odl-vtn-manager-neutron/pom.xml;h=600411e41a52ddc8ac90e9a3c5c58b73ed774b8c;hb=refs/heads/stable/nitrogen#l24
- **Feature Description:** This feature provides support for integration with Openstack (L2 API)
- **Top Level:** Yes
- **User Facing:** Yes
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/vtn/job/vtn-csit-1node-openstack-newton-neutron-nitrogen/

Documentation

- **Installation Guide(s):**
  - *VTN Installation Guide*
- **User Guide(s):**
  - *VTN User Guide*
- **Developer Guide(s):**
  - *VTN Developer Guide*
  - *VTN Openstack Developer Guide*

Security Considerations

- No Issues.

Quality Assurance

- Link to Sonar Report (56.2%)
- Link to CSIT Jobs
- CSIT covers most of the options in RESTCONF
- The 3 node deployment has not been tested well.

Migration

- Not Supported.

Compatibility

- No Specific Compatibility issues.

Bugs Fixed

- 8761 - VTN coordinator Portmapping fails due to unnecessary hex conversion.
- 9024 - VTN Manager Set null to bundle version qualifier if it is empty.

Known Issues

- Link to Open Bugs

End-of-life

- None

Standards

- None

Release Mechanics

- Link to release plan
- There was no deviation from the plan.

YANG Tools

Major Features

Nitrogen release marks the seventh release of YANG Tools components. We have fixed lot of issues ranging from small annoyances to major reworks.

Major changes delivered in this release are

- Migration to new XML Parser Bug 5824 Bug 5825
- Fix of new XML parser design flaws Bug 8675 Bug 8715 Bug 8745.
- InMemoryDataTree can be configured to perform full mandatory leaf validation Bug 8291
- Deviation statements are properly activated Bug 8307
- TrieMap implementation Bug 7464
- Improvements, bug fixing and clean up yang-model-export, yang-parser-impl, yang-model-api, yang-data-impl, yang-data-api

odl-yangtools-yang-data

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=yangtools.git;a=blob_plain;f=features/odl-yangtools-yang-data/pom.xml;hb=refs/heads/v1.2.x
- **Feature Description:** to install YANG Data APIs and implementation.
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** Yes
- **CSIT Test:** System test waiver request pending.

odl-yangtools-common

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=yangtools.git;a=blob;f=features/odl-yangtools-common/pom.xml;hb=refs/heads/v1.2.x
- **Feature Description:** to install common concepts and utilities.
- **Top Level:** Yes
- **User Facing:** No
- **Experimental:** Yes
- **CSIT Test:** System test waiver request pending.

odl-yangtools-yang-parser

- **Feature URL:** https://git.opendaylight.org/gerrit/gitweb?p=yangtools.git;a=blob;f=features/odl-yangtools-yang-parser/pom.xml;hb=refs/heads/v1.2.x
- **Feature Description:** to install YANG model APIs and YANG Parser
- **Top Level:** Yes

- **User Facing:** No
- **Experimental:** No
- **CSIT Test:** https://jenkins.opendaylight.org/releng/view/yangtools/job/yangtools-csit-1node-periodic-system-only-nitrogen/

Documentation

- **Developer Guide(s):**
  - *YANG Tools Developer Guide*

Security Considerations

- YANG Tools libraries are designed to be embedded and not to be a stand-alone application so security concerns need to be addressed by the application using this library.

Quality Assurance

- Link to Sonar Report (Test coverage 75.4%)
- Link to CSIT Jobs

Migration

- It is possible to migrate from the previous release. Aside from adjusting to the removal of deprecated API elements and changed elements, there are no additional steps needed for migration to this release.

Compatibility

- Release is compatible with the previous one.
- API changes:
  - XmlParserStream.traverse (DOMSource)
  - Added UnrecognizedStatement and refactoring of StmtContextUtils
  - Moving of SubstatementValidator into spi.meta package
  - EffectiveSchemaContext was moved, users are advised to use SimpleSchemaContext
  - org.opendaylight.yangtools.yang.parser.spi package was adjusted
- No configuration changes.
- Behavior changes: * concepts.Registration does not allow nulls * SemVer and the associated Semantic Version is now bound to OpenConfig * YANG/YIN parser is less forgiving in face of ambiguous constructs

  (like 'mandatory ture' or 'status foobar')

Bugs Fixed

- List of fixed Bugs.

Known Issues

- Link to Open Bugs

End-of-life

- This release has not introduced any new deprecation of a major feature or API. However, there are some minor deprecations such as:
  - Yang-data-impl XML codec has been deprecated https://git.opendaylight.org/gerrit/#/c/60558/
  - YangParseException has been deprecated https://git.opendaylight.org/gerrit/#/c/58751/
- Major development is shifting to 2.0.0 version, which is guaranteed to be API incompatible. Users are advised to eliminate any reliance on @Deprecated components as soon as possible after adopting this release. Furthermore, the use of CheckedFuture will be eliminated, hence users should reduce their reliance on this class. Next release will also move Optional, Function, Predicate and others from Guava to their java.util equivalents.

Standards

- YANG and YIN parser processing according to RFC 6020, RFC 7950 and RFC 8040
- XML parser for YANG-modeled data according to RFC 6020 and RFC 7950.
- JSON parser for YANG-modeled data according to RFC 7951

Release Mechanics

- Link to the release plan

1.1.5 Service Release Notes

Nitrogen-SR1 Release Notes

This page details changes and bug fixes between the Nitrogen Release and the Nitrogen Stability Release 1 (Nitrogen-SR1) of OpenDaylight.

Projects with No Noteworthy Changes

- cardinal
- ocpplugin
- topoprocessing
- ttp
- yangtools

aaa

- 78fa45e AAA-151 : AAA-151: Invalidate claim cache for CLI initiated changes
- 4818c31 : Declare odl-aaa-encryption-service in artifacts
- 1c06dca : Bump odlparent 2.0.4 to 2.0.5
- 8bdc47a AAA-144 : Bug 9040: avoid using dynamicAuthorization for cluster-admin operations
- ce33d9c : Bump versions by x.y.(z+1)

alto

- 70804e0 : Bump odlparent 2.0.4 to 2.0.5
- ba54e2d : Bump versions by x.y.(z+1)

bgpcep

- fe5f547 BGPCEP-686 : BUG-9079 Make PCEP session recoverable from exception
- 76d764b : BUG-9192 / BUG-9191
- 7f6dea5 : BUG-9218: eliminate duplicate bundles
- 97c4803 BGPCEP-691 : Bug 9205: NPE received while receiving BGP peers
- f75843c BGPCEP-680 : BUG-8929: NPE during singleton startup
- e2095f8 BGPCEP-682 : Bug-8942: Fix DelegatedLspsCount
- 43c0193 BGPCEP-683 : BUG-8987: Print Exception when Css registration fails
- c2d5b09 BGPCEP-652 : BUG-8156 : conflicting listener fix
- 1944dec BGPCEP-652 : BUG-8156 : fixed start of session manager
- 3a5a210 : Bump odlparent 2.0.4 to 2.0.5
- 3f632b2 BGPCEP-652 : BUG-8156 : duplicate session up fixed
- 2c4a21a : Bump versions by x.y.(z+1)

bier

- 4ac680f : Cleanup bierapp-bundle dependencies
- cfff0b0 : Rework odl-bier-adapter feature
- 6b9970e : Bump odlparent 2.0.4 to 2.0.5
- f35ad9b : Do not depend on features-bgp
- f6a468c : Bump versions by x.y.(z+1)

controller

- 36b453e CONTROLLER-1771 : Bug 9165: Log config subsystem readiness as INFO
- 30a5b11 : Add debug to pinpoint lastApplied movement
- 0b7a749 : Lower verbosity in SimpletxDomRead
- 3a1a2b3 CONTROLLER-1713 : BUG-8639: always invalidate primary info cache
- 4883784 : Bump odlparent 2.0.4 to 2.0.5
- c570de3 CONTROLLER-1752 : Bug 9008: Fix the error of the persisted journal data format
- 8fd8a5c : Add an explicit null data check
- 72f3d16 : Fix testLeaderAndFollowerEntityOwnersReassignedAfterShutdown failure
- 5fdf80c : Fix intermitent testFollowerResyncWith*LeaderRestart failure
- 5ebbb1f : Bump versions by x.y.(z+1)

coe

- dc65935 : Bump odlparent 2.0.4 to 2.0.5
- cb18425 : Bump versions by x.y.(z+1)

daexim

- 8fceaa3 : Bump odlparent 2.0.4 to 2.0.5
- 895adf7 : Bump versions by x.y.(z+1)

dlux

- 1d176a1 : Bump odlparent 2.0.4 to 2.0.5
- 097139b : Bump versions by x.y.(z+1)

dluxapps

- 2b4104b : Bump odlparent 2.0.4 to 2.0.5
- 1b96094 : Bump versions by x.y.(z+1)

eman

- 0813520 : Bump odlparent 2.0.4 to 2.0.5
- 4ef702e : Bump versions by x.y.(z+1)

faas

- 32832e9 : Bump odlparent 2.0.4 to 2.0.5
- ac39ca5 : Bump versions by x.y.(z+1)

genius

- c05794e : Enable bound services update
- 3a79ac6 : This patch implements the genius mdsal interface for supporting conntrack ct_mark match (with mask) and action (without mask).
- 0bdc6f9 : Replacing DS read with cache read in ShowVlan CLI
- 716853f : Ignore newTunnelInterface() test as it is flaky
- 59c98f8 : Add support for mpls-gre tunnels
- e157a42 : Bump odlparent 2.0.4 to 2.0.5
- 1c03cd1 GENIUS-84 : Bug 8938 - Resource-batch manager enhancement
- f4ec27e : Use INFO for logging null DPID
- 54ab701 : bug 7380: Add getIfaceInfoFromConfigDataStore
- ac6a818 : Genius CSIT Failure : Missing Mandatory Node Error
- 20de66f : Clean up collections of Futures
- ebcf473 : Add gpe option to itm-config.xml
- 4f63446 GENIUS-89 : Bug 9099 - Suspected WriteOnlyTransaction leak in interfacemanager
- 5298a23 : Store original iface correctly in ISL worker
- 40950ce : Cleanup
- 20e07f4 : Cleanup
- 143480d : Remove start() method
- a622960 : Bump versions by x.y.(z+1)

groupbasedpolicy

- 9d9adaa : Bump odlparent 2.0.4 to 2.0.5
- 92941b6 : Bump versions by x.y.(z+1)

honeycomb/vbd

- 22361b9 : Bump odlparent 2.0.4 to 2.0.5
- d3813a8 : Bump versions by x.y.(z+1)

infrautils

- 71715ac : Bump odlparent 2.0.4 to 2.0.5
- 8777b96 : Bump versions by x.y.(z+1)

integration/distribution

- e301870 : Fix mdsal dependency in int/dist
- 87c31e0 INTDIST-92 : Bug 9189: Add missing version feature dependencies
- 1dfec4a : Revert "Bug 9307: Split features to avoid incompatiblities"
- 5d85bd0 : Bug 9307: Split features to avoid incompatiblities
- ab012b6 : Add odl-infrautils-ready to compatible with all
- eed1f25 : Move Cardinal features to not compatible
- f38a481 : Bug 9060: Add odl-mdsal-trace
- 0f428f1 : Add bier features to K4 distribution
- 108e314 : Bump odlparent 2.0.4 to 2.0.5
- 0c89576 : Bump versions by x.y.(z+1)

l2switch

- ab643dd : Bump odlparent 2.0.4 to 2.0.5
- 444f1c8 : Bump versions by x.y.(z+1)

lispflowmapping

- 99631c5 LISPMAP-166 : Bug 9127: Make IT more robust when receiving packets
- 92249f9 : Bump odlparent 2.0.4 to 2.0.5
- d5f6457 LISPMAP-169 : Bug 9172: Don't store subscribers with "No Address" source EID
- 3cc8437 LISPMAP-164 : Bug 9037: Fix positive overlapping negative
- d5c36eb LISPMAP-151 : Bug 7947: Move subscribers to a separate cache
- 5f69849 LISPMAP-151 : Revert "Bug 7947: Store MappingOrigin in MappingData"
- 02cff71 LISPMAP-163 : Bug 9023: Fix merging of negative prefixes
- 991c222 LISPMAP-160 : Bug 8746: Multi-threading improvements
- 2e860ec : Bump versions by x.y.(z+1)

mdsal

- a6433bb : Binding v2 generator - fix getting elements from empty array.
- 6ce7a04 MDSAL-291 : BUG-9145: rework singleton service group state tracking

- c61a8cc : Fix use of deprecated Futures.addCallback()
- 5d8111e : Binding v2 Generator - fix dependency yang-ext of mdsal-binding2-test-model - It should depend on yang-ext in binding2 model for using generated codes v2.
- e9c9a0b : Add yang-ext to model-binding2
- 7345aa3 : Fix revisions format in tests
- 952d570 : Bump odlparent 2.0.4 to 2.0.5
- 96a12ed : MDSAL Binding 2 Features for Karaf 4
- f1d7abd : Bump versions by x.y.(z+1)

nemo

- 21825a7 : Bump odlparent 2.0.4 to 2.0.5
- 92ad34c : Bump versions by x.y.(z+1)

netconf

- b5505bd SR-1 : Extra superfluous edit-config RPC sent - Netconf-482
- 1e8350d : Transition ListenerAdapter to ClusteredDOMDataTreeListener
- 34b91cb : Add unit tests for ListenerAdapter
- 96914ac NETCONF-475 : Bug 9256: Add websocket server config knob for ip
- 6978e89 NETCONF-465 : BUG 9112: NPE in karaf cli when device is still connecting
- 5c7ca5c : Remove aaa version declarations
- ccce725 : BUG-9218: fix features to not duplicate upstream bundles
- 9ae46c7 NETCONF-469 : BUG-9132: don't provide a value for restconf/streams/events
- 487cb8a : Bump odlparent 2.0.4 to 2.0.5
- 1552e67 : Bump versions by x.y.(z+1)
- 7b06550 : Minor cleanup of blueprint config
- 0434081 NETCONF-453 : Bug 8989 - Create just one DS for each test-tool's simulated netconf device

netvirt

- db4080e : Refactor/cleanup BgpConfigurationManager
- 73f2a21 NETVIRT-940 : Bug9245: Table=21 related exceptions fixes
- 60054d6 NETVIRT-926 : NETVIRT-926 - Maxpath value should be between 1 to 64 in BGP multipath
- c64a3ce NETVIRT-935 : Bug 9234: CSS programmed wrongly with TOR mac
- 3ffe9d5 : Refactor/cleanup BgpRouter
- 460a47f : Updated to use bind-service update instead of bind and unbind in Acl VPN listener
- eb221c6 : Handle usecase when ELAN is null, and ACL service BIND/ADD fails due to NPE

- 13aa527 NETVIRT-929 : BUG 9221: Improve logical SFF handling
- 87fa9a0 NETVIRT-928 : BUG 9220: don't use tun_gpe_np as match field
- 4504702 : Use nitrogen version for mdsal-trace
- 242f984 : Ignore addElanInterface
- f96ef54 NETVIRT-919 : Bug 9181: Code changes for conflicting modifications exceptions of table=19
- 2bce3ec NETVIRT-941 : Bug 9246: Conflicting modification from ARP and Router-GW-Mac
- 2e2208f NETVIRT-843 : Bug 8976 - Upstreaming fixes to master
- fca9cc2 NETVIRT-835 : Bug 8964 - Neutron test neutron.tests.tempest.scenario.test_floatingip.FloatingIpSameNetwork.test_east_ fails
- e34567f : Test SNAT mostSignificantBit()
- 98e9f1c NETVIRT-936 : Bug 9237 - NPE: InternalToExternalPortMapKey
- 70214f8 NETVIRT-931 : Bug 9226: VPN Traffic fails after VM Migration
- 0841ecc NETVIRT-918 : Bug 9180: Conflicting modification Exception from NAT Module
- e2d4059 NETVIRT-437 : BUG 7596 - Update to handle change in Neutron Network external attribute
- 5c48a5b : Remove unneeded mdsal and yangtools artifacts
- cfbcd8a : Remove unneeded pom version values
- 877a333 : Undo incorrect code changes made during merge conflict.
- f68d929 : Bump odlparent 2.0.4 to 2.0.5
- 3be0df9 NETVIRT-872 : Bug 9066:Use Single Transaction for DNAT Flow Install and Remove
- ee1a7ba : rm remaining it artifacts
- f8e7310 NETVIRT-875 : Bug 9077: Fix of issue that the existing NW communication failure when new NW is created
- 78a3153 : Lower log level for non errors
- 56bce23 NETVIRT-927 : Bug 9209: PNF learned on external networks to skip local FIB Processing
- 2235c8d : Renamed acl-impl.rst to acl-reflection-on-existing-traffic.rst
- 4092336 : sync cleanup
- fc11257 NETVIRT-923 : bug-9190: NullPointerException at getIsExternal
- 31a9df0 NETVIRT-853 : Bug 9012 : BGP not connecting to config server
- 3f3196e : Lower log level for non errors
- d24fc86 : Remove explicit default super-constructor calls
- 035fe7c : Bug9091 : Removing uncessary MD-SAL Read Operation in NAT
- 1416915 NETVIRT-829 : Bug 8953 - IllegalArgumentException: vrfEntry is missing mandatory descendant origin
- a1c58b0 : lower log levels for non-errors
- 9828258 : ClearBgpCli reads from socket to send/receive from bgpd (some previous commit modifed to read from session parameters)
- 3ce3792 : bgpmanager: change API of bgpmanager to add VRF IPv4 or IPv6

- 84a2457 : neutronvpn: create ipv4 or ipv6 context
- a0c5703 : bgpmanager thrift upgrade to 0.9.3
- d931c60 NETVIRT-834 : Bug 8963 - Option to configure EVPN address family
- 7380ac2 NETVIRT-926 : Bug 9196 - Maxpath value should be between 1 to 64 in BGP multipath
- 655de2e NETVIRT-821 : BUG 8930 - delete Op VPN interface when deleting external network
- a928467 : Lower log levels for non error's
- 397ea5d : elanmanager: clean up Futures collections
- 8a60da7 NETVIRT-924 : Bug 9193 - In conntrack SNAT , flows are programmed twice on a router g/w set.
- 1ae59879 : IfMgr clean-up
- 902d44b : Restrict NeutronvpnUtils.read
- 2ed1fcc NETVIRT-923 : bug-9190: NullPointerException at getIsExternal
- 5399653 NETVIRT-838 : BUG 8969 - Fix Exeption when clearing external router GW
- 3e15936 NETVIRT-888 : Bug 9105: close removeElanInterface transaction
- 6aebb4c : ElanUtils clean-up: ElanL2GatewayUtils
- 0f0ac42 : ElanUtils clean-up: L2GatewayConnectionUtils
- e1dae98 : ElanUtils clean-up: remove unnecessary references
- 021112d : ElanUtils clean-up: ElanL2Gateway{Multicast,}Utils
- da10b34 : ElanUtils clean-up: more ElanL2GatewayMulticastUtils
- 681fae2 : ElanUtils clean-up: ElanL2GatewayMulticastUtils
- 757d7ed : ElanUtils clean-up: make read() static
- cca12c8 : aclservice: clean up Futures collections
- d1d1f44 : dhcpservice: clean up Futures collections
- af7eebc : coe: clean up Futures collections
- 18d2f14 : Remove un-used SynchronousEachOperationNewWriteTransaction
- c4b2066 NETVIRT-829 : Bug 8953: Fix exceptions raised due to PNF confused with FIP
- 5e7933f : Remove aggregator from artifactId
- 266eacf : Dualstack support for L3VPN - single router Dual stack
- dab4df4 NETVIRT-864 : Bug 9030 - port and port security groups cannot be null
- 1967565 NETVIRT-862 : Bug 9026: ACL issue in handling port-create
- 8289943 : Add LogCaptureRule to AclServiceTestBase & ElanServiceTest
- 3091531 NETVIRT-367 : Bug 7380: service-binding exceptions from ACL
- 9d695bb : Remove learn mode from aclserivce.
- 03381dd : Remove transparent mode from aclservice.
- 61517e6 : Remove stateless mode from AclService.
- 1ad4d08 : Fix wrongly keyed network map in CoeUtils

- 9085dc4 NETVIRT-829 : Bug 8953 - IllegalArgumentException: vrfEntry is missing mandatory descendant origin.
- 22c1ba3 : Remove deprecated CheckedFuture
- 92cc55f : Minor code cleanup in QoS
- e449ed0 : Cleanup
- 0ab0ada : L2 Support for Pods
- 3ef5a81 NETVIRT-367 : Bug 7380: service-binding exceptions from ACL
- 34aec1f NETVIRT-789 : Bug 8860 : Populate elantag at time of elanInstance creation
- 06f600b : Spec for Acl change reflection on existing communication
- 09c4355 NETVIRT-835 : Bug 8964 - Neutron test neutron.tests.tempest.scenario.test_floatingip.FloatingIpSameNetwork.test_east fails
- 80865cb : Fix exception handling in neutronvpn shell
- 95ca2b1 : Bug7380:CSIT FIP ping is getting failed for Ext Flat/VLAN Network
- d93f513 : Fix cloud-servicechain YANG
- 5626383 : Remove unused references to DataChangeListener
- 8d62dc2 NETVIRT-899 : Bug 9136 - Suspected ReadOnlyTransaction leak in QosNeutronUtils
- 8d7f5a5 : fix coe nitrogen versions
- 38e0946 NETVIRT-884 : Bug 9100 : tx leak in DhcpExternalTunnelManager
- e0ea63d : Bug9016:Using Single Transaction during NAPT SwitchOver
- 51f7268 : Cluster support for netvirt QoS
- 0f285e1 NETVIRT-867 : Bug 9035: - NPE at org.opendaylight.netvirt.elan.arp.responder.ArpResponderUtil.getMatchCriteria
- 7a0ca73 : Bug:9013 ElanUtils: RPC Call to Get egress actions for interface, OptimisticLockFailedException
- a7f3b65 NETVIRT-879 : Bug 9085 - CSIT Sporadic failures - test_security_groups_basic_ops.TestSecurityGroupsBasicOps.test_c
- 39c01c7 : Replace size()==0 by isEmpty()
- ac9734f : Replace <? extends Object> by <?>
- ce8d9d6 NETVIRT-49 : Bug 6349: try connecting to qthrift only when configured. - default values are set to invalid host/port. - verify whether port/host configured before connecting
- 930d4bb NETVIRT-803 : Bug 8882 - With conntrack SNAT communication with PNF fails
- 3a0184b NETVIRT-885 : Bug 9102 Fix ReadOnlyTransaction leak in NeutronvpnUtils
- 484b600 NETVIRT-829 : Bug 8953 - IllegalArgumentException: vrfEntry is missing mandatory descendant origin
- 1a80e3e NETVIRT-864 : Bug 9030 - port and port security groups cannot be null
- fc82b17 : Bug 9060: Package mdsal trace utility in netvirt Karaf distribution
- c80eb76 : Bug 8801 - EVPN remote routes are not pushed to ovs flow table
- ea8b6aa NETVIRT-829 : Bug 8953 - IllegalArgumentException: vrfEntry is missing mandatory descendant origin
- b8cf946 : bgpmanager BgpUtil code clean-up: Make private what can, and rm unused
- cec0cc4 : Bug 9034: bgpmanager BgpUtil rm unused pendingWrTransaction

- bce2347 NETVIRT-789 : Bug 8860: NPE in getElanTag from SubnetmapChangeListener
- 87a9334 NETVIRT-870 : Bug 9051 - Failed to handle router GW flow in GW-MAC table. DPN id is missing for router-id
- be46ddf : Initial Bundle setup for coe renderer
- f59e001 : Bump versions by x.y.(z+1)

neutron

- 3f31de7 : BUG-9218: make hostconfig plugins depend on spi
- 1e2ee7a : Bump odlparent 2.0.4 to 2.0.5
- f73d8e5 : Bump versions by x.y.(z+1)

nic

- b518195 : Bump odlparent 2.0.4 to 2.0.5
- b02e175 : Bump versions by x.y.(z+1)

of-config

- 664f48e : Bump odlparent 2.0.4 to 2.0.5
- 48fcc2b : Bump versions by x.y.(z+1)

openflowplugin

- 36fcca7 OPNFLWPLUG-930 : OPNFLWPLUG-930 Inconsistent flow IDs between flows in config and operational data stores
- ac07bed : Remove deprecated
- 450b1a1 : Do not use fix custom version in ofp feature
- 4d5c7af OPNFLWPLUG-898 : Fix checkstyle warnings for impl/device package
- 2ab36c6 OPNFLWPLUG-898 : Fix checkstyle warnings for impl/connection package and OpenFlowPluginProviderImpl
- e6e6412 OPNFLWPLUG-898 : Fix codestyle
- 739d901 OPNFLWPLUG-898 : Fix codestyle
- c02495e : Cli to display all the connected DPNs
- 1168029 : Fix log message
- 7cdb645 OPNFLWPLUG-898 : Remove unsupported statistics warning
- 172e48f OPNFLWPLUG-950 : BUG-9223:Remove hardcoded value of lldp interval
- c4b0b4a : This patch implements ct-mark support in nicira extensions.
- da11ae9 OPNFLWPLUG-898 : Fix checkstyle warnings for impl/karaf, lifecycle, common, mastership
- 067b512 OPNFLWPLUG-898 : Fix checkstyle warnings for util package

- afc011e : Fix issues related to checkstyle enforcement
- e93494e : Fix issues related to checkstyle enforcement
- c07d277 : Fix issues related to checkstyle enforcement
- eb2d654 : BUG8607 Fix checkstyle issues
- e1b26b8 OPNFLWPLUG-898 : Fix checkstyle warnings for rpc package
- 563558a OPNFLWPLUG-898 : Fix checkstyle warnings for impl/protocol package
- c2d91ef OPNFLWPLUG-898 : Fix checkstyle warnings for services package
- 1829a63 OPNFLWPLUG-898 : Fix checkstyle warnings for translator and registry package
- 4a724f3 : Bump odlparent 2.0.4 to 2.0.5
- 8064a4f OPNFLWPLUG-948 : Sort bucket actions
- da13c64 : Bump versions by x.y.(z+1)
- b14867f : Fix issues related to checkstyle enforcement
- e6acc16 OPNFLWPLUG-898 : Fix codestyle
- 187291a : Add missing bundle converters
- 5e9b83f OPNFLWPLUG-938 : Do not mark device as connecting when closing it
- b23364d OPNFLWPLUG-926 : Redesign statistics context and manager
- ee9c2d0 OPNFLWPLUG-898 : Fix checkstyle warnings for impl/datastore package
- 294cce8 OPNFLWPLUG-898 : Fix checkstyle warnings for impl/protocol test package
- 1b1888c OPNFLWPLUG-898 : Fix checkstyle warnings for impl/role package

ovsdb

- 512179a OVSDB-396 : bug 7599 avoid unnecessary mdsal reads
- cf70b38 : bug 8712 vlan bindings update fix
- 451e720 OVSDB-421 : Bug 8874 - Tunnel_ips of hardware_vtep is cleared when Open vSwitch process restarted in Open vSwitch HWVTEP Emulator
- 035e3d9 OVSDB-406 : bug 8029 added ref counts for physical locators.
- 122a37c OVSDB-429 : BUG 9166 - Fix Netvirt L2GW Illegal state exception
- c91ad95 : Refactor compareDbVersionToMinVersion
- acd89a1 OVSDB-422 : Bug 8991 - Add dpdkvhostuserclient interface type to model
- 5a7dd9e : Convert DataChangeListeners to DataTreeChangeListeners
- 7fe1aed : Remove explicit default super-constructor calls
- 5f6dbf4 : Bump odlparent 2.0.4 to 2.0.5
- 9558f56 : Bump versions by x.y.(z+1)

packetcable

- bb6a3c5 : Bump odlparent 2.0.4 to 2.0.5
- e673c9e : Bump versions by x.y.(z+1)

sfc

- aafbc35 SFC-204 : BUG 9305: Unbind SFC service when removing SFs
- c47795d : Add SFC shell's command to show Service Nodes
- 6fa05ff : Add an API to handle ServiceNode entities
- aa14e3f : Add SFC shell's command to show Service Function Types
- f13c04a : BUG-9218: Fix odl-sfc-shell dependencies
- b065fa6 : Make utility classes final and other minor changes
- 39c400e : Add SFC shell's command to show Service Function Chains
- 8b227b4 : Add SFC shell's command to show Service Function Paths
- 83d5063 : Bump odlparent 2.0.4 to 2.0.5
- 42ca744 : Bump versions by x.y.(z+1)

snmp

- c57a6b0 : Bump odlparent 2.0.4 to 2.0.5
- 0014df2 : Bump versions by x.y.(z+1)

snmp4sdn

- 589029c : Bump odlparent 2.0.4 to 2.0.5
- 8ced555 : Bump versions by x.y.(z+1)

sxp

- c02a4b2 SXP-130 : SXP-130 Delete entire node from Operational DS
- 252efec SXP-126 : BUG-9255 Fix race conditions in md5update
- 6ccebfe : Fix feature dependencies of sxp-api
- f6200f5 : Bump odlparent 2.0.4 to 2.0.5
- 59636d4 : Bump versions by x.y.(z+1)
- f8c9fd8 SXP-125 : BUG-9126 Bump jrobot remote server
- 9ec4264 SXP-124 : BUG-9062 - generate positive retry times

unimgr

- fa88027 : Bump odlparent 2.0.4 to 2.0.5
- ef2b85b : Bump versions by x.y.(z+1)

usc

- d1c9e44 : Bump odlparent 2.0.4 to 2.0.5
- aa220fd : Bump versions by x.y.(z+1)

vtn

- 59c5fc2 VTN-166 : Bug 9224 - Fix for mapping issue of protocol and dscp values
- 4d5551e VTN-165 : Bug 9208: Fixed UDP L4 match details creation failures
- b570e1e VTN-167 : Bug 9225: Upgrade Apache Tomcat for VTN coordinator to 7.0.82.
- 46730f7 VTN-167 : Bug 9225: Upgrade Apache Tomcat for VTN coordinator to 7.0.81.
- a6abc3f : Bump odlparent 2.0.4 to 2.0.5
- f23614f : Bump versions by x.y.(z+1)
- 52dd810 VTN-164 : Bug 9174: Fix for VTN Coordinator Flowlistentry Creation failure

Nitrogen-sr2 Release Notes

This page details changes and bug fixes between the Nitrogen Stability Release -1 (Nitrogen-SR-1) and the Nitrogen Stability Release sr2 (nitrogen-sr2) of OpenDaylight.

Projects with No Noteworthy Changes

- aaa
- alto
- bgpcep
- bier
- cardinal
- controller
- coe
- daexim
- dlux
- dluxapps
- eman
- faas
- genius

- groupbasedpolicy
- honeycomb/vbd
- infrautils
- integration/distribution
- l2switch
- lispflowmapping
- mdsal
- nemo
- netconf
- netvirt
- neutron
- nic
- ocpplugin
- of-config
- openflowplugin
- ovsdb
- packetcable
- sfc
- snmp
- snmp4sdn
- sxp
- topoprocessing
- ttp
- unimgr
- usc
- vtn
- yangtools

1.2 Getting Started Guide

1.2.1 Introduction

The OpenDaylight project is an open source platform for Software Defined Networking (SDN) that uses open protocols to provide centralized, programmatic control and network device monitoring. Like many other SDN controllers, OpenDaylight supports OpenFlow, as well as offering ready-to-install network solutions as part of its platform.

Much as your operating system provides an interface for the devices that comprise your computer, OpenDaylight provides an interface that allows you to connect network devices quickly and intelligently for optimal network performance.

It's extremely helpful to understand that setting up your networking environment with OpenDaylight is not a single software installation. While your first chronological step is to install OpenDaylight, you install additional functionality packaged as Karaf features to suit your specific needs.

Before walking you through the initial OpenDaylight installation, this guide presents a fuller picture of OpenDaylight's framework and functionality so you understand how to set up your networking environment. The guide then takes you through the installation process.

What's different about OpenDaylight

Major distinctions of OpenDaylight's SDN compared to traditional SDN options are the following:

- A microservices architecture, in which a "microservice" is a particular protocol or service that a user wants to enable within their installation of the OpenDaylight controller, for example:
  - A plugin that provides connectivity to devices via the OpenFlow or BGP protocols
  - An L2-Switch or a service such as Authentication, Authorization, and Accounting (AAA).
- Support for a wide and growing range of network protocols beyond OpenFlow, including SNMP, NETCONF, OVSDB, BGP, PCEP, LISP, and more.
- Support for developing new functionality comprised of additional networking protocols and services.

Note: A thorough understanding of the microservices architecture is important for experienced network developers who want to create new solutions in OpenDaylight. If you are new to networking and OpenDaylight, you most likely won't design solutions, but you should comprehend the microservices concept to understand how OpenDaylight works and how it differs from other SDN programs.

What you'll find in this guide

To set up your environment, you first install OpenDaylight followed by the Apache Karaf features that offer the functionality you require. The OpenDaylight Getting Started Guide covers feature descriptions, OpenDaylight installation procedures, and feature installation.

The Getting Started Guide also includes other helpful information, with the following organization:

1. An overview of OpenDaylight and common use models
2. Who should use this guide?
3. OpenDaylight concepts and tools
4. Explanations of OpenDaylight Apache Karaf features and other features that extend network functionality
5. OpenDaylight system requirements and Release Notes
6. OpenDaylight installation instructions
7. Feature tables with installation names and compatibility notes

1.2.2 Overview

OpenDaylight performs the following functions:

- Logically centralizes programmatic control of the physical and virtual devices in your network.
- Controls devices with standard, open protocols.

- Provides higher-level abstractions of its capabilities so experienced network engineers and developers can create new applications to customize network setup and administration.

Common use cases for SDN are as follows:

1. Centralized network monitoring, management, and orchestration
2. Proactive network management and traffic engineering
3. Chaining packets through the different VMs, which is known as service function chaining (SFC). SFC enables Network Functions Virtualization (NFV), which is a network architecture concept that virtualizes entire classes of network node functions into building blocks that may connect, or chain together, to create communication services.
4. Cloud - managing both the virtual overlay and the physical underlay beneath it.

1.2.3 Who should use this guide?

OpenDaylight is for users considering open options in network programming. This guide provides information for the following types of users:

1. Those new to OpenDaylight who want to install it and select the features they need to run their network environment using only the command line and GUI. Such users include:
   (a) Students
   (b) Network administrators and engineers.
2. Network engineers and network application developers who want to use OpenDaylight's REST APIs to manage their network programmatically.
3. Network engineers and network application developers who want to write their own OpenDaylight services and plugins for greater functionality. This group of users needs a significant level of expertise in the following areas, which is beyond the scope of this document:
   (a) The YANG modeling language
   (b) The Model-Driven Service Abstraction Layer (MD-SAL)
   (c) Maven build tool
   (d) Management of the shared data store
   (e) How to handle notifications and/or Remote Procedure Calls (RPCs)
4. Developers who would like to join the OpenDaylight community and contribute code upstream. People in this group design offerings such as applications/services, protocol implementations, and so on, to increase OpenDaylight functionality for the benefit of all end-users.

Note: If you develop code to build new functionality for OpenDaylight and push it upstream (not required), it can become part of the OpenDaylight release. Users can then install the features to implement the solution you've created.

1.2.4 OpenDaylight concepts and tools

In this section we discuss some of the concepts and tools you encounter with basic use of OpenDaylight. The guide walks you through the installation process in a subsequent section, but for now familiarize yourself with the information below.

- To date, OpenDaylight developers have formed more than 50 projects to address ways to extend network functionality. The projects are a formal structure for developers from the community to meet, document release plans, code, and release the functionality they create in an OpenDaylight release.

  The typical OpenDaylight user will not join a project team, but you should know what projects are as we refer to their activities and the functionality they create. The Karaf features to install that functionality often share the project team's name.

- **Apache Karaf** provides a lightweight runtime to install the Karaf features you want to implement and is included in the OpenDaylight platform software. By default, OpenDaylight has no pre-installed features.

- After installing OpenDaylight, you install your selected features using the Karaf console to expand networking capabilities. In the Karaf feature list below are the ones you're most likely to use when creating your network environment.

  As a short example of installing a Karaf feature, OpenDaylight offers Application Layer Traffic Optimization (ALTO). The Karaf feature to install ALTO is odl-alto-all. On the Karaf console, the command to install it is:

  feature:install odl-alto-all

- **DLUX** is a web-based interface that OpenDaylight provides for you to manage your network. Its Karaf feature installation name is "odl-dlux-core".

  1. DLUX draws information from OpenDaylight's topology and host databases to display the following information:

     (a) The network

     (b) Flow statistics

     (c) Host locations

  2. To enable the DLUX UI after installing OpenDaylight, run:

     feature:install odl-dlux-core

     on the Karaf console.

- **Network embedded Experience (NeXt)** is a developer toolkit that provides tools to draw network-centric topology UI elements that offer visualizations of the following:

  1. Large complex network topologies
  2. Aggregated network nodes
  3. Traffic/path/tunnel/group visualizations
  4. Different layout algorithms
  5. Map overlays
  6. Preset user-friendly interactions

  NeXt can work with DLUX to build OpenDaylight applications. Check out the NeXt_demo for more information on the interface.

- Model-Driven Service Abstraction Layer (MD-SAL) is the OpenDaylight framework that allows developers to create new Karaf features in the form of services and protocol drivers and connects them to one another. You can think of the MD-SAL as having the following two components:

  1. A shared datastore that maintains the following tree-based structures:

  1. The Config Datastore, which maintains a representation of the desired network state.
  2. The Operational Datastore, which is a representation of the actual network state based on data from the managed network elements.

2. A message bus that provides a way for the various services and protocol drivers to notify and communicate with one another.

- If you're interacting with OpenDaylight through DLUX or the REST APIs while using the the OpenDaylight interfaces, the microservices architecture allows you to select available services, protocols, and REST APIs.

1.2.5 OpenDaylight Karaf Features

This section provides brief descriptions of the most commonly used Karaf features developed by OpenDaylight project teams. They are presented in alphabetical order. OpenDaylight installation instructions and a feature table that lists installation commands and compatibility follow.

- *AAA*
- *ALTO*
- *Border Gateway Protocol (including Link-state Distribution (BGP)*
- *Border Gateway Monitoring Protocol (BMP)*
- *Control and Provisioning of Wireless Access Points (CAPWAP)*
- *Controller Shield*
- *Device Identification and Driver Management (DIDM)*
- *DLUX*
- *Fabric as a Service (FaaS)*
- *Group Based Policy (GBP)*
- *Internet of Things Data Management (IoTDM)*
- *Link Aggregation Control Protocol (LACP)*
- *Location Identifier Separation Protocol (LISP) Flow Mapping Service (LISP)*
- *NEMO*
- *NETCONF*
- *NetIDE*
- *OVSDB-based Network Virtualization Services*
- *OpenFlow Configuration Protocol (OF-CONFIG)*
- *OpenFlow plugin*
- *Path Computation Element Protocol (PCEP)*
- *Secure Network Bootstrapping Interface (SNBi)*
- *Service Function Chaining (SFC)*
- *SNMP Plugin*
- *SNMP4SDN*
- *Source-Group Tag Exchange Protocol (SXP)*
- *Topology Processing Framework*
- *Time Series Data Repository (TSDR)*

- *Unified Secure Channel (USC)*
- *Virtual Tenant Network (VTN)*

AAA

Standards-compliant Authentication, Authorization and Accounting Services. RESTCONF is the most common consumer of AAA, which installs the AAA features automatically. AAA provides:

- Support for persistent data stores
- Federation and SSO with OpenStack Keystone

This release of AAA includes experimental support for having the database of users and credentials stored in the cluster-aware MD-SAL datastore.

ALTO

Implements the Application-Layer Traffic Optimization (ALTO) base IETF protocol to provide network information to applications. It defines abstractions and services to enable simplified network views and network services to guide application usage of network resources and includes five services:

1. Network Map Service - Provides batch information to ALTO clients in the forms of ALTO network maps.
2. Cost Map Service - Provides costs between defined groupings.
3. Filtered Map Service - Allows ALTO clients to query an ALTO server on ALTO network maps and/or cost maps based on additional parameters.
4. Endpoint Property Service - Allows ALTO clients to look up properties for individual endpoints.
5. Endpoint Cost Service - Allows an ALTO server to return costs directly amongst endpoints.

Border Gateway Protocol (including Link-state Distribution (BGP)

Is a southbound plugin that provides support for Border Gateway Protocol (including Link-state Distribution) as a source of L3 topology information.

Border Gateway Monitoring Protocol (BMP)

Is a southbound plugin that provides support for BGP Monitoring Protocol as a monitoring station.

Control and Provisioning of Wireless Access Points (CAPWAP)

Enables OpenDaylight to manage CAPWAP-compliant wireless termination point (WTP) network devices. Intelligent applications, e.g., radio planning, can be developed by tapping into the operational states made available via REST APIs of WTP network devices.

Controller Shield

Creates a repository called the Unified-Security Plugin (USecPlugin) to provide controller security information to northbound applications, such as the following:

- Collating the source of different attacks reported in southbound plugins

- Gathering information on suspected controller intrusions and trusted controllers in the network

Information collected at the plugin may also be used to configure firewalls and create IP blacklists for the network.

Device Identification and Driver Management (DIDM)

Provides device-specific functionality, which means that code enabling a feature understands the capability and limitations of the device it runs on. For example, configuring VLANs and adjusting FlowMods are features, and there may be different implementations for different device types. Device-specific functionality is implemented as Device Drivers.

DLUX

Web based OpenDaylight user interface that includes:

- An MD-SAL flow viewer
- Network topology visualizer
- A tool box and YANG model that execute queries and visualize the YANG tree

Fabric as a Service (FaaS)

Creates a common abstraction layer on top of a physical network so northbound APIs or services can be more easily mapped onto the physical network as a concrete device configuration.

Group Based Policy (GBP)

Defines an application-centric policy model for OpenDaylight that separates information about application connectivity requirements from information about the underlying details of the network infrastructure. Provides support for:

- Integration with OpenStack Neutron
- Service Function Chaining
- OFOverlay support for NAT, table offsets

Internet of Things Data Management (IoTDM)

Developing a data-centric middleware to act as a oneM2M-compliant IoT Data Broker (IoTDB) and enable authorized applications to retrieve IoT data uploaded by any device.

Link Aggregation Control Protocol (LACP)

LACP can auto-discover and aggregate multiple links between an OpenDaylight-controlled network and LACP-enabled endpoints or switches.

Location Identifier Separation Protocol (LISP) Flow Mapping Service (LISP)

LISP (RFC6830) enables separation of Endpoint Identity (EID) from Routing Location (RLOC) by defining an overlay in the EID space, which is mapped to the underlying network in the RLOC space.

LISP Mapping Service provides the EID-to-RLOC mapping information, including forwarding policy (load balancing, traffic engineering, and so on) to LISP routers for tunneling and forwarding purposes. The LISP Mapping Service can serve the mapping data to data plane nodes as well as to OpenDaylight applications.

To leverage this service, a northbound API allows OpenDaylight applications and services to define the mappings and policies in the LISP Mapping Service. A southbound LISP plugin enables LISP data plane devices to interact with OpenDaylight via the LISP protocol.

NEMO

Is a Domain Specific Language (DSL) for the abstraction of network models and identification of operation patterns. NEMO enables network users/applications to describe their demands for network resources, services, and logical operations in an intuitive way that can be explained and executed by a language engine.

NETCONF

Offers four features:

- odl-netconf-mdsal: NETCONF Northbound for MD-SAL and applications
- odl-netconf-connector: NETCONF Southbound plugin - configured through the configuration subsystem
- odl-netconf-topology: NETCONF Southbound plugin - configured through the MD-SAL configuration datastore
- odl-restconf: RESTCONF Northbound for MD-SAL and applications

NetIDE

Enables portability and cooperation inside a single network by using a client/server multi-controller architecture. It provides an interoperability layer allowing SDN Applications written for other SDN Controllers to run on OpenDaylight. NetIDE details:

- Architecture follows a client/server model: other SDN controllers represent clients with OpenDaylight acting as the server.
- OpenFlow v1.0/v1.3 is the only southbound protocol supported in this initial release. We are planning for other southbound protocols in later releases.
- The developer documentation contains the protocol specifications required for developing plugins for other client SDN controllers.
- The NetIDE Configuration file contains the configurable elements for the engine.

OVSDB-based Network Virtualization Services

Several services and plugins in OpenDaylight work together to provide simplified integration with the OpenStack Neutron framework. These services enable OpenStack to offload network processing to OpenDaylight while enabling OpenDaylight to provide enhanced network services to OpenStack.

OVSDB Services are at parity with the Neutron Reference Implementation in OpenStack, including support for:

- L2/L3
  - The OpenDaylight Layer-3 Distributed Virtual Router is fully on par with what OpenStack offers and now provides completely decentralized Layer 3 routing for OpenStack. ICMP rules for responding on behalf of the L3 router are fully distributed as well.
  - Full support for distributed Layer-2 switching and distributed IPv4 routing is now available.
- Clustering - Full support for clustering and High Availability (HA) is available in the this OpenDaylight release. In particular, the OVSDB southbound plugin supports clustering that any application can use, and the Openstack network integration with OpenDaylight (through OVSDB Net-Virt) has full clustering support. While there is no specific limit on cluster size, a 3-node cluster has been tested extensively as part of the release.
- Security Groups - Security Group support is available and implemented using OpenFlow rules that provide superior functionality and performance over OpenStack Security Groups, which use IPTables. Security Groups also provide support for ConnTrack with stateful tracking of existing connections. Contract-based Security Groups require OVS v2.5 with contract support.
- Hardware Virtual Tunnel End Point (HW-VTEP) - Full HW-VTEP schema support has been implemented in the OVSDB protocol driver. Support for HW-VTEP via OpenStack through the OVSDB-NetVirt implementation has not yet been provided as we wait for full support of Layer-2 Gateway (L2GW) to be implemented within OpenStack.
- Service Function Chaining
- Open vSwitch southbound support for quality of service and Queue configuration Load Balancer as service (LBaaS) with Distributed Virtual Router
- Network Virtualization User interface for DLUX

OpenFlow Configuration Protocol (OF-CONFIG)

Provides a process for an Operation Context containing an OpenFlow Switch that uses OF-CONFIG to communicate with an OpenFlow Configuration Point, enabling remote configuration of OpenFlow datapaths.

OpenFlow plugin

Supports connecting to OpenFlow-enabled network devices via the OpenFlow specification. It currently supports OpenFlow versions 1.0 and 1.3.2.

In addition to support for the core OpenFlow specification, OpenDaylight also includes preliminary support for the Table Type Patterns and OF-CONFIG specifications.

Path Computation Element Protocol (PCEP)

Is a southbound plugin that provides support for performing Create, Read, Update, and Delete (CRUD) operations on Multiprotocol Label Switching (MPLS) tunnels in the underlying network.

Secure Network Bootstrapping Interface (SNBi)

Leverages manufacturer-installed IEEE 802.1AR certificates to secure initial communications for a zero-touch approach to bootstrapping using Docker. SNBi devices and controllers automatically do the following:

1. Discover each other, which includes:
   (a) Revealing the physical topology of the network

(b) Exposing each type of a device

(c) Assigning the domain for each device

2. Get assigned an IP-address
3. Establish secure IP connectivity

SNBi creates a basic infrastructure to host, run, and lifecycle-manage multiple network functions within a network device, including individual network element services, such as:

- Performance measurement
- Traffic-sniffing functionality
- Traffic transformation functionality

SNBi also provides a Linux side abstraction layer to forward elements as well as enhancements to feature the abstraction and bootstrapping infrastructure. You can also use the device type and domain information to initiate controller federation processes.

Service Function Chaining (SFC)

Provides the ability to define an ordered list of network services (e.g. firewalls, load balancers) that are then "stitched" together in the network to create a service chain. SFC provides the chaining logic and APIs necessary for OpenDaylight to provision a service chain in the network and an end-user application for defining such chains. It includes:

- YANG models to express service function chains
- SFC receiver for Intent expressions from REST & RPC
- UI for service chain construction
- LISP support
- Function grouping for load balancing
- OpenFlow renderer for Network Service Headers, MPLS, and VLAN
- Southbound REST interface
- IP Tables-based classifier for grouping packets into selected service chains
- Integration with OpenDaylight GBP project
- Integration with OpenDaylight OVSDB NetVirt project

SNMP Plugin

The SNMP southbound plugin allows applications acting as an SNMP Manager to interact with devices that support an SNMP agent. The SNMP plugin implements a general SNMP implementation, which differs from the SNMP4SDN as that project leverages only select SNMP features to implement the specific use case of making an SNMP-enabled device emulate some features of an OpenFlow-enabled device.

SNMP4SDN

Provides a southbound SNMP plugin to optimize delivery of SDN controller benefits to traditional/legacy ethernet switches through the SNMP interface. It offers support for flow configuration on ACLs and enables flow configuration via REST API and multi-vendor support.

Source-Group Tag Exchange Protocol (SXP)

Enables creation of a tag that allows you to filter traffic instead of using protocol-specific information like addresses and ports. Via SXP an external entity creates the tags, assigns them to traffic appropriately, and publishes information about the tags to network devices so they can enforce the tags appropriately.

More specifically, SXP Is an IETF-published control protocol designed to propagate the binding between an IP address and a source group, which has a unique source group tag (SGT). Within the SXP protocol, source groups with common network policies are endpoints connecting to the network. SXP updates the firewall with SGTs, enabling the firewalls to create topology-independent Access Control Lists (ACLs) and provide ACL automation.

SXP source groups have the same meaning as endpoint groups in OpenDaylight's Group Based Policy (GBP), which is used to manipulate policy groups, so you can use OpenDaylight GPB with SXP SGTs. The SXP topology-independent policy definition and automation can be extended through OpenDaylight for other services and networking devices.

Topology Processing Framework

Provides a framework for simplified aggregation and topology data query to enable a unified topology view, including multi-protocol, Underlay, and Overlay resources.

Time Series Data Repository (TSDR)

Creates a framework for collecting, storing, querying, and maintaining time series data in OpenDaylight. You can leverage various data-driven applications built on top of TSDR when you install a datastore and at least one collector.

Functionality of TDSR includes:

- Data Query Service - For external data-driven applications to query data from TSDR through REST APIs
- ElasticSearch - Use external elastic search engine with TSDR integrated support.
- NBI integration with Grafana - Allows visualization of data collected in TSDR using Grafana
- Data Aggregation Service - Periodically aggregates raw data into larger time granularities
- Data Purging Service - Periodically purges data from TSDR
- Data Collection Framework - Data Collection framework to allow plugging in of various types of collectors
- HSQL data store - Replacement of H2 data store to remove third party component dependency from TSDR
- Cassandra data store - Cassandra implementation of TSDR SPIs
- NetFlow data collector - Collect NetFlow data from network elements
- NetFlowV9 - version 9 Netflow collector
- sFlowCollector - Collects sFlow data from network elements
- SNMP Data Collector - Integrates with SNMP plugin to bring SNMP data into TSDR
- Syslog data collector - Collects syslog data from network elements
- Web Activity data collector - Collects ODL RESTCONF queries made to TSDR

TSDR has multiple features to enable the functionality above. To begin, select one of these data stores:

- odl-tsdr-hsqldb-all
- odl-tsdr-hbase
- odl-tsdr-cassandra

Then select any "collectors" you want to use:

- odl-tsdr-openflow-statistics-collector
- odl-tsdr-netflow-statistics-collector
- odl-tsdr-sflow-statistics-collector
- odl-tsdr-controller-metrics-collector
- odl-tsdr-snmp-data-collector
- odl-tsdr-syslog-collector
- odl-tsdr-restconf-collector

Enable ElasticSearch support:

- odl-tsdr-elasticsearch

See these TSDR_Directions for more information.

Unified Secure Channel (USC)

Provides a central server to coordinate encrypted communications between endpoints. Its client-side agent informs the controller about its encryption capabilities and can be instructed to encrypt select flows based on business policies.

A possible use case is encrypting controller-to-controller communications; however, the framework is very flexible, and client side software is available for multiple platforms and device types, enabling USC and OpenDaylight to centralize the coordination of encryption across a wide array of endpoint and device types.

Virtual Tenant Network (VTN)

Provides multi-tenant virtual network on an SDN controller, allowing you to define the network with a look and feel of a conventional L2/L3 network. Once the network is designed on VTN, it automatically maps into the underlying physical network and is then configured on the individual switch, leveraging the SDN control protocol.

By defining a logical plane with VTN, you can conceal the complexity of the underlying network and better manage network resources to reduce network configuration time and errors.

1.2.6 OpenDaylight Experimental Features

Network Intent Composition (NIC)

Offers an interface with an abstraction layer for you to communicate "intentions," i.e., what you expect from the network. The Intent model, which is part of NIC's core architecture, describes your networking services requirements and transforms the details of the desired state to OpenDaylight. NIC has four features:

- odl-nic-core-hazelcast: Provides the following:

  - A distributed intent mapping service implemented using hazelcast, which stores metadata needed to process Intent correctly
  - An intent REST API to external applications for Create, Read, Update, and Delete (CRUD) operations on intents, conflict resolution, and event handling
- odl-nic-core-mdsal: Provides the following:
  - A distributed Intent mapping service implemented using MD-SAL, which stores metadata needed to process Intent correctly
  - An Intent rest API to external applications for CRUD operations on Intents, conflict resolution, and event handling
- odl-nic-console: Provides a Karaf CLI extension for Intent CRUD operations and mapping service operations
- Four renderers to provide specific implementations to render the Intent:
  - Virtual Tenant Network Renderer
  - Group Based Policy Renderer
  - OpenFlow Renderer
  - Network MOdeling Renderer

UNI Manager Plug-in (Unimgr)

Formed to initiate the development of data models and APIs that facilitate OpenDaylight software applications' and/or service orchestrators' ability to configure and provision connectivity services.

YANG-PUBSUB

An experimental feature Plugin that allows subscriptions to be placed on targeted subtrees of YANG datastores residing on remote devices. Changes in YANG objects within the remote subtree can be pushed to OpenDaylight as specified and don't require OpenDaylight to make continuous fetch requests. YANG-PUBSUB is developed as a Java project. Development requires Maven version 3.1.1 or later.

1.2.7 Other features

OpFlex

Provides the OpenDaylight OpFlex Agent , which is a policy agent that works with Open vSwitch (OVS), to enforce network policy, e.g., from Group-Based Policy, for locally-attached virtual machines or containers.

Network embedded Experience (NeXt)

Provides a network-centric topology UI that offers visualizations of the following:

1. Large complex network topologies
2. Aggregated network nodes
3. Traffic/path/tunnel/group visualizations
4. Different layout algorithms
5. Map overlays

6. Preset user-friendly interactions

NeXt can work with DLUX to build OpenDaylight applications. NeXt does not support Internet Explorer. Check out the NeXt_demo for more information on the interface.

1.2.8 API

We are in the process of creating automatically generated API documentation for all of OpenDaylight. The following are links to the preliminary documentation that you can reference. We will continue to add more API documentation as it becomes available.

- mdsal
- odlparent
- yangtools

1.2.9 Installing OpenDaylight

You complete the following steps to install your networking environment, with specific instructions provided in the subsections below.

Before detailing the instructions for these, we address the following: Java Runtime Environment (JRE) and operating system information Target environment Known issues and limitations

Install OpenDaylight

Downloading and installing OpenDaylight

The default distribution can be found on the OpenDaylight software download page: http://www.opendaylight.org/software/downloads

The Karaf distribution has no features enabled by default. However, all of the features are available to be installed.

Note: For compatibility reasons, you cannot enable all the features simultaneously. We try to document known incompatibilities in the *Install the Karaf features* section below.

Running the karaf distribution

To run the Karaf distribution:

1. Unzip the zip file.
2. Navigate to the directory.
3. run `./bin/karaf`.

For Example:

```
$ ls distribution-karaf-0.7.x-Nitrogen.zip
distribution-karaf-0.7.x-Nitrogen.zip
$ unzip distribution-karaf-0.7.x-Nitrogen.zip
Archive:  distribution-karaf-0.7.x-Nitrogen.zip
   creating: distribution-karaf-0.7.x-Nitrogen/
```

```
   creating: distribution-karaf-0.7.x-Nitrogen/configuration/
   creating: distribution-karaf-0.7.x-Nitrogen/data/
   creating: distribution-karaf-0.7.x-Nitrogen/data/tmp/
   creating: distribution-karaf-0.7.x-Nitrogen/deploy/
   creating: distribution-karaf-0.7.x-Nitrogen/etc/
   creating: distribution-karaf-0.7.x-Nitrogen/externalapps/
   ...
   inflating: distribution-karaf-0.7.x-Nitrogen/bin/start.bat
   inflating: distribution-karaf-0.7.x-Nitrogen/bin/status.bat
   inflating: distribution-karaf-0.7.x-Nitrogen/bin/stop.bat
$ cd distribution-karaf-0.7.x-Nitrogen
$ ./bin/karaf

    ________                       ________                .__  .__       .__     __
    \_____  \ ______   ____   ____ \_____  \ ______  ___.__.\|  \| \|__\| ____ \|  \|__
↪_/  \|_
     /   \|   \\____ \_/ __ \ /    \ \|    \|  \\__  \<   \|  \|\|  \| \|  \|/ ___\\|␣
↪ \|  \   __\
    /    \|    \  \|_> >  ___/\|   \|  \\|    `   \/ __ \\___  \|\|  \|_\|  / /_/  > ␣
↪ Y  \  \|
    \_______  /   __/ \___  >___\|  /_______  (____  / ____\|\|____/__\___  /\|___\| ␣
↪/__\|
            \/\|__\|         \/     \/         \/     \/\/            /_____/      \/
```

- Press `tab` for a list of available commands
- Typing `[cmd] --help` will show help for a specific command.
- Press `ctrl-d` or type `system:shutdown` or `logout` to shutdown OpenDaylight.

Note: Please take a look at the *Deployment Recommendations* and following sections under *Security Considerations* if you're planning on running OpenDaylight outside of an isolated test lab environment.

Install the Karaf features

To install a feature, use the following command, where feature1 is the feature name listed in the table below:

```
feature:install <feature1>
```

You can install multiple features using the following command:

```
feature:install <feature1> <feature2> ... <featureN-name>
```

Note: For compatibility reasons, you cannot enable all Karaf features simultaneously. The table below documents feature installation names and known incompatibilities.Compatibility values indicate the following:

- **all** - the feature can be run with other features.
- **self+all** - the feature can be installed with other features with a value of **all**, but may interact badly with other features that have a value of **self+all**. Not every combination has been tested.

Uninstalling features

To uninstall a feature, you must shut down OpenDaylight, delete the data directory, and start OpenDaylight up again.

Important: Uninstalling a feature using the Karaf feature:uninstall command is not supported and can cause unexpected and undesirable behavior.

Listing available features

To find the complete list of Karaf features, run the following command:

```
feature:list
```

To list the installed Karaf features, run the following command:

```
feature:list -i
```

Features to implement networking functionality provide release notes, which you can find in the *Project-specific Release Notes* section.

Karaf running on Windows 10

Windows 10 cannot be identify by Karaf (equinox). Issue occurs during installation of karaf features e.g.:

```
opendaylight-user@root>feature:install odl-restconf
Error executing command: Can't install feature odl-restconf/0.0.0:
Could not start bundle mvn:org.fusesource.leveldbjni/leveldbjni-all/1.8-odl in␣
↪feature(s) odl-akka-leveldb-0.7: The bundle "org.fusesource.leveldbjni.leveldbjni-
↪all_1.8.0 [300]" could not be resolved. Reason: No match found for native code:␣
↪META-INF/native/windows32/leveldbjni.dll; processor=x86; osname=Win32, META-INF/
↪native/windows64/leveldbjni.dll; processor=x86-64; osname=Win32, META-INF/native/
↪osx/libleveldbjni.jnilib; processor=x86; osname=macosx, META-INF/native/osx/
↪libleveldbjni.jnilib; processor=x86-64; osname=macosx, META-INF/native/linux32/
↪libleveldbjni.so; processor=x86; osname=Linux, META-INF/native/linux64/
↪libleveldbjni.so; processor=x86-64; osname=Linux, META-INF/native/sunos64/amd64/
↪libleveldbjni.so; processor=x86-64; osname=SunOS, META-INF/native/sunos64/sparcv9/
↪libleveldbjni.so; processor=sparcv9; osname=SunOS
```

Workaround is to add

    org.osgi.framework.os.name = Win32

to the karaf file

    etc/system.properties

The workaround and further info are in this thread: http://stackoverflow.com/questions/35679852/karaf-exception-is-thrown-while-installing-org-fusesource-leveldbjni

Karaf OpenDaylight Features

Table 1.1: Karaf OpenDaylight features

Feature Name	Feature Description	Karaf feature name	Compatibility
Authentication	Enables authentication with support for federation using Apache Shiro	odl-aaa-shiro	all
BGP	Provides support for Border Gateway Protocol (including Link-State Distribution) as a source of L3 topology information	odl-bgpcep-bgp	all
BMP	Provides support for BGP Monitoring Protocol as a monitoring station	odl-bgpcep-bmp	all
DIDM	Device Identification and Driver Management	odl-didm-all	all
Centinel	Provides interfaces for streaming analytics	odl-centinel-all	all
DLUX	Provides an intuitive graphical user interface for OpenDaylight	odl-dluxapps-applications	all
Fabric as a Service (Faas)	Creates a common abstraction layer on top of a physical network so northbound APIs or services can be more easiliy mapped onto the physical network as a concrete device configuration	odl-faas-all	all
Group Based Policy	Enables Endpoint Registry and Policy Repository REST APIs and associated functionality for Group Based Policy with the default renderer for OpenFlow renderers	odl-groupbasedpolicy-ofoverlay	all
GBP User Interface	Enables a web-based user interface for Group Based Policy	odl-groupbasedpolicyi-ui	all
GBP FaaS renderer	Enables the Fabric as a Service renderer for Group Based Policy	odl-groupbasedpolicy-faas	self+all
GBP Neutron Support	Provides OpenStack Neutron support using Group Based Policy	odl-groupbasedpolicy-neutronmapper	all
L2 Switch	Provides L2 (Ethernet) forwarding across connected OpenFlow switches and support for host tracking	odl-l2switch-switch-ui	self+all

Continued on next page

Table 1.1 – continued from previous page

Feature Name	Feature Description	Karaf feature name	Compatibility
LACP	Enables support for the Link Aggregation Control Protocol	odl-lacp-ui	self+all
LISP Flow Mapping	Enables LISP control plane services including the mapping system services REST API and LISP protocol SB plugin	odl-lispflowmapping-msmr	all
NEMO CLI	Provides intent mappings and implementation with CLI for legacy devices	odl-nemo-cli-renderer	all
NEMO OpenFlow	Provides intent mapping and implementation for OpenFlow devices	odl-nemo-openflow-renderer	self+all
NetIDE	Enables portabilty and cooperation inside a single network by using a client/server multi-controller architecture	odl-netide-rest	all
NETCONF over SSH	Provides support to manage NETCONF-enabled devices over SSH	odl-netconf-connector-ssh	all
OF-CONFIG	Enables remote configuration of OpenFlow datapaths	odl-of-config-rest	all
OVSDB OpenStack Neutron	OpenStack Network Virtualization using OpenDaylight's OVSDB support	odl-ovsdb-openstack	all
OVSDB Southbound	OVSDB MDSAL southbound plugin for Open_vSwitch schema	odl-ovsdb-southbound-impl-ui	all
OVSDB HWVTEP Southbound	OVSDB MDSAL hwvtep southbound plugin for the hw_vtep schema	odl-ovsdb-hwvtepsouthbound-ui	all
OVSDB NetVirt SFC	OVSDB NetVirt support for SFC	odl-ovsdb-sfc-ui	all
OpenFlow Flow Programming	Enables discovery and control of OpenFlow switches and the topoology between them	odl-openflowplugin-flow-services-ui	all
OpenFlow Table Type Patterns	Allows OpenFlow Table Type Patterns to be manually associated with network elements	odl-ttp-all	all
Packetcable PCMM	Enables flow-based dynamic QoS management of CMTS use in the DOCSIS infrastructure and a policy server	odl-packetcable-policy-server	self+all

Continued on next page

Table 1.1 – continued from previous page

Feature Name	Feature Description	Karaf feature name	Compatibility
PCEP	Enables support for PCEP	odl-bgpcep-pcep	all
RESTCONF API Support	Enables REST API access to the MD-SAL including the data store	odl-restconf	all
SDNinterface	Provides support for interaction and sharing of state between (non-clustered) OpenDaylight instances	odl-sdninterfaceapp-all	all
SFC over L2	Supports implementing Service Function Chaining using Layer 2 forwarding	odl-sfcofl2	self+all
SFC over LISP	Supports implementing Service Function Chaining using LISP	odl-sfclisp	all
SFC over REST	Supports implementing Service Function Chaining using REST CRUD operations on network elements	odl-sfc-sb-rest	all
SFC over VXLAN	Supports implementing Service Function Chaining using VXLAN tunnels	odl-sfc-ovs	self+all
SNMP Plugin	Enables monitoring and control of network elements via SNMP	odl-snmp-plugin	all
SNMP4SDN	Enables OpenFlow-like control of network elements via SNMP	odl-snmp4sdn-all	all
SSSD Federated Authentication	Enables support for federated authentication using SSSD	odl-aaa-sssd-plugin	all
Secure tag eXchange Protocol (SXP)	Enables distribution of shared tags to network devices	odl-sxp-controller	all

Continued on next page

Table 1.1 – continued from previous page

Feature Name	Feature Description	Karaf feature name	Compatibility
Time Series Data Repository (TSDR)	Enables support for collecting, storing and querying time series data. TSDR supports the following collection data: • OpenFlow statistics • NETFLOW statistics • sFlow statistics • OpenFlow Controller metrics • SNMP data • SysLog data • RestConf data TSDR supports the following data stores: • HSQLDB • HBase • Cassandra TSDR supports the default OpenDaylight RESTCONF and API interfaces and an ElasticSearch interface for all data stores.	odl-tsdr-core, odl-tsdr-hsqldb	all
TSDR Data Collectors	TSDR collector features include support for collecting the following data: • OpenFlow statistics • NETFLOW statistics • sFlow statistics • OpenFlow Controller metrics • SNMP data • SysLog data • RESTCONF data.	• odl-tsdr-openflow-statistics-collector • odl-tsdr-netflow-statistics-collector • odl-tsdr-sflow-statistics-collector • odl-tsdr-controller-metrics-collector • odl-tsdr-snmp-data-collector • odl-tsdr-syslog-collector • odl-tsdr-restconf-collector	all
TSDR Data Stores	TSDR enables support for the following data stores: * HSQLDB * HBase * Cassandra	• odl-tsdr-hsqldb • odl-tsdr-hbase • odl-tsdr-cassandra	all

Continued on next page

Table 1.1 – continued from previous page

Feature Name	Feature Description	Karaf feature name	Compatibility
TSDR Data Query	TSDR supports the default OpenDaylight RESTCONF and ODL API interfaces for queries to all data stores. It also supports an integrated ElasticSearch query.	odl-tsdr-elasticsearch	all
Topology Processing Framework	Enables merged and filtered views of network topologies	odl-topoprocessing-framework	all
Unified Secure Channel (USC)	Enables support for secure, remote connections to network devices	odl-usc-channel-ui	all
VTN Manager	Enables Virtual Tenant Network support	odl-vtn-manager-rest	self+all
VTN Manager Neutron	Enables OpenStack Neutron support of VTN Manager	odl-vtn-manager-neutron	self+all

Other OpenDaylight features

Table 1.2: Other OpenDaylight features

Feature Name	Feature Description	Karaf feature name	Compatibility
OpFlex	Provides OpFlex agent for Open vSwitch to enforce network policy, such as GBP, for locally-attached virtual machines or containers	n/a	all
NeXt	Provides a developer toolkit for designing network-centric topology user interfaces	n/a	all

Experimental OpenDaylight Features

The following functionality is labeled as experimental in this OpenDaylight release and should be used accordingly. In general, it is not supposed to be used in production unless its limitations are well understood by those deploying it.

Table 1.3: Other features

Feature Name	Feature Description	Karaf feature name	Compatibility
Authorization	Enables configurable role-based authorization	odl-aaa-authz	all
ALTO	Enables support for Application-Layer Traffic Optimization	odl-alto-core	self+all
CAPWAP	Enables control of supported wireless APs	odl-capwap-ac-rest	all
Clustered Authentication	Enables the use of the MD-SAL clustered data store for the authentication database	odl-aaa-authn-mdsal-cluster	all
Controller Shield	Provides controller security information to northbound applications	odl-usecplugin	all
GBP IO Visor Renderer	Provides support for rendering Group Based Policy to IO Visor	odl-groupbasedpolicy-iovisor	all
Internet of Things Data Management	Enables support for the oneM2M specification	odl-iotdm-onem2m	all
LISP Flow Mapping OpenStack Network Virtualization	Experimental support for OpenStack Neutron virtualization	odl-lispflowmapping-neutron	self+all
Network Intent Composition (NIC)	Provides abstraction layer for communcating network intents (including a distributed intent mapping service REST API) using either Hazelcast or the MD-SAL as the backing data store for intents	odl-nic-core-hazelcast or odl-nic-core-mdsal	all
NIC Console	Provides a Karaf CLI extension for intent CRUD operations and mapping service operations	odl-nic-console	all
NIC VTN renderer	Virtual Tenant Network renderer for Network Intent Composition	odl-nic-renderer-vtn	self+all
NIC GBP renderer	Group Based Policy renderer for Network Intent Composition	odl-nic-renderer-gbp	self+all
NIC OpenFlow renderer	OpenFlow renderer for Network Intent Composition	odl-nic-renderer-of	self+all
NIC NEMO renderer	NEtwork MOdeling renderer for Network Intent Composition	odl-nic-renderer-nemo	self+all
OVSDB NetVirt UI	OVSDB DLUX UI	odl-ovsdb-ui	all
Secure Networking Bootstrap	Defines a SNBi domain and associated white lists of devices to be accommodated to the domain	odl-snbi-all	self+all
UNI Manager	Initiates the development of data models and APIs to facilitate configuration and provisioning connectivity services for OpenDaylight applications and services	odl-unimgr	all
YANG PUBSUB	Allows subscriptions to be placed on targeted subtrees of YANG datastores residing on remote devices to obviate the need for OpenDaylight to make continuous fetch requests	odl-yangpush-rest	all

Install support for REST APIs

Most components that offer REST APIs will automatically load the RESTCONF API Support component, but if for whatever reason they seem to be missing, install the "odl-restconf" feature to activate this support.

1.2.10 Project-Specific Installation Guides

Centinel Installation Guide

This document is for the user to install the artifacts that are needed for using Centinel functionality in the OpenDaylight by enabling the default Centinel feature. Centinel is a distributed reliable framework for collection, aggregation and analysis of streaming data which is added in this OpenDaylight release.

Overview

The Centinel project aims at providing a distributed, reliable framework for efficiently collecting, aggregating and sinking streaming data across Persistence DB and stream analyzers (e.g., Graylog, Elasticsearch, Spark, Hive). This framework enables SDN applications/services to receive events from multiple streaming sources (e.g., Syslog, Thrift, Avro, AMQP, Log4j, HTTP/REST).

In this release, we develop a "Log Service" and plug-in for log analyzer (e.g., Graylog). The Log service process real time events coming from log analyzer. Additionally, we provide stream collector (Flume- and Sqoop-based) that collects logs from OpenDaylight and sinks them to persistence service (integrated with TSDR). Centinel also includes a RESTCONF interface to inject events to north bound applications for real-time analytic/network configuration. Further, a Centinel User Interface (web interface) will be available to operators to enable rules/alerts/dashboard etc.

Pre Requisites for Installing Centinel

- Recent Linux distribution - 64bit/16GB RAM
- Java Virtual Machine 1.7 or above
- Apache Maven 3.1.1 or above

Preparing for Installation

There are some additional pre-requisites for Centinel, which can be done by integrate Graylog server, Apache Drill, Apache Flume and HBase.

Graylog server2 Installation

- Install MongoDB
  - import the MongoDB public GPG key into apt:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
```

  - Create the MongoDB source list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen
↪' | sudo tee /etc/apt/sources.list.d/mongodb.list
```

  - Update your apt package database:

```
sudo apt-get update
```

  - Install the latest stable version of MongoDB with this command:

```
sudo apt-get install mongodb-org
```

- Install Elasticsearch
  - Graylog2 v0.20.2 requires Elasticsearch v.0.90.10. Download and install it with these commands:

```
cd ~; wget https://download.elasticsearch.org/elasticsearch/elasticsearch/
↪elasticsearch-0.90.10.deb
sudo dpkg -i elasticsearch-0.90.10.deb
```

  - We need to change the Elasticsearch cluster.name setting. Open the Elasticsearch configuration file:

```
sudo vi /etc/elasticsearch/elasticsearch.yml
```

  - Find the section that specifies cluster.name. Uncomment it, and replace the default value with graylog2:

```
cluster.name: graylog2
```

  - Find the line that specifies network.bind_host and uncomment it so it looks like this:

```
network.bind_host: localhost
script.disable_dynamic: true
```

  - Save and quit. Next, restart Elasticsearch to put our changes into effect:

```
sudo service elasticsearch restart
```

  - After a few seconds, run the following to test that Elasticsearch is running properly:

```
curl -XGET 'http://localhost:9200/_cluster/health?pretty=true'
```

- Install Graylog2 server
  - Download the Graylog2 archive to /opt with this command:

```
cd /opt; sudo wget https://github.com/Graylog2/graylog2-server/releases/
↪download/0.20.2/graylog2-server-0.20.2.tgz
```

  - Then extract the archive:

```
sudo tar xvf graylog2-server-0.20.2.tgz
```

  - Let's create a symbolic link to the newly created directory, to simplify the directory name:

```
sudo ln -s graylog2-server-0.20.2 graylog2-server
```

  - Copy the example configuration file to the proper location, in /etc:

```
sudo cp /opt/graylog2-server/graylog2.conf.example /etc/graylog2.conf
```

  - Install pwgen, which we will use to generate password secret keys:

```
sudo apt-get install pwgen
```

  - Now must configure the admin password and secret key. The password secret key is configured in graylog2.conf, by the password_secret parameter. Generate a random key and insert it into the Graylog2 configuration with the following two commands:

```
SECRET=$(pwgen -s 96 1)
sudo -E sed -i -e 's/password_secret =.*/password_secret = '$SECRET'/' /etc/
↪graylog2.conf

PASSWORD=$(echo -n password | shasum -a 256 | awk '{print $1}')
sudo -E sed -i -e 's/root_password_sha2 =.*/root_password_sha2 = '$PASSWORD'/
↪' /etc/graylog2.conf
```

- Open the Graylog2 configuration to make a few changes: (sudo vi /etc/graylog2.conf):

```
rest_transport_uri = http://127.0.0.1:12900/
elasticsearch_shards = 1
```

- Now let's install the Graylog2 init script. Copy graylog2ctl to /etc/init.d:

```
sudo cp /opt/graylog2-server/bin/graylog2ctl /etc/init.d/graylog2
```

- Update the startup script to put the Graylog2 logs in /var/log and to look for the Graylog2 server JAR file in /opt/graylog2-server by running the two following sed commands:

```
sudo sed -i -e 's/GRAYLOG2_SERVER_JAR=\${GRAYLOG2_SERVER_JAR:=graylog2-server.
↪jar}/GRAYLOG2_SERVER_JAR=\${GRAYLOG2_SERVER_JAR:=\/opt\/graylog2-server\/
↪graylog2-server.jar}/' /etc/init.d/graylog2
sudo sed -i -e 's/LOG_FILE=\${LOG_FILE:=log\/graylog2-server.log}/LOG_FILE=\$
↪{LOG_FILE:=\/var\/log\/graylog2-server.log}/' /etc/init.d/graylog2
```

- Install the startup script:

```
sudo update-rc.d graylog2 defaults
```

- Start the Graylog2 server with the service command:

```
sudo service graylog2 start
```

Install Graylog Server using Virtual Machine

- Download the OVA image from link given below and save it to your disk locally: https://github.com/Graylog2/graylog2-images/tree/master/ova
- Run the OVA in many systems like VMware or VirtualBox.

HBase Installation

- Download hbase-0.98.15-hadoop2.tar.gz
- Unzip the tar file using below command:

```
tar -xvf hbase-0.98.15-hadoop2.tar.gz
```

- Create directory using below command:

```
sudo mkdir /usr/lib/hbase
```

- Move hbase-0.98.15-hadoop2 to hbase using below command:

```
mv hbase-0.98.15-hadoop2/usr/lib/hbase/hbase-0.98.15-hadoop2 hbase
```

- Configuring HBase with java
  - Open your hbase/conf/hbase-env.sh and set the path to the java installed in your system:

```
export JAVA_HOME=/usr/lib/jvm/jdk1.7.0_25
```

  - Set the HBASE_HOME path in bashrc file
    - Open bashrc file using this command:

```
gedit ~/.bashrc
```

    - In bashrc file append the below 2 statements:

```
export HBASE_HOME=/usr/lib/hbase/hbase-0.98.15-hadoop2

export PATH=$PATH:$HBASE_HOME/bin
```

- To start HBase issue following commands:

```
HBASE_PATH$ bin/start-hbase.sh

HBASE_PATH$ bin/hbase shell
```

- Create centinel table in HBase with stream,alert,dashboard and stringdata as column families using below command:

```
create 'centinel','stream','alert','dashboard','stringdata'
```

- To stop HBase issue following command:

```
HBASE_PATH$ bin/stop-hbase.sh
```

Apache Flume Installation

- Download apache-flume-1.6.0.tar.gz
- Copy the downloaded file to the directory where you want to install Flume.
- Extract the contents of the apache-flume-1.6.0.tar.gz file using below command. Use sudo if necessary:

```
tar -xvzf apache-flume-1.6.0.tar.gz
```

- Starting flume
  - Navigate to the Flume installation directory.
  - Issue the following command to start flume-ng agent:

```
./flume-ng agent --conf conf --conf-file multiplecolumn.conf --name a1 -
↪Dflume.root.logger=INFO,console
```

Apache Drill Installation

- Download apache-drill-1.1.0.tar.gz
- Copy the downloaded file to the directory where you want to install Drill.
- Extract the contents of the apache-drill-1.1.0.tar.gz file using below command:

```
tar -xvzf apache-drill-1.1.0.tar.gz
```

- Starting Drill:
  - Navigate to the Drill installation directory.
  - Issue the following command to launch Drill in embedded mode:

```
bin/drill-embedded
```

- Access the Apache Drill UI on link: http://localhost:8047/
- Go to "Storage" tab and enable "HBase" storage plugin.

Deploying plugins

- Use the following command to download git repository of Centinel:

```
git clone https://git.opendaylight.org/gerrit/p/centinel
```

- Navigate to the installation directory and build the code using maven by running below command:

```
mvn clean install
```

- After building the maven project, a jar file named `centinel-SplittingSerializer-0.0.1-SNAPSHOT.jar` will be created in `centinel/plugins/centinel-SplittingSerializer/target` inside the workspace directory. Copy and rename this jar file to `centinel-SplittingSerializer.jar` (as mentioned in configuration file of flume) and save at location `apache-flume-1.6.0-bin/lib` inside flume directory.
- After successful build, copy the jar files present at below locations to `/opt/graylog/plugin` in graylog server(VM):

```
centinel/plugins/centinel-alertcallback/target/centinel-alertcallback-0.1.0-
↪SNAPSHOT.jar

centinel/plugins/centinel-output/target/centinel-output-0.1.0-SNAPSHOT.jar
```

- Restart the server after adding plugin using below command:

```
sudo graylog-ctl restart graylog-server
```

Configure rsyslog

Make changes to following file:

```
/etc/rsyslog.conf
```

- Uncomment `$InputTCPServerRun 1514`
- Add the following lines:

```
module(load="imfile" PollingInterval="10") #needs to be done just once
input(type="imfile"
File="<karaf.log>" #location of log file
StateFile="statefile1"
Tag="tag1")
*.* @@127.0.0.1:1514 # @@used for TCP
```

  - Use the following format and comment the previous one:

```
$ActionFileDefaultTemplate RSYSLOG_SyslogProtocol23Format
```

- Use the below command to send Centinel logs to a port:

```
tail -f <location of log file>/karaf.log|logger
```

- Restart rsyslog service after making above changes in configuration file:

```
sudo service rsyslog restart
```

Install the following feature

Finally, from the Karaf console install the Centinel feature with this command:

```
feature:install odl-centinel-all
```

Verifying your Installation

If the feature install was successful you should be able to see the following Centinel commands added:

```
centinel:list

centinel:purgeAll
```

Troubleshooting

Check the `../data/log/karaf.log` for any exception related to Centinel related features

Upgrading From a Previous Release

Only fresh installation is supported.

Uninstalling Centinel

To uninstall the Centinel functionality, you need to do the following from Karaf console:

```
feature:uninstall centinel-all
```

Its recommended to restart the Karaf container after uninstallation of the Centinel functionality.

NetVirt Installation Guide

NetVirt Design Specifications

Starting from Carbon, NetVirt uses an RST format Design Specification document for all new features. These specifications are a perfect way to understand various NetVirt features.

Contents:

Table of Contents

Title of the feature

[gerrit filter: https://git.opendaylight.org/gerrit/#/q/topic:cool-topic]

Brief introduction of the feature.

Problem description

Detailed description of the problem being solved by this feature

Use Cases

Use cases addressed by this feature.

Proposed change

Details of the proposed change.

Pipeline changes

Any changes to pipeline must be captured explicitly in this section.

Yang changes

This should detail any changes to yang models.

Listing 1.1: example.yang

```
module example {
    namespace "urn:opendaylight:netvirt:example";
    prefix "example";

    import ietf-yang-types {prefix yang; revision-date "2013-07-15";}

    description "An example YANG model.";

    revision 2017-02-14 { description "Initial revision"; }
}
```

Configuration impact

Any configuration parameters being added/deprecated for this feature? What will be defaults for these? How will it impact existing deployments?

Note that outright deletion/modification of existing configuration is not allowed due to backward compatibility. They can only be deprecated and deleted in later release(s).

Clustering considerations

This should capture how clustering will be supported. This can include but not limited to use of CDTCL, EOS, Cluster Singleton etc.

Other Infra considerations

This should capture impact from/to different infra components like MDSAL Datastore, karaf, AAA etc.

Security considerations

Document any security related issues impacted by this feature.

Scale and Performance Impact

What are the potential scale and performance impacts of this change? Does it help improve scale and performance or make it worse?

Targeted Release

What release is this feature targeted for?

Alternatives

Alternatives considered and why they were not selected.

Usage

How will end user use this feature? Primary focus here is how this feature will be used in an actual deployment.

e.g. For most netvirt features this will include OpenStack APIs.

This section will be primary input for Test and Documentation teams. Along with above this should also capture REST API and CLI.

Features to Install

odl-netvirt-openstack

Identify existing karaf feature to which this change applies and/or new karaf features being introduced. These can be user facing features which are added to integration/distribution or internal features to be used by other projects.

REST API

Sample JSONS/URIs. These will be an offshoot of yang changes. Capture these for User Guide, CSIT, etc.

CLI

Any CLI if being added.

Implementation

Assignee(s)

Who is implementing this feature? In case of multiple authors, designate a primary assignee and other contributors.

Primary assignee: <developer-a>, <irc nick>, <email>

Other contributors: <developer-b>, <irc nick>, <email> <developer-c>, <irc nick>, <email>

Work Items

Break up work into individual items. This should be a checklist on a Trello card for this feature. Provide the link to the trello card or duplicate it.

Dependencies

Any dependencies being added/removed? Dependencies here refers to internal [other ODL projects] as well as external [OVS, karaf, JDK etc]. This should also capture specific versions if any of these dependencies. e.g. OVS version, Linux kernel version, JDK etc.

This should also capture impacts on existing projects that depend on Netvirt.

Following projects currently depend on Netvirt: Unimgr

Testing

Capture details of testing that will need to be added.

Unit Tests

Integration Tests

CSIT

Documentation Impact

What is the impact on documentation for this change? If documentation changes are needed call out one of the <contributors> who will work with the Project Documentation Lead to get the changes done.

Don't repeat details already discussed but do reference and call them out.

References

Add any useful references. Some examples:

- Links to Summit presentation, discussion etc.
- Links to mail list discussions
- Links to patches in other projects
- Links to external documentation

[1] OpenDaylight Documentation Guide

[2] https://specs.openstack.org/openstack/nova-specs/specs/kilo/template.html

---

Note: This template was derived from [2], and has been modified to support our project.

Table of Contents

ACL Statistics

https://git.opendaylight.org/gerrit/#/q/topic:acl-stats

This feature is to provide additional operational support for ACL through statistical counters. ACL rules provide security to VMs by filtering packets in either directions (ingress/egress). Using OpenFlow statistical counters, ODL will provide additional information on the number of packets dropped by the ACL rules. This information is made available to the operator "on demand".

Drop statistics will be provided for below cases:

- Packets dropped due to ACL rules
- Packets dropped due to INVALID state. The INVALID state means that the packet can't be identified or that it does not have any state. This may be due to several reasons, such as the system running out of memory or ICMP error messages that do not respond to any known connections.

The packet drop information provided through the statistical counters enable operators to trouble shoot any misbehavior and take appropriate actions through automated or manual intervention.

Collection and retrieval of information on the number of packets dropped by the SG rules

- Done for all (VM) ports in which SG is configured
- Flow statistical counters (in OpenFlow) are used for this purpose
- The information in these counters are made available to the operator, on demand, through an API

This feature will only be supported with Stateful ACL mode.

Problem description

With only ACL support, operators would not be able to tell how many packets dropped by ACL rules. This enhancement planned is about ACL module supporting aforementioned limitation.

Use Cases

Collection and retrieval of information on the number of packets dropped by the ACL rules

- Done for all (VM) ports in which ACL is configured
- The information in these counters are made available to the operator, on demand, through an API
- Service Orchestrator/operator can also specify ports selectively where ACL rules are configured

Proposed change

ACL Changes

Current Stateful ACL implementation has drop flows for all ports combined for a device. This needs to be modified to have drop flows for each of the OF ports connected to VMs (Neutron Ports).

With the current implementation, drop flows are as below:

```
cookie=0x6900000, duration=938.964s, table=252, n_packets=0, n_bytes=0,␣
↪priority=62020,
        ct_state=+inv+trk actions=drop

cookie=0x6900000, duration=938.969s, table=252, n_packets=0, n_bytes=0, priority=50,
        ct_state=+new+trk actions=drop
```

Now, for supporting Drop packets statistics per port, ACL will be updated to replace above flows with new DROP flows with lport tag as metadata for each of the VM (Neutron port) being added to OVS as specified below:

```
cookie=0x6900001, duration=938.964s, table=252, n_packets=0, n_bytes=0,␣
↪priority=62015,
        metadata=0x10000000000/0xffffff0000000000, ct_state=+inv+trk actions=drop

cookie=0x6900001, duration=938.969s, table=252, n_packets=0, n_bytes=0, priority=50,
        metadata=0x10000000000/0xffffff0000000000, ct_state=+new+trk actions=drop
```

Drop flows details explained above are for pipeline egress direction. For ingress side, similar drop flows would be added with `table=41`.

Also, new cookie value `0x6900001` would be added with drop flows to identify it uniquely and priority `62015` would be used with +inv+trk flows to give higher priority for +est and +rel flows.

Drop packets statistics support

ODL Controller will be updated to provide a new RPC/NB REST API `<get-acl-port-statistics>` in ACL module with `ACL Flow Stats Request` and `ACL Flow Stats Response` messages. This RPC/API will retrieve details of dropped packets by Security Group rules for all the neutron ports specified as part of `ACL Flow Stats Request`. The retrieved information (instantaneous) received in the OF reply message is formatted as `ACL Flow Stats Response` message before sending it as a response towards the NB.

`<get-acl-port-statistics>` RPC/API implementation would be triggering `opendaylight-direct-statistics:get-flow-statistics` request of OFPlugin towards OVS to get the flow statistics of ACL tables (ingress / egress) for the required ports.

ACL Flow Stats Request/Response messages are explained in subsequent sections.

Pipeline changes

No changes needed in OF pipeline. But, new flows as specified in above section would be added for each of the Neutron ports being added.

Yang changes

New yang file will be created with RPC as specified below:

Listing 1.2: acl-live-statistics.yang

```
module acl-live-statistics {
    namespace "urn:opendaylight:netvirt:acl:live:statistics";

    prefix "acl-stats";

    import ietf-interfaces {prefix if;}
    import aclservice {prefix aclservice; revision-date "2016-06-08";}

    description "YANG model describes RPC to retrieve ACL live statistics.";

    revision "2016-11-29" {
        description "Initial revision of ACL live statistics";
    }

    typedef direction {
        type enumeration {
            enum ingress;
            enum egress;
            enum both;
        }
    }

    grouping acl-drop-counts {
        leaf drop-count {
            description "Packets/Bytes dropped by ACL rules";
            type uint64;
        }
        leaf invalid-drop-count {
            description "Packets/Bytes identified as invalid";
            type uint64;
        }
    }

    grouping acl-stats-output {
        description "Output for ACL port statistics";
        list acl-interface-stats {
            key "interface-name";
            leaf interface-name {
                type leafref {
                    path "/if:interfaces/if:interface/if:name";
                }
            }
            list acl-drop-stats {
                max-elements "2";
                min-elements "0";
                leaf direction {
                    type identityref {
                        base "aclservice:direction-base";
                    }
                }
                container packets {
                    uses acl-drop-counts;
                }
                container bytes {
                    uses acl-drop-counts;
                }
```

```
            }
            container error {
                leaf error-message {
                    type string;
                }
            }
        }
    }

    grouping acl-stats-input {
        description "Input parameters for ACL port statistics";

        leaf direction {
            type identityref {
                base "aclservice:direction-base";
            }
            mandatory "true";
        }
        leaf-list interface-names {
            type leafref {
                path "/if:interfaces/if:interface/if:name";
            }
            max-elements "unbounded";
            min-elements "1";
        }
    }

    rpc get-acl-port-statistics {
        description "Get ACL statistics for given list of ports";

        input {
            uses acl-stats-input;
        }
        output {
            uses acl-stats-output;
        }
    }
}
```

Configuration impact

No configuration parameters being added/deprecated for this feature

Clustering considerations

No additional changes required to be done as only one RPC is being supported as part of this feature.

Other Infra considerations

N.A.

Security considerations

N.A.

Scale and Performance Impact

N.A.

Targeted Release

Carbon

Alternatives

Dispatcher table (table 17 and table 220) based approach of querying drop packets count was considered. ie., arriving drop packets count by below rule:

<total packets entered ACL tables> - <total packets entered subsequent service>

This approach was not selected as this only provides total packets dropped count per port by ACL services and does not provide details of whether it's dropped by ACL rules or for some other reasons.

Usage

Features to Install

odl-netvirt-openstack

REST API

Get ACL statistics

Following API gets ACL statistics for given list of ports.

Method: POST

URI: /operations/acl-live-statistics:get-acl-port-statistics

Parameters:

Parameter	Type	Possible Values	Comments
"direction"	Enum	ingress/egress/both	Required
"interface-names"	Array [UUID String]	[<UUID String>,<UUID String>,..]	Required (1,N)

Example:

```
{
    "input":
    {
         "direction": "both",
         "interface-names": [
             "4ae8cd92-48ca-49b5-94e1-b2921a2661c5",
```

```
            "6c53df3a-3456-11e5-a151-feff819cdc9f"
        ]
    }
}
```

Possible Responses:

RPC Success:

```
{
    "output": {
    "acl-port-stats": [
    {
        "interface-name": "4ae8cd92-48ca-49b5-94e1-b2921a2661c5",
        "acl-drop-stats": [
        {
            "direction": "ingress",
            "bytes": {
                "invalid-drop-count": "0",
                "drop-count": "300"
            },
            "packets": {
                "invalid-drop-count": "0",
                "drop-count": "4"
            }
        },
        {
            "direction": "egress",
            "bytes": {
                "invalid-drop-count": "168",
                "drop-count": "378"
            },
            "packets": {
                "invalid-drop-count": "2",
                "drop-count": "9"
            }
        }]
    },
    {
        "interface-name": "6c53df3a-3456-11e5-a151-feff819cdc9f",
        "acl-drop-stats": [
        {
            "direction": "ingress",
            "bytes": {
                "invalid-drop-count": "1064",
                "drop-count": "1992"
            },
            "packets": {
                "invalid-drop-count": "18",
                "drop-count": "23"
             }
        },
        {
            "direction": "egress",
            "bytes": {
                "invalid-drop-count": "462",
                "drop-count": "476"
             },
```

```
            "packets": {
                "invalid-drop-count": "11",
                "drop-count": "6"
            }
        }]
    }]
}
```

RPC Success (with error for one of the interface):

```
{
    "output":
    {
        "acl-port-stats": [
        {
            "interface-name": "4ae8cd92-48ca-49b5-94e1-b2921a2661c5",
            "acl-drop-stats": [
            {
                "direction": "ingress",
                "bytes": {
                    "invalid-drop-count": "0",
                    "drop-count": "300"
                },
                "packets": {
                    "invalid-drop-count": "0",
                    "drop-count": "4"
                }
            },
            {
                "direction": "egress",
                "bytes": {
                    "invalid-drop-count": "168",
                    "drop-count": "378"
                },
                "packets": {
                    "invalid-drop-count": "2",
                    "drop-count": "9"
                }
            },
            {
                "interface-name": "6c53df3a-3456-11e5-a151-feff819cdc9f",
                "error": {
                    "error-message": "Interface not found in datastore."
                }
            }]
        }]
    }
}
```

Note: Below are error messages for the interface:

1. "Interface not found in datastore."
2. "Failed to find device for the interface."
3. "Unable to retrieve drop counts due to error: <<error message>>"
4. "Unable to retrieve drop counts as interface is not configured for statistics collection."

5. “Operation not supported for ACL <<Stateless/Transparent/Learn>> mode”

CLI

No CLI being added for this feature

Implementation

Assignee(s)

Primary assignee: <Somashekar Byrappa>

Other contributors: <Shashidhar R>

Work Items

1. Adding new drop rules per port (in table 41 and 252)
2. Yang changes
3. Supporting new RPC

Dependencies

This doesn’t add any new dependencies.

This feature has dependency on below bug reported in OF Plugin:

Bug 7232 - Problem observed with “get-flow-statistics” RPC call

Testing

Unit Tests

Following test cases will need to be added/expanded

1. Verify ACL STAT RPC with single Neutron port
2. Verify ACL STAT RPC with multiple Neutron ports
3. Verify ACL STAT RPC with invalid Neutron port
4. Verify ACL STAT RPC with mode set to “transparent/learn/stateless”

Also, existing unit tests will be updated to include new drop flows.

Integration Tests

Integration tests will be added, once IT framework is ready

CSIT

Following test cases will need to be added/expanded

1. Verify ACL STAT RPC with single Neutron port with different directions (ingress, egress, both)
2. Verify ACL STAT RPC with multiple Neutron ports with different directions (ingress, egress, both)
3. Verify ACL STAT RPC with invalid Neutron port
4. Verify ACL STAT RPC with combination of valid and invalid Neutron ports
5. Verify ACL STAT RPC with combination of Neutron ports with few having port-security-enabled as true and others having false

Documentation Impact

This will require changes to User Guide. User Guide needs to be updated with details about new RPC being supported and also about its REST usage.

References

N.A.

Table of Contents

ACL Remote ACL - Indirection Table to Improve Scale

ACL Remote ACL Indirection patches: https://git.opendaylight.org/gerrit/#/q/topic:remote_acl_indirection

This spec is to enhance the initial implementation of ACL remote ACLs filtering which was released in Boron. The Boron release added full support for remote ACLs, however the current implementation does not scale well in terms of flows. The Carbon release will update the implementation to introduce a new indirection table for ACL rules with remote ACLs, to reduce the number of necessary flows, in cases where the port is associated with a single ACL. Due to the complication of supporting multiple ACLs on a single port, the current implementation will stay the same for these cases.

Problem description

Today, for each logical port, an ACL rule results in a flow in the ACL table (ACL2). When a remote ACL is configured on this rule, this flow is multiplied for each VM in the remote ACL, resulting in a very large number of flows.

For example, consider we have:

- 100 computes
- 50 VMs on each compute (5000 VMs total),
- All VMs are in a SG (SG1)
- This SG has a security rule configured on it with remote SG=SG1 (it is common to set the remote SG as itself, to set rules within the SG).

  This would result in 50*5000 = 250,000 flows on each compute, and 25M flows in ODL MDSAL (!).

Use Cases

Neutron configuration of security rules, configured with remote SGs. This optimization will be relevant only when there is a single security group that is associated with the port. In case more than one security group is associated with the port - we will fallback to the current implementation which allows full functionality but with possible flow scaling issues.

Rules with a remote ACL are used to allow certain types of packets only between VMs in certain security groups. For example, configuring rules with the parent security group also configured as a remote security group, allows to configure rules applied only for traffic between VMs in the same security group.

This will be done in the ACL implementation, so any ACL configured with a remote ACL via a different northbound or REST would also be handled.

Proposed change

This blueprint proposes adding a new indirection table in the ACL service in each direction, which will attempt to match the "remote" IP address associated with the packet ("dst_ip" in Ingress ACL, "src_ip" in Egress ACL), and set the ACL ID as defined by the ietf-access-control-list in the metadata. This match will also include the ELAN ID to handle ports with overlapping IPs.

These flows will be added to the ACL2 table. In addition, for each such ip->SG flow inserted in ACL2, we will insert a single SG metadata match in ACL3 for each SG rule on the port configured with this remote SG.

If the IP is associated with multiple SGs - it is impossible to do a 1:1 matching of the SG, so we will not set the metadata at this time and fallback to the current implementation of matching all possible IPs in the ACL table - for this ACL2 will have a default flow passing the unmatched packets to ACL3 with an empty metadata SG_ID write (e.g. 0x0), to prevent potential garbage in the metadata SG ID.

This means that on transition from a single SG on the port to multiple SG (and back), we would need to remove/add these flows from ACL2, and insert the correct rules into ACL3.

ACL1 (211/241):

- This is the ACL that has default allow rules - it is left untouched, and usually goes to ACL2.

ACL2 (212/242):

- For each port with a single SG - we will match on the IPs and the ELAN ID (for tenant awareness) here, and set the SG ID in the metadata, before going to the ACL3 table.
- For any port with multiple SGs (or with no SG) - an empty value (0x0) will be set as the SG ID in the metadata, to avoid potential garbage in the SG ID, and goto ACL3 table.

ACL3 (213/243):

- For each security rule that *doesn't have* a remote SG, we keep the behavior the same: ACL3 matches on rule, and resubmits to dispatcher if there is a match (Allow). The SG ID in the metadata will **not** be matched.
- For each security rule that *does have* a remote SG, we have two options:
  - For ports belonging to the remote SG that are associated with a single SG - there will be a single flow per rule, matching the SG ID from the metadata (in addition to the other rule matches) and allowing the packet.
  - For ports belonging to the remote SG that are associated with multiple SGs - the existing implementation will stay the same, multiplying the rule with all possible IP matches from the remote security groups.

Considering the example from the problem description above, the new implementation would result in a much reduced number of flows:

5000+50 = 5050 flows on each compute, and 505,000 flows in ODL MDSAL.

As noted above, this would require using part of the metadata for writing/matching of an ACL ID. We would likely require at least 12 bits for this, to support up to 4K SGs, where 16 bits to support up to 65K would be ideal. If the metadata bits are not available, we can use a register for this purpose (16 bits).

In addition, the dispatcher will set the ELAN ID in the metadata before entering the ACL services, to allow tenant aware IP to SG detection, supporting multi-tenants with IP collisions.

Pipeline changes

ACL3 will be added, and the flows in ACL2/ACL3 will be modified as noted above in the proposed change:

Table	Match	Action
Dispatcher	metadata=service_id:ACL	write_metadata:(elan_id=ELAN, service_id=NEXT), goto_table:ACL1
ACL1 (211/241)	goto_table:ACL2	
ACL2 (212/242)	metadata=ELAN_ID, ip_src/dst=VM1_IP	write_metadata:(remote_acl=id), goto_table:ACL3
ACL2 (212/242)	metadata=ELAN_ID, ip_src/dst=VM2_IP	write_metadata:(remote_acl=id), goto_table:ACL3
...		
ACL2 (212/242)		goto_table:ACL3
ACL3 (213/243)	metadata=lport, <acl_rule>	resubmit(,DISPATCHER) (X)
ACL3 (213/243)	metadata=lport+remote_acl, <acl_rule>	resubmit(,DISPATCHER) (XX)
ACL3 (213/243)	metadata=lport,ip_src/dst=VM1_IP, <acl_rule>	resubmit(,DISPATCHER) (XXX)
ACL3 (213/243)	metadata=lport,ip_src/dst=VM2_IP, <acl_rule>	resubmit(,DISPATCHER) (XXX)
...		

(X) These are the regular rules, not configured with any remote SG.

(XX) These are the proposed rules with the optimization - assuming the lport is using a single ACL.

(XXX) These are the remote SG rules in the current implementation, which we will fall back to if the lport has multiple ACLs.

Table Numbering:

Currently the Ingress ACLs use tables *40,41,42* and the Egress ACLs use tables *251,252,253*.

Table 43 is already proposed to be taken by SNAT, and table 254 is considered invalid by OVS. To overcome this and align Ingress/Egress with symmetric numbering, I propose the following change:

- Ingress ACLs: 211, 212, 213, 214
- Egress ACLs: 241, 242, 243, 244

ACL1: INGRESS/EGRESS_ACL_TABLE ACL2: INGRESS/EGRESS_ACL_REMOTE_ACL_TABLE ACL3: INGRESS/EGRESS_ACL_FILTER_TABLE

ACL4 is used only for Learn implementation for which an extra table is required.

Yang changes

None.

Configuration impact

None.

Clustering considerations

None.

Other Infra considerations

None.

Security considerations

None.

Scale and Performance Impact

See example in description. The scale of the flows will be drastically reduced when using remote ACLs.

Targeted Release

Carbon

Alternatives

For fully optimized support in all scenarios for remote SGs, meaning including support for ports with multiple ACLs on them, we did consider implementing a similar optimization.

However, for this to happen due to OpenFlow limitations we would need to introduce an internal dispatcher inside the ACL services, meaning we loop the ACL service multiple times, each time setting a different metadata SG value for the port.

For another approach we could use a bitmask, but this would limit the number of possible SGs to be the number of bits in the mask, which is much too low for any reasonable use case.

Usage

Any configuration of ACL rules with remote ACLs will receive this optimization if the port is using a single SG.

Functionality should remain as before in any case.

Features to Install

Install the ODL Karaf feature for NetVirt (no change):

- odl-netvirt-openstack

REST API

None.

CLI

Refer to the Neutron CLI Reference[1] for the Neutron CLI command syntax for managing Security Rules with Remote Security Groups.

Implementation

Assignee(s)

Who is implementing this feature? In case of multiple authors, designate a primary assigne and other contributors.

Primary assignee:

- Alon Kochba <alonko@hpe.com>
- Aswin Suryanarayanan <asuryana@redhat.com>

Other contributors:

- ?

Work Items

Task list in Carbon Trello

Dependencies

None.

Testing

Unit Tests

Integration Tests

CSIT

We should add tests verifying remote SG configuration functionality. There should be at least:

- One security rule allowing ICMP traffic between VMs in the same SG.
- One positive test, checking ICMP connectivity works between two VMs using the same SG.
- One negative test, checking ICMP connectivity does not work between two VMs, one using the SG configured with the rule above, and the other using a separate security group with all directions allowed.

[1] Neutron Security Groups http://docs.openstack.org/user-guide/cli-nova-configure-access-security-for-instances.html

Documentation Impact

None.

References

Table of Contents

ACL - Reflecting the ACL changes on existing traffic

ACL patches: https://git.opendaylight.org/gerrit/#/q/topic:acl-reflection-on-existing-traffic

This spec describes the new implementation for applying ACL changes on existing traffic.

In current ACL implementation, once a connection had been committed to the connection tracker, the connection would continue to be allowed, even if the policy defined in the ACL table has changed. This spec will explain the new approach that ensures ACL policy changes will affect existing connections as well. This approach will improve the pipeline behaviour in terms of reliable traffic between the VMs.

Problem description

When the traffic between two VMs starts, the first packet will match the actual SG flow, which commits the packets in connection tracker. It changes the state of the packets to established. Further traffic will match the global conntrack flow and go through the connection tracker straightly. This will continue until we terminate the established traffic.

When a rule is removed from the VM, the ACL flow getting removed from the respective tables. But, the already established traffic is still working, because the connection still exists as 'committed' in the conntrack tracker.

For example, consider the below scenario which explains the problem in detail,

- Create a VM and associate the rule which allows ICMP
- Ping the DHCP server from the VM
- Remove the ICMP rule and check the ongoing traffic

When we remove the ICMP rule, the respective ICMP flow getting removed from the respective table (For egress, table 213 and For Ingress, table 243). But, Since the conntrack flow having high priority than the SG flow, the packets are matched by the conntrack flow and the live traffic is unaware of the flow removal.

The traffic between the VMs should be reliable and it should be succeeded accordance with SG flow. When a SG rule is removed from the VM, the packets of ongoing traffic should be dropped.

Use Cases

The new ACL implementation will affect the below use cases,

- VM Creation/Deletion with SG
- SG Rule addition and removal to/from existing SG associated to ports

Proposed change

This spec proposes the fix that requires a new table (210/240) in the existing pipeline.

In this approach, we will use the "ct_mark" flag of connection tracker. The default value of ct_mark is zero.

- ct_mark=0 matches the packet in new state
- ct_mark=1 matches the packet in established state

For every new traffic, the ct_mark value will be zero. When the traffic begins, the first packet of every new traffic will be matched by the respective SG flow which commits the packets into the connection tracker and changes the ct_mark value to 1. So, every packets of established traffic will have the ct_mark value as 1.

In conntrack flow, we will have a ct_mark=1 match condition. After first packet committed to the connection tracker, further packets of established traffic will be matched by the conntrack flow straightly.

In every SG flow, we will have below changes, "table=213/243, priority=3902, **ct_state=+trk** ,icmp,reg6=0x200/0xfffff00 actions=ct(commit,zone=6001, **exec(set_field:0x1->ct_mark)**),resubmit(,17/220)"

- The SG flow will match the packets which are in tracked state. It will commit the packet into the connection tracker. It will change the ct_mark value to 1.
- When a VM having duplicate flows, the removal of one flow should not affect the existing traffic.

  For example, consider a VM having ingress ICMP and Other protocol (ANY) rule. Ping the VM from the DHCP server. Removal of ingress ICMP rule from the VM should not affect the existing traffic. Because the Other protocol ANY flow will match the established packets of existing ICMP traffic and should make the communication possible. To make the communication possible in above specific scenarios, we should match the established packets in every SG flow. So, We will remove the "+new" check from the ct_state condition of every ACL flow to recommit the established packets again into the conntrack.

In conntrack flow, "table=213/243, priority=62020,ct_state=-new+est-rel-inv+trk, **ct_mark=0x1** actions=resubmit(,17/220)" "table=213/243, priority=62020,ct_state=-new-est+rel-inv+trk, **ct_mark=0x1** actions=resubmit(,17/220)"

- The conntrack flow will match the packet which are in established state.
- For every new traffic, the first packet will be matched by the SG flow, which will change the ct_mark value to 1. So, further packets will match the conntrack flow straightly.

In default drop flow of table 213/243, "table=213, n_packets=0, n_bytes=0, priority=50, **ct_state=+trk** ,metadata=0x20000000000/0xfffff0000000000 actions=drop" "table=243, n_packets=6, n_bytes=588, priority=50, **ct_state=+trk** ,reg6=0x300/0xfffff00 actions=drop"

- For every VM, we are having a default drop flow to measure the drop statistics of particular VM. So, we will remove the "+new" state check from the ct_state to measure the drop counts accurately.

Deletion of SG flow will add the below flow with configured hard time out in the table 212/242.

[1] "table=212/242, n_packets=73, n_bytes=7154, priority=40,icmp,reg6=0x200/0xfffff00,ct_mark=1 actions=ct(commit, zone=5500, **exec(set_field:0x0->ct_mark)**),goto_table:ACL4"

- It will match the ct_mark value with the one and change the ct_mark to zero.

The below tables describes the default hard time out of each protocol as configured in the conntrack.

Protocol	Time out (secs)
ICMP	30
TCP	18000
UDP	180

Please refer the Pipeline Changes for table information.

For Egress, Dispatcher table (table 17) will forward the packets to the new table 210 where we will check the source match. It will forward the packet to 211 to match the destination of the packets. After the destination of the packet verified, The packets will forward to the table 212. New flow in the table, will match the ct_mark value and forward the packets to the 213 table.

Similarly, for Ingress, the packets will be forwarded through, Dispatcher table (220) >> New table (240) >> 241 >> 242 >> 243.

In dispatcher flows, we will have the below changes which will change the table 211/241 from the goto_table action to the new table 210/240.

"table=17, priority=10,metadata=0x20000000000/0xffffff0000000000 actions=write_metadata:0x900002157f000000/0xfffffffffffffffe, **goto_table:210**"

"table=220, priority=6,reg6=0x200 actions=load:0x90000200->NXM_NX_REG6[],write_metadata:0x157f000000/0xfffffffffe, **goto_table:240**"

Deletion of SG rule will add a new flow in the table 212/242 as mentioned above. The first packet after SG got deleted, will match the above new flow and will change the ct_mark value to zero. So this packet will not match the conntrack flow and will check the ACL4 table whether it having any other flows to match this packet. If the SG flow found, the packet will be matched and change the ct_mark value 1.

If we restore the SG rule again, we will delete the added flow [1] from the 212/242 table, so the packets of existing traffic will match the newly added SG flow in ACL4 table and proceed successfully.

Sample flows to be installed in each scenario,

SG rule addition

SG flow: [ADD] "table=213/243, n_packets=33, n_bytes=3234, priority=62021, **ct_state=+trk**, icmp, reg6=0x200/0xfffff00 actions=ct(commit,zone=6001, **exec(set_field:0x1->ct_mark)**),resubmit(,17/220)"

Conntrack flow: [DEFAULT] "table=213/243, n_packets=105, n_bytes=10290, priority=62020,ct_state=-new+est-rel-inv+trk, **ct_mark=0x1** actions=resubmit(,17/220)"

SG Rule deletion

SG flow: [DELETE] "table=213/243, n_packets=33, n_bytes=3234, priority=62021, ct_state=+trk,icmp,reg6=0x200/0xfffff00 actions=ct(commit,zone=6001,exec(set_field:0x1->ct_mark)),resubmit(,17/220)"

New flow: [ADD] "table=212/242, n_packets=73, n_bytes=7154, priority=62021, **ct_mark=0x1**,icmp,reg6=0x200/0xfffff00 actions=ct(commit, **exec(set_field:0x0->ct_mark)**),goto_table:213/243"

Rule Restore

SG flow: [ADD] "table=213/243, n_packets=33, n_bytes=3234, priority=62021, ct_state=+trk, icmp,reg6=0x200/0xfffff00 actions=ct(commit,zone=6001,exec(set_field:0x1->ct_mark)),resubmit(,17/220)"

New flow: [DELETE] "table=212/242, n_packets=73, n_bytes=7154, priority=62021,ct_mark=0x1,icmp,reg6=0x200/0xfffff00 actions=ct(commit,exec(set_field:0x0->ct_mark)),goto_table:213/243"

The new tables (210/240) will matches the source and the destination of the packets respectively. So, a default flow will be added in the table 210/240 with least priority to drop the packets.

"table=210/240, n_packets=1, n_bytes=98, priority=0 actions=drop"

Flow Sample:

Egress flows before the changes,

cookie=0x6900000, duration=30.590s, table=17, n_packets=108, n_bytes=10624, priority=10,metadata=0x20000000000/0xffffff0000000000 actions=write_metadata:0x9000021389000000/0xfffffffffffffffe,goto_table:211
cookie=0x6900000, duration=30.247s, table=211, n_packets=0, n_bytes=0, priority=61010,ipv6,dl_src=fa:16:3e:93:dc:92,ipv6_src=fe80::f816:3eff:fe93:dc92 actions=ct(table=212,zone=5001) cookie=0x6900000, duration=30.236s, table=211, n_packets=96, n_bytes=9312, priority=61010,ip,dl_src=fa:16:3e:93:dc:92,nw_src=10.100.5.3 actions=ct(table=212,zone=5001) cookie=0x6900000, duration=486.527s, table=211, n_packets=2, n_bytes=180, priority=0 actions=drop

cookie=0x6900000, duration=30.157s, table=212, n_packets=0, n_bytes=0, priority=50,ipv6,metadata=0x1389000000/0xffff000000,ipv6_dst=fe80::f816:3eff:fe93:dc92 actions=write_metadata:0x2/0xfffffe,goto_table:212 cookie=0x6900000, duration=30.152s, table=212, n_packets=0, n_bytes=0, priority=50,ip,metadata=0x1389000000/0xffff000000,nw_dst=10.100.5.3 actions=write_metadata:0x2/0xfffffe,goto_table:212 cookie=0x6900000, duration=486.527s, table=212, n_packets=96, n_bytes=9312, priority=0 actions=goto_table:212 cookie=0x6900000, duration=486.056s, table=213, n_packets=80, n_bytes=8128, priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,17) cookie=0x6900000, duration=485.948s, table=213, n_packets=0, n_bytes=0, priority=62020,ct_state=-new-est+rel-inv+trk actions=resubmit(,17) cookie=0x6900001, duration=30.184s, table=213, n_packets=0, n_bytes=0, priority=62015,ct_state=+inv+trk,metadata=0x20000000000/0xfffff0000000000 actions=drop cookie=0x6900000, duration=30.177s, table=213, n_packets=16, n_bytes=1184, priority=1000,ct_state=+new+trk,ip,metadata=0x20000000000/0xfffff0000000000 actions=ct(commit,zone=5001),resubmit(,17) cookie=0x6900000, duration=30.168s, table=213, n_packets=0, n_bytes=0, priority=1001,ct_state=+new+trk,ipv6,metadata=0x20000000000/0xfffff0000000000 actions=ct(commit,zone=5001),resubmit(,17) cookie=0x6900001, duration=30.207s, table=213, n_packets=0, n_bytes=0, priority=50,ct_state=+new+trk,metadata=0x20000000000/0xfffff0000000000 actions=dro

After the changes, flows will be,

cookie=0x6900000, duration=30.590s, table=17, n_packets=108, n_bytes=10624, priority=10,metadata=0x20000000000/0xffffff0000000000 actions=write_metadata:0x9000021389000000/0xfffffffffffffffe,goto_table:210 cookie=0x6900000, duration=30.247s, table=210, n_packets=0, n_bytes=0, priority=61010,ipv6,dl_src=fa:16:3e:93:dc:92,ipv6_src=fe80::f816:3eff:fe93:dc92 actions=ct(table=211,zone=5001) cookie=0x6900000, duration=30.236s, table=210, n_packets=96, n_bytes=9312, priority=61010,ip,dl_src=fa:16:3e:93:dc:92,nw_src=10.100.5.3 actions=ct(table=211,zone=5001) cookie=0x6900000, duration=486.527s, table=210, n_packets=2, n_bytes=180, priority=0 actions=drop cookie=0x6900000, duration=30.157s, table=211, n_packets=0, n_bytes=0, priority=50,ipv6,metadata=0x1389000000/0xffff000000,ipv6_dst=fe80::f816:3eff:fe93:dc92 actions=write_metadata:0x2/0xfffffe,goto_table:212 cookie=0x6900000, duration=30.152s, table=211, n_packets=0, n_bytes=0, priority=50,ip,metadata=0x1389000000/0xffff000000,nw_dst=10.100.5.3 actions=write_metadata:0x2/0xfffffe,goto_table:212 cookie=0x6900000, duration=486.527s, table=211, n_packets=96, n_bytes=9312, priority=0 actions=goto_table:212 cookie=0x6900000, duration=486.527s, table=212, n_packets=96, n_bytes=9312, priority=0 actions=goto_table:213 cookie=0x6900000, duration=486.056s, table=213, n_packets=80, n_bytes=8128, priority=62020,ct_state=-new+est-rel-inv+trk,ct_mark=0x1 actions=resubmit(,17) cookie=0x6900000, duration=485.948s, table=213, n_packets=0, n_bytes=0, priority=62020,ct_state=-new-est+rel-inv+trk,ct_mark=0x1 actions=resubmit(,17) cookie=0x6900001, duration=30.184s, table=213, n_packets=0, n_bytes=0, priority=62015,ct_state=+inv+trk,metadata=0x20000000000/0xfffff0000000000 actions=drop cookie=0x6900000, duration=30.177s, table=213, n_packets=16, n_bytes=1184, priority=1000,ct_state=+trk,ip,metadata=0x20000000000/0xfffff0000000000 actions=ct(commit,zone=5001,exec(set_field:0x1->ct_mark)),resubmit(,17)

cookie=0x6900000, duration=30.168s, table=213, n_packets=0, n_bytes=0, priority=1001,ct_state=+new+trk,ipv6,metadata=0x20000000000/0xffffff0000000000 actions=ct(commit,zone=5001),resubmit(,17) cookie=0x6900001, duration=30.207s, table=213, n_packets=0, n_bytes=0, priority=50,ct_state=+trk,metadata=0x20000000000/0xffffff0000000000 actions=drop

New flow will be installed in table 212 when we delete SG rule, "cookie=0x6900000, duration=30.177s, table=212, n_packets=16, n_bytes=1184, priority=1000,ct_state=+trk,ip,metadata=0x20000000000/0xffffff0000000000,ct_mark=1,idle_timeout=1800 actions=ct(commit,zone=5001,exec(set_field:0x0->ct_mark)),goto_table:213"

Similarly, the ingress related flows will have the same changes as mentioned above.

Pipeline changes

The propose changes includes:

- New tables 210 and 240
- Re-purposed tables 211, 212, 241, 242

The propose will re-purpose the table 211 and 212 of egress, table 241 and 242 of ingress.

Currently, for egress, we are using the table 211 for source match and 212 for destination match. In new propose, we will use the new table 210 for source match, table 211 for destination match and table 212 for new flow installation when we delete the SG flow.

For Egress, the traffic will use the tables in following order, 17 >> 210 >> 211 >> 212 >> 213.

Similarly, for ingress, currently we are using the table 241 for destination match and 242 for source match. In new propose, we will use the new table 240 for destination match, table 241 for source match and table 242 for new flow installation when we delete the SG flow.

For Ingress, the traffic will use the tables in following order, 220 >> 240 >> 241 >> 242 >> 243

flow will be added in table 212/242, and the match condition of ACL4 flows will be modified as noted above in the proposed change:

Table	Match	Action
Dispatcher	metadata=service_id:ACL	write_metadata:(elan_id=ELAN, service_id=NEXT), goto_table:210/240 (ACL1)
ACL1 (210/240)		goto_table:ACL2
...		
ACL2 (211/241)		goto_table:ACL3
ACL3 (212/242)	ip,ct_mark=0x1,reg6=0x200/0xffffff00	(set_field:0x0->ct_mark), goto_table:ACL4
ACL3 (212/242)		goto_table:ACL4
ACL4 (213/243)	ct_state=-new+est-rel-inv+trk,ct_mark=0x1	resubmit(,DISPATCHER)
ACL4 (213/243)	ct_state=+trk,priority=3902,ip,reg6=0x200/0xffffff00	set_field:0x1->ct_mark, resubmit(,DISPATCHER)
ACL4 (213/243)	ct_state=+trk, reg6=0x200/0xffffff00	drop
...		

Yang changes

The nicira-action.yang and the openflowplugin-extension-nicira-action.yang needs to be updated with ct_mark action. The action structure shall be

```
grouping ofj-nx-action-conntrack-grouping {
    container nx-action-conntrack {
        leaf flags {
            type uint16;
        }
        leaf zone-src {
            type uint32;
        }
        leaf conntrack-zone {
            type uint16;
        }
        leaf recirc-table {
            type uint128;
        }
        leaf experimenter-id {
            type oft:experimenter-id;
        }
        list ct-actions{
            uses ofpact-actions;
        }
    }
 }
```

The nicira-match.yang and the openflowplugin-extension-nicira-match.yang needs to be updated with the ct_mark match.

```
grouping ofj-nxm-nx-match-ct-mark-grouping{
       container ct-mark-values {
          leaf ct-mark {
             type uint32;
          }
           leaf mask {
             type uint32;
          }
      }
  }
```

Configuration impact

None.

Clustering considerations

None.

Other Infra considerations

None.

Security considerations

None.

Scale and Performance Impact

When we delete the SG rule from the VM, A new flow will be added in the flow table 212 to flip the value of ct_mark of ongoing traffics. This flow will have a time out based on the protocol as mentioned in the proposed changes section. The packets of ongoing traffic will be recommitted and will do the set filed of ct_mark until the flow reaches the time out.

Targeted Release

Carbon

Alternatives

While deleting a SG flow from the flow table, we will add a DROP flow with the highest priority in the ACL4 table. This DROP flow will drop the packets and it will stop the existing traffic. Similarly, when we restore the same rule again, we will delete the DROP flow from the ACL4 table which will enable the existing traffic.

But this approach will be effective only if the VM do not have any duplicate flows. With the current ACL implementation, if we associate two SGs which having similar set of SG rule, netvirt will install the two set of flows with different priority for the same VM.

As per above approach, if we dissociate any one of SG from the VM, It will add the DROP flow in ACL4 table which will stops the existing traffic irrespective of there is still another flow available in ACL4, to make the traffic possible.

Usage

Traffic between VMs will work accordance with the SG flow existence in the flow table.

Features to Install

Install the ODL Karaf feature for NetVirt (no change):

- odl-netvirt-openstack

REST API

None.

CLI

Refer to the Neutron CLI Reference[1] for the Neutron CLI command syntax for managing Security Rules.

[1] Neutron Security Groups http://docs.openstack.org/user-guide/cli-nova-configure-access-security-for-instances.html

Implementation

Assignee(s)

Who is implementing this feature? In case of multiple authors, designate a primary assignee and other contributors.

Primary assignee:

- VinothB <vinothb@hcl.com>
- Balakrishnan Karuppasamy <balakrishnan.ka@hcl.com>

Other contributors:

- ?

Work Items

None

Dependencies

None.

Testing

Unit Tests

Integration Tests

CSIT

We should add tests verifying ACL change reflection on existing traffic. There should be at least:

- One security rule allowing ICMP traffic between VMs in the same SG.
- One positive test, checking ICMP connectivity works between two VMs using the same SG. Delete all the rules from the SG without disturbing the already established traffic. It should stop the traffic.
- One positive test, checking ICMP connectivity works between two VMs,one using the SG, configured with the ICMP rule, Delete and restore the ICMP rule immediately. This should stop and resume the ICMP traffic after restoring the ICMP rule.
- One positive test, checking ICMP connectivity between VMs, using the SG, configured with ICMP ALL and Other protocol ANY rule. Delete the ICMP rule from the SG, It should not stop the ICMP traffic.
- One negative test, checking ICMP connectivity between two VMs, one using the SG, configured with the ICMP and TCP rules above, and delete the TCP rule. This should not affect the ICMP traffic.

Documentation Impact

None.

References

Table of Contents

Conntrack Based SNAT

https://git.opendaylight.org/gerrit/#/q/topic:snat_conntrack

The ovs conntrack based SNAT implements Source Network Address Translation using openflow rules by leveraging ovs-netfilter integration.

Problem description

Today SNAT is done in Opendaylight netvirt using controller punting and thus controller installing the rules for inbound and outbound NAPT. This causes significant delay as the first packet of all the new connections needs to go through the controller.The number of flows grows linearly with the increase in the vms. Also the current implementation does not support ICMP.

The current algorithm for selecting the NAPT switch does not work well with conntrack based SNAT. For a NAPT switch to remain as designated NAPT switch, it requires at least one port from any of the subnets present in the router. When such a port cease to exist a new NAPT switch will be elected. With the controller based implementation the failover is faster as the NAT flows are reinstalled to the new NAPT switch and should not lead to termination of existing connection. With the conntrack based approach, the translation will be lost and the newly elected switch will have to redo the translation. This will lead to connection timeout for TCP like connections. So the re-election needs to be prevented unless switch is down. Also the current implementation tends to select the node running the DHCP agent as the designated NAPT switch as the DHCP port is the first port created for a subnet.

Use Cases

The following use case will be realized by the implementation

External Network Access The SNAT enables the VM in a tenant network access the external network without using a floating ip. It uses NAPT for sharing the external ip address across multiple VMs that share the same router gateway.

Proposed change

The proposed implementation uses linux netfilter framework to do the NAPT (Network Address Port Translation) and for tracking the connection. The first packet of a traffic will be committed to the netfilter for translation along with the external ip. The subsequent packets will use the entry in the netfilter for inbound and outbound translation. The router id will be used as the zone id in the netfilter. Each zone tracks the connection in its own table. The rest of the implementation for selecting the designated NAPT switch and non designated switches will remain the same. The pipeline changes will happen in the designated switch. With this implementation we will be able to do translation for icmp as well.

The openflow plugin needs to support new set of actions for conntrack based NAPT. This shall be added in the nicira plugin extension of OpenFlow plugin.

The new implementation will not re-install the existing NAT entries to the new NAPT switch during fail-over. Also spec does not cover the use case of having multiple external subnets in the same router.

The HA framework will have a new algorithm to elect the designated NAPT switch. The new logic will be applicable only if the conntrack mode is selected. The switch selection logic will also be modified to use round robin logic with weights associated with each switch. It will not take into account whether a port belonging to a subnet in the router is present in the switch. The initial weight of all the switches shall be 0 and will be incremented by 1 when the switch is selected as the designated NAPT. The weights shall be decremented by 1 when the router is deleted. At any point of time the switch with the lowest weight will be selected as the designated NAPT switch for a new router. If there are multiple the first one with the lowest weight will be selected. A pseudo port will be added in the switch which is selected as the designated NAPT switch. This port will be deleted only when the switch cease to be a designated

NAPT switch. This helps the switch to maintain the remote flows even when there are no ports in the router subnet in the switch. Only if the switch hosting the designated NAPT switch is down a new NAPT switch will be elected.

Pipeline changes

The ovs based NAPT flows will replace the controller based NAPT flows. The changes are limited to the designated switch for the router. Below is the illustration for flat external network.

Outbound NAPT

Table 26 (PSNAT Table) => submits the packet to netfilter to check whether it is an existing connection. Resubmits the packet back to 46.

Table 46 (NAPT OUTBOUND TABLE) => if it is an established connection, it indicates the translation is done and the packet is forwarded to table 47 after writing the external network metadata.

If it is a new connection the connection will be committed to netfilter and this entry will be used for NAPT. The translated packet will be resubmitted to table 47. The external network metadata will be written before sending the packet to netfilter.

Table 47 (NAPT FIB TABLE) => The translated packet will be sent to the egress group.

Sample Flows

```
table=26, priority=5,ip,metadata=0x222e2/0xfffffffe actions=ct(table=46,zone=5003,nat)
table=46, priority=6,ct_state=+snat,ip,metadata=0x222e2/0xfffffffe actions=set_
↪field:0x222e0->metadata,resubmit(,47)
table=46, priority=5,ct_state=+new+trk,ip,metadata=0x222e2/0xfffffffe actions=set_
↪field:0x222e0->metadata,ct(commit,table=47,zone=5003,nat(src=192.168.111.21))
table=47, n_packets=0, n_bytes=0, priority=6,ct_state=+snat,ip,nw_src=192.168.111.21␣
↪actions=group:200000
```

Inbound NAPT

Table 44 (NAPT INBOUND Table)=> submits the packet to netfilter to check for an existing connection after changing the metadata to that of the internal network. The packet will be submitted back to table 47.

Table 47 (NAPT FIB TABLE) => The translated packet will be submitted back to table 21.

Sample Flows

```
table=21, priority=42,ip,metadata=0x222e0/0xfffffffe,nw_dst=192.168.111.21␣
↪actions=resubmit(,44)
table=44, priority=10,ip,metadata=0x222e0/0xfffffffe,nw_dst=192.168.111.21␣
↪actions=set_field:0x222e2->metadata,ct(table=47,zone=5003,nat)
table=47, priority=5,ct_state=+dnat,ip actions=resubmit(,21)
```

Yang changes

The nicira-action.yang and the openflowplugin-extension-nicira-action.yang needs to be updated with nat action. The action structure shall be

```
typedef nx-action-nat-range-present {
    type enumeration {
        enum NX_NAT_RANGE_IPV4_MIN {
            value 1;
            description "IPV4 minimum value is present";
        }
```

```
        enum NX_NAT_RANGE_IPV4_MAX {
            value 2;
            description "IPV4 maximum value is present";
        }
        enum NX_NAT_RANGE_IPV6_MIN {
            value 4;
            description "IPV6 minimum value is present in range";
        }
        enum NX_NAT_RANGE_IPV6_MAX {
            value 8;
            description "IPV6 maximum value is present in range";
        }
        enum NX_NAT_RANGE_PROTO_MIN {
            value 16;
            description "Port minimum value is present in range";
        }
        enum NX_NAT_RANGE_PROTO_MAX {
            value 32;
            description "Port maximum value is present in range";
        }
    }
 }

typedef nx-action-nat-flags {
    type enumeration {
        enum NX_NAT_F_SRC {
            value 1;
            description "Source nat is selected ,Mutually exclusive with NX_NAT_F_DST
↪";
        }
        enum NX_NAT_F_DST {
            value 2;
            description "Destination nat is selected";
        }
        enum NX_NAT_F_PERSISTENT {
            value 4;
            description "Persistent flag is selected";
        }
        enum NX_NAT_F_PROTO_HASH {
            value 8;
            description "Hash mode is selected for port mapping, Mutually exclusive␣
↪with
            NX_NAT_F_PROTO_RANDOM ";
        }
        enum NX_NAT_F_PROTO_RANDOM {
            value 16;
            description "Port mapping will be randomized";
        }
    }
 }

grouping ofj-nx-action-conntrack-grouping {
    container nx-action-conntrack {
        leaf flags {
            type uint16;
        }
        leaf zone-src {
            type uint32;
```

```
        }
        leaf conntrack-zone {
            type uint16;
        }
        leaf recirc-table {
            type uint8;
        }
        leaf experimenter-id {
            type oft:experimenter-id;
        }
        list ct-actions{
            uses ofpact-actions;
        }
    }
 }

grouping ofpact-actions {
    description
       "Actions to be performed with conntrack.";
    choice ofpact-actions {
         case nx-action-nat-case {
            container nx-action-nat {
                leaf flags {
                    type uint16;
                }
                leaf range_present {
                    type uint16;
                }
                leaf ip-address-min {
                    type inet:ip-address;
                }
                leaf ip-address-max {
                    type inet:ip-address;
                }
                leaf port-min {
                    type uint16;
                }
                leaf port-max {
                    type uint16;
                }
            }
        }
    }
}
```

For the new configuration knob a new yang natservice-config shall be added in NAT service, with the container for holding the NAT mode configured. It will have two options controller and conntrack, with controller being the default.

```
container natservice-config {
  config true;
  leaf nat-mode {
      type enumeration {
          enum "controller";
          enum "conntrack";
      }
      default "controller";
  }
}
```

Configuration impact

The proposed change requires the NAT service to provide a configuration knob to switch between the controller based/conntrack based implementation. A new configuration file netvirt-natservice-config.xml shall be added with default value controller.

```
<natservice-config xmlns="urn:opendaylight:netvirt:natservice-config">
  <nat-mode>controller</nat-mode>
</natservice-config>
```

The dynamic update of nat-mode will not be supported. To change the nat-mode the controller cluster needs to be restarted after changing the nat-mode. On restart the NAT translation lifecycle will be reset and after the controller comes up in the updated nat-mode, a new set of switches will be elected as designated NAPT switches and it can be different from the ones that were forwarding traffic earlier.

Clustering considerations

NA

Other Infra considerations

The implementation requires ovs2.6 with the kernel module installed. OVS currently does not support SNAT connection tracking for dpdk datapath. It would be supported in some future release.

Security considerations

NA

Scale and Performance Impact

The new SNAT implementation is expected to improve the performance when compared to the existing one and will reduce the flows in ovs pipeline.

Targeted Release

Carbon

Alternatives

An alternative implementation of X NAPT switches was discussed, which will not be a part of this document but will be considered as a further enhancement.

Usage

Create External Network

Create an external flat network and subnet

```
neutron net-create ext1 --router:external  --provider:physical_network public --
↪provider:network_type flat
neutron subnet-create --allocation-pool start=<start-ip>,end=<end-ip> --gateway=<gw-
↪ip> --disable-dhcp --name subext1 ext1 <subnet-cidr>
```

Create Internal Network

Create an internal n/w and subnet

```
neutron net-create vx-net1 --provider:network_type vxlan
neutron subnet-create vx-net1 <subnet-cidr> --name vx-subnet1
```

Create Router

Create a router and add an interface to internal n/w. Set the external n/w as the router gateway.

```
neutron router-create router1
neutron router-interface-add  router1 vx-subnet1
neutron router-gateway-set router1 ext1
nova boot --poll --flavor m1.tiny --image $(nova image-list | grep 'uec\s' | awk '
↪{print $2}' | tail -1) --nic net-id=$(neutron net-list | grep -w vx-net1 | awk '
↪{print $2}') vmvx2
```

Features to Install

odl-netvirt-openstack

REST API

NA

CLI

A new command line, display-napt-switch, will be added to display the current designated NAPT switch selected for each router. It shall show the below info.

```
router id | Host Name of designated NAPT switch | Management Ip of the designated␣
↪NAPT switch
```

Implementation

Assignee(s)

Aswin Suryanarayanan <asuryana@redhat.com>

Work Items

https://trello.com/c/DMLsrLfq/9-snat-decentralized-ovs-nat-based

- Write a framework which can support multiple modes of NAT implementation.
- Add support in openflow plugin for conntrack nat actions.
- Add support in genius for conntrack nat actions.
- Add a config parameter to select between controller based and conntrack based.
- Add the flow programming for SNAT in netvirt.
- Add the new HA framework.
- Add the command to display the designated NAPT switch.
- Write Unit tests for conntrack based snat.

Dependencies

NA

Testing

Unit Tests

Unit test needs to be added for the new snat mode. It shall use the component tests framework

Integration Tests

Integration tests needs to be added for the conntrack snat flows.

CSIT

Run the CSIT with conntrack based SNAT configured.

Documentation Impact

Necessary documentation would be added on how to use this feature.

References

Table of Contents

Cross site connectivity with federation service

https://git.opendaylight.org/gerrit/#/q/topic:federation-plugin

Enabling neutron networks to expand beyond a single OpenStack instance to allow L2 switching and L3 routing between sites. Sites may be geographically remote or partitioned in a single data center.

Each site is deployed with independent local ODL cluster. The clusters communicate using the federation infrastructure [2] in order to publish MDSAL events whenever routable entities e.g. VM instances are added/removed from remote sites.

VxLAN tunnels are used to form the overlay for cross site communication between OpenStack compute nodes.

Problem description

Today, communication between VMs in remote sites is based on BGP control plane and requires DC-GW. Overlay network between data centers is based on MPLSoverGRE or VxLAN if the DC-GW supports EVPN RT5 [4]. The purpose of this feature is to allow inter-DC communication independent from BGP control plane and DC-GW.

Use Cases

This feature will cover the following use cases:

L2 switching use cases

- L2 Unicast frames exchanged between VMs sharing federated neutron network between OVS datapaths in remote sites
- L2 Unicast frames exchanged between VM and PNF sharing federated neutron network between OVS and HWVTEP datapath in remote sites
- L2 Broadcast frames exchanged between VMs sharing federated neutron network between OVS datapaths in remote sites
- L2 Broadcast frames exchanged between VM and PNF sharing federated neutron network between OVS and HWVTEP datapath in remote sites

L3 forwarding use cases

- L3 traffic exchanged between VMs sharing federated neutron router between OVS datapaths in remote sites

Proposed change

For Carbon release, cross-site connectivity will be based on the current HPE downstream federation plugin code-base. This plugin implements the federation service API [3] to synchronize the following MDSAL subtrees between connected sites:

- config/ietf-interfaces:interfaces
- config/elan:elan-interfaces
- config/l3vpn:vpn-interfaces
- config/network-topology:network-topology/topology/ovsdb:1
- operational/network-topology:network-topology/topology/ovsdb:1
- config/network-topology:network-topology/topology/hwvtep:1
- operational/network-topology:network-topology/topology/hwvtep:1
- config/opendaylight-inventory:nodes
- operational/opendaylight-inventory:nodes
- config/neutron:neutron/l2gateways
- config/neutron:neutron/l2gatewayConnections

The provisioning of connected networks between remote sites is out of the scope of this spec and described in [6].

Upon receiving a list of shared neutron networks and subnets, the federation plugin will propagate MDSAL entities from all of the subtrees detailed above to remote sites based on the federation connection definitions. The federated entities will be transformed to match the target network/subnet/router details in each remote site.

For example, ELAN interface will be federated with elan-instance-name set to the remote site elan-instance-name. VPN interface will be federated with the remote site vpn-instance-name i.e. router-id and remote subnet-id contained in the primary VPN interface adjacency.

This would allow remotely federated entities a.k.a shadow entities to be handled the same way local entities are handled thus shadow entities will appear as if they were local entities in remote sites. As a result, the following pipeline elements will be added for shadow entities on all compute nodes in each connected remote site:

- ELAN remote DMAC flow for L2 unicast packets to remote site
- ELAN remote broadcast group buckets for L2 multicast packets to remote site
- FIB remote nexthop flow for L3 packet to remote site

The following limitations exist for the current federation plugin implementation:

- Federated networks use VxLAN network type and the same VNI is used across sites.
- The IP addresses allocated to VM instances in federated subnets do not overlap across sites.
- The neutron-configured VNI will be passed on the wire for inter-DC L2/L3 communication between VxLAN networks. The implementation is described in [5].

As part of Nitrogen, the federation plugin is planned to go through major redesign. The scope and internals have not been finalized yet but this spec might be a good opportunity to agree on an alternate solution.

Some initial thoughts:

- For L3 cross site connectivity, it seems that federating the FIB vrf-entry associated with VMs in connected networks should be sufficient to form remote nexthop connectivity across sites.
- In order to create VxLAN tunnels to remote sites, it may be possible to use the external tunnel concept instead of creating internal tunnels that are dependent on federation of the OVS topology nodes from remote sites.
- L2 cross site connectivity is the most challenging part for federation of MAC addresses of both VM instances and PNFs connected to HWVTEP. If the ELAN model could be enhanced to have remote-mac-entry model containing MAC address, ELAN instance name and remote TEP ip, it would be possible to federate such entity to remote sites in order to create remote DMAC flows for cases of remote VM instances and PNFs connected HWVTEP in remote sites.

Pipeline changes

No new pipeline changes are introduced as part of this feature. The pipeline flow between VM instances in remote sites is similar to the current implementation of cross compute intra-DC traffic since the realization of remote compute nodes is similar to local ones.

Yang changes

The following new yang models will be introduced as part of the federation plugin API bundle:

Federation Plugin Yang

Marking for each federated entity using `shadow-properties` augmentation

```
module federation-plugin {
   yang-version 1;
   namespace "urn:opendaylight:netvirt:federation:plugin";
   prefix "federation-plugin";

   import yang-ext {
        prefix ext;
        revision-date "2013-07-09";
   }

   import ietf-yang-types {
        prefix yang;
   }

   import network-topology {
        prefix topo;
   }

   import opendaylight-inventory {
        prefix inv;
   }

   import ietf-interfaces {
        prefix if;
   }

   import elan {
        prefix elan;
   }

   import l3vpn {
        prefix l3vpn;
   }

   import neutronvpn {
        prefix nvpn;
   }

   revision "2017-02-19" {
       description "Federation plugin model";
   }

   grouping shadow-properties {
        leaf shadow {
            type boolean;
            description "Represents whether this is a federated entity";
        }
        leaf generation-number {
            type int32;
            description "The current generation number of the federated entity";
        }
        leaf remote-ip {
            type string;
            description "The IP address of the original site of the federated entity";
```

```
        }
    }

    augment "/topo:network-topology/topo:topology/topo:node" {
        ext:augment-identifier "topology-node-shadow-properties";
        uses shadow-properties;
    }

    augment "/inv:nodes/inv:node" {
        ext:augment-identifier "inventory-node-shadow-properties";
        uses shadow-properties;
    }

    augment "/if:interfaces/if:interface" {
        ext:augment-identifier "if-shadow-properties";
        uses shadow-properties;
    }

    augment "/elan:elan-interfaces/elan:elan-interface" {
        ext:augment-identifier "elan-shadow-properties";
        uses shadow-properties;
    }

    augment "/l3vpn:vpn-interfaces/l3vpn:vpn-interface" {
        ext:augment-identifier "vpn-shadow-properties";
        uses shadow-properties;
    }
}
```

Federation Plugin Manager Yang

Management of federated networks and routed RPCs subscription

```
module federation-plugin-manager {
    yang-version 1;
    namespace "urn:opendaylight:netvirt:federation:plugin:manager";
    prefix "federation-plugin-manager";

    import yang-ext {
        prefix ext;
        revision-date "2013-07-09";
   }

   import ietf-yang-types {
        prefix yang;
   }

   revision "2017-02-19" {
       description "Federation plugin model";
   }

    identity mgr-context {
        description "Identity for a routed RPC";
    }

    container routed-container {
```

```
        list route-key-item {
            key "id";
            leaf id {
                type string;
            }

            ext:context-instance "mgr-context";
        }
    }

    container federated-networks {
        list federated-network {
            key self-net-id;
            uses federated-nets;
        }
    }

    container federation-generations {
        description
                "Federation generation information for a remote site.";
        list remote-site-generation-info {
            max-elements "unbounded";
            min-elements "0";
            key "remote-ip";
            leaf remote-ip {
                mandatory true;
                type string;
                description "Remote site IP address.";
            }
            leaf generation-number {
                type int32;
                description "The current generation number used for the remote site.";
            }
        }
    }

    grouping federated-nets {
        leaf self-net-id {
            type string;
            description "UUID representing the self net";
        }
        leaf self-subnet-id {
            type yang:uuid;
            description "UUID representing the self subnet";
        }
        leaf self-tenant-id {
            type yang:uuid;
            description "UUID representing the self tenant";
        }
        leaf subnet-ip {
            type string;
            description "Specifies the subnet IP in CIDR format";
        }

        list site-network {
            key id;
            leaf id {
                type string;
```

```
                description "UUID representing the site ID (from xsite manager)";
            }
            leaf site-ip {
                type string;
                description "Specifies the site IP";
            }
            leaf site-net-id {
                type string;
                description "UUID of the network in the site";
            }
            leaf site-subnet-id {
                type yang:uuid;
                description "UUID of the subnet in the site";
            }
            leaf site-tenant-id {
                type yang:uuid;
                description "UUID of the tenant holding this network in the site";
            }
        }
    }
}
```

Federation Plugin RPC Yang

FederationPluginRpcService yang definition for `update-federated-networks` RPC

```
module federation-plugin-rpc {
   yang-version 1;
   namespace "urn:opendaylight:netvirt:federation:plugin:rpc";
   prefix "federation-plugin-rpc";

   import yang-ext {
        prefix ext;
        revision-date "2013-07-09";
  }

  import ietf-yang-types {
        prefix yang;
  }

  import federation-plugin-manager {
        prefix federation-plugin-manager;
  }

  revision "2017-02-19" {
      description "Federation plugin model";
  }

   rpc update-federated-networks {
        input {
            list federated-networks-in {
                key self-net-id;
                uses federation-plugin-manager:federated-nets;
                description "Contain all federated networks in this site that are still
                             connected, a federated network that does not appear will␣
↪be considered
```

```
                            disconnected";
            }
        }
    }
}
```

Federation Plugin routed RPC Yang

Routed RPCs will be used only within the cluster to route connect/disconnect requests to the federation cluster singleton.

```
module federation-plugin-routed-rpc {
    yang-version 1;
    namespace "urn:opendaylight:netvirt:federation:plugin:routed:rpc";
    prefix "federation-plugin-routed-rpc";

    import yang-ext {
        prefix ext;
        revision-date "2013-07-09";
    }

    import ietf-yang-types {
        prefix yang;
    }

    import federation-plugin-manager {
        prefix federation-plugin-manager;
    }

    revision "2017-02-19" {
        description "Federation plugin model";
    }

    rpc update-federated-networks {
        input {
            leaf route-key-item {
                type instance-identifier;
                ext:context-reference federation-plugin-manager:mgr-context;
            }

            list federated-networks-in {
                key self-net-id;
                uses federation-plugin-manager:federated-nets;
            }
        }
    }
}
```

Configuration impact

None.

Clustering considerations

The federation plugin will be active only on one of the ODL instances in the cluster. The cluster singleton service infrastructure will be used in order to register the federation plugin routed RPCs only on the selected ODL instance.

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

None

Targeted Release

Carbon

Alternatives

None

Usage

Features to Install

odl-netvirt-federation

This is a new feature that will load odl-netvirt-openstack and the federation service features. It will not be installed by default and requires manual startup using karaf `feature:install` command.

REST API

Connecting neutron networks from remote sites

URL: restconf/operations/federation-plugin-manager:update-federated-networks

Sample JSON data

```
{
   "input": {
       "federated-networks-in": [
           {
               "self-net-id": "c4976ee7-c5cd-4a5e-9cf9-261f28ba7920",
               "self-subnet-id": "93dee7cb-ba25-4318-b60c-19a15f2c079a",
```

```
                "subnet-ip": "10.0.123.0/24",
                "site-network": [
                    {
                        "id": "c4976ee7-c5cd-4a5e-9cf9-261f28ba7922",
                        "site-ip": "10.0.43.146",
                        "site-net-id": "c4976ee7-c5cd-4a5e-9cf9-261f28ba7921",
                        "site-subnet-id": "93dee7cb-ba25-4318-b60c-19a15f2c079b",
                    }
                ]
            }
        ]
    }
}
```

CLI

None.

Implementation

Assignee(s)

Primary assignee: Tali Ben-Meir <tali@hpe.com>

Other contributors: Guy Sela <guy.sela@hpe.com>

Shlomi Alfasi <shlomi.alfasi@hpe.com>

Yair Zinger <yair.zinger@hpe.com>

Work Items

Trello card https://trello.com/c/mgdUO6xx/154-federation-plugin-for-netvirt

Since the code was already implemented in downstream no work items will be defined

Dependencies

This feature will be implemented in 2 new bundles - `federation-plugin-api` and `federation-plugin-impl` the implementation will be dependent on `federation-service-api` [3] bundle from OpenDaylight federation project.

The new karaf feature `odl-netvirt-federation` will encapsulate the federation-plugin api and impl bundles and will be dependant on the followings features:

- `federation-with-rabbit` from federation project
- `odl-netvirt-openstack` from netvirt project

Testing

Unit Tests

End-to-end component service will test the federation plugin on top of the federation service.

Integration Tests

None

CSIT

The CSIT infrastructure will be enhanced to support connect/disconnect operations between sites using update-federated-networks RPC call.

A new federation suite will test L2 and L3 connectivity between remote sites and will be based on the existing L2/L3 connectivity suites. CSIT will load sites A,B and C in 1-node/3-node deployment options to run the following tests:

1 Install odl-netvirt-federation feature

- Basic L2 connectivity test within the site
- Basic L3 connectivity test within the site
- L2 connectivity between sites - expected to fail since sites are not connected
- L3 connectivity between sites - expected to fail since sites are not connected

2 Connect sites A,B

- Basic L2 connectivity test within the site
- L2 connectivity test between VMs in sites A,B
- L2 connectivity test between VMs in sites A,C and B,C - expected to fail since sites are not connected
- Basic L3 connectivity test within the site
- L3 connectivity test between VMs in sites A,B
- L3 connectivity test between VMs in sites A,C and B,C - expected to fail since sites are not connected

3 Connect site C to A,B

- L2 connectivity test between VMs in sites A,B B,C and A,C
- L3 connectivity test between VMs in sites A,B B,C and A,C
- Connectivity test between VMs in non-federated networks in sites A,B,C - expected to fail

4 Disconnect site C from A,B

- Repeat the test steps from 2 after C disconnect. Identical results expected.

5 Disconnect sites A,B

- Repeat the test steps from 1 after A,B disconnect. Identical results expected.

6 Federation cluster test

- Repeat test steps 1-5 while rebooting the ODLs between the steps similarly to the existing cluster suite.

Documentation Impact

None.

References

[1] OpenDaylight Documentation Guide

[2] Federation project

[3] Federation service API

[4] Support of VxLAN based connectivity across Datacenters

[5] VNI based L2 switching, L3 forwarding and NATing

[6] Cross site manager presentation ODL Summit 2016

Table of Contents

DHCP Server Dynamic Allocation Pool

[gerrit filter: https://git.opendaylight.org/gerrit/#/q/topic:dhcp_server_pool]

Extension of the ODL based DHCP server, which add support for dynamic address allocation to end point users, that are not controlled (known) by OpenStack Neutron. Each DHCP pool can be configured with additional information such as DNS servers, lease time (not yet), static allocations based on MAC address, etc.

The feature supports IPv4 only.

Problem description

In a non-neutron northbounds environment e.g. SD-WAN solution (unimgr), there is currently no dynamic DHCP service for end-points or networks that are connected to OVS. Every DHCP packet that is received by the controller, the controller finds the neutron port based on the inport of the packet, extracts the ip which was allocated by neutron for that vm, and replies using that info. If the dhcp packet is from a non-neutron port, the packet won't even reach the controller.

Use Cases

a DHCP packet that is received by the odl, from a port that is managed by Netvirt and was configured using the netvirt API, rather then the neutron API, in a way that there is no pre-allocated IP for network interfaces behind that port - will be handled by the DHCP dynamic allocation pool that is configured on the network associated with the receiving OVS port.

Proposed change

We wish to forward to the controller, every dhcp packet coming from a non-neutron port as well (as long as it is configured to use the controller dhcp). Once a DHCP packet is recieved by the controller, the controller will check if there is already a pre-allocated address by checking if packet came from a neutron port. if so, the controller will reply using the information from the neutron port. Otherwise, the controller will find the allocation pool for the network which the packet came from and will allocate the next free ip. The operation of each allocation pool will be managed through the Genius ID Manager service that will support the allocation and release of IP addresses (ids), persistent

mapping across controller restarts and more. Neutron IP allocations will be added to the relevant pools to avoid allocation of the same addresses.

The allocation pool DHCP server will support:

- DHCP methods: Discover, Request, Release, Decline and Inform (future)
- Allocation of a dynamic or specific (future) available IP address from the pool
- (future) Static IP address allocations
- (future) IP Address Lease Time + Rebinding and Renewal Time
- Classless Static Routes for each pool
- Domain names (future) and DNS for each pool
- (future) Probe an address before allocation
- (future) Relay agents

Pipeline changes

This new rule in table 60 will be responsible for forwarding dhcp packets to the controller:

```
cookie=0x6800000, duration=121472.576s, table=60, n_packets=1, n_bytes=342,␣
↪priority=49,udp,tp_src=68,tp_dst=67 actions=CONTROLLER:65535
```

Yang changes

New YANG model to support the configuration of the DHCP allocation pools and allocations, per network and subnet.

- Allocation-Pool: configuration of allocation pool parameters like range, gateway and dns servers.
- Allocation-Instance: configuration of static IP address allocation and Neutron pre-allocated addresses, per MAC address.

Listing 1.3: dhcp_allocation_pool.yang

```
container dhcp_allocation_pool {
    config true;
    description "contains DHCP Server dynamic allocations";

    list network {
        key "network-id";
        leaf network-id {
            description "network (elan-instance) id";
            type string;
        }
        list allocation {
            key "subnet";
            leaf subnet {
                description "subnet for the dhcp to allocate ip addresses";
                type inet:ip-prefix;
            }

            list allocation-instance {
                key "mac";
                leaf mac {
```

```
                    description "requesting mac";
                    type yang:phys-address;
                }
                leaf allocated-ip {
                    description "allocated ip address";
                    type inet:ip-address;
                }
            }
        }
        list allocation-pool {
            key "subnet";
            leaf subnet {
                description "subnet for the dhcp to allocate ip addresses";
                type inet:ip-prefix;
            }
            leaf allocate-from {
                description "low allocation limit";
                type inet:ip-address;
            }
            leaf allocate-to {
                description "high allocation limit";
                type inet:ip-address;
            }
            leaf gateway {
                description "default gateway for dhcp allocation";
                type inet:ip-address;
            }
            leaf-list dns-servers {
                description "dns server list";
                type inet:ip-address;
            }
            list static-routes {
                description "static routes list for dhcp allocation";
                key "destination";
                leaf destination {
                    description "destination in CIDR format";
                    type inet:ip-prefix;
                }
                leaf nexthop {
                    description "router ip address";
                    type inet:ip-address;
                }
            }
        }
    }
}
```

Configuration impact

The feature is activated in the configuration (disabled by default).

adding **dhcp-dynamic-allocation-pool-enabled** leaf to dhcpservice-config:

Listing 1.4: dhcpservice-config.yang

```
container dhcpservice-config {
    leaf controller-dhcp-enabled {
        description "Enable the dhcpservice on the controller";
        type boolean;
        default false;
    }

    leaf dhcp-dynamic-allocation-pool-enabled {
        description "Enable dynamic allocation pool on controller dhcpservice";
        type boolean;
        default false;
    }
}
```

and netvirt-dhcpservice-config.xml:

```
<dhcpservice-config xmlns="urn:opendaylight:params:xml:ns:yang:dhcpservice:config">
  <controller-dhcp-enabled>false</controller-dhcp-enabled>
  <dhcp-dynamic-allocation-pool-enabled>false</dhcp-dynamic-allocation-pool-enabled>
</dhcpservice-config>
```

Clustering considerations

Support clustering.

Other Infra considerations

None.

Security considerations

None.

Scale and Performance Impact

None.

Targeted Release

Carbon.

Alternatives

Implement and maintain an external DHCP server.

Usage

Features to Install

This feature can be used by installing odl-netvirt-openstack. This feature doesn't add any new karaf feature.

REST API

Introducing a new REST API for the feature

Dynamic allocation pool

URL: /config/dhcp_allocation_pool:dhcp_allocation_pool/

Sample JSON data

```
{"dhcp_allocation_pool": {
  "network": [
    {
      "network-id": "d211a14b-e5e9-33af-89f3-9e43a270e0c8",
      "allocation-pool": [
        {
          "subnet": "10.1.1.0/24",
          "dns-servers": [
            "8.8.8.8"
          ],
          "gateway": "10.1.1.1",
          "allocate-from": "10.1.1.2",
          "allocate-to": "10.1.1.200"
          "static-routes": [
            {
              "destination": "5.8.19.24/16",
              "nexthop": "10.1.1.254"
            }
          ]
]}]}}
```

Static address allocation

URL: /config/dhcp_allocation_pool:dhcp_allocation_pool/

Sample JSON data

```
{"dhcp_allocation_pool": {
  "network": [
    {
      "network-id": "d211a14b-e5e9-33af-89f3-9e43a270e0c8",
      "allocation": [
        {
          "subnet": "10.1.1.0/24",
          "allocation-instance": [
            {
              "mac": "fa:16:3e:9d:c6:f5",
              "allocated-ip": "10.1.1.2"
```

```
            }
]}]}]}]}
```

CLI

None.

Implementation

Assignee(s)

Primary assignee: Shai Haim (shai.haim@hpe.com)

Other contributors: Alex Feigin (alex.feigin@hpe.com)

Work Items

Here is the link for the Trello Card: https://trello.com/c/0mgGyJuV/153-dhcp-server-dynamic-allocation-pool

Dependencies

None.

Testing

Unit Tests

N.A.

Integration Tests

N.A.

CSIT

N.A.

Documentation Impact

??

References

Table of Contents

Discovery of directly connected PNFs in Flat/VLAN provider networks

https://git.opendaylight.org/gerrit/#/q/topic:directly_connected_pnf_discovery

This features enables discovering and directing traffic to Physical Network Functions (PNFs) in Flat/VLAN provider and tenant networks, by leveraging Subnet-Route feature.

Problem description

PNF is a device which has not been created by Openstack but connected to the hypervisors L2 broadcast domain and configured with ip from one of the neutron subnets.

Ideally, L2/L3 communication between VM instances and PNFs on flat/VLAN networks would be routed similarly to inter-VM communication. However, there are two main issues preventing direct communication to PNFs.

- L3 connectivity of tenant network and VLAN provider network, between VMs and PNFs. A VM is located in a tenant network, A PNF is located in a provider network (external network). Both networks are connected via a router. The only way for VMs to communicate with a PNF is via additional hop which is the external gateway, instead of directly.
- L3 connectivity between VMs and PNFs in a two diffrent tenant networks, connected by a router, which is not supported and have two problems. First, traffic initiated from a VMs towards a PNF is dropped because there isn't an appropriate rule in FIB table (table 21) to route that traffic. Second, in the other direction, PNFs are not able to resolve their default gateway.

We want to leverage the Subnet-Route and Aliveness-Monitor features in order to address the above issues.

Subnet-Route

Today, Subnet-Route feature enables ODL to route traffic to a destination IP address, even for ip addresses that have not been statically configured by OpenStack, in the FIB table. To achieve that, the FIB table contains a flow that match all IP packets in a given subnet range. How that works?

- A flow is installed in the FIB table, matching on subnet prefix and vpn-id of the network, with a goto-instruction directing packets to table 22. There, packets are punted to the controller.
- ODL hold the packets, and initiate an ARP request towards the destination IP.
- Upon receiving ARP reply, ODL installs exact IP match flow in FIB table to direct all further traffic towards the newly learnt MAC of the destination IP

Current limitations of Subnet-Route feature:

- Works for BGPVPN only
- May cause traffic lost due to "swallowing" the packets punted from table 22.
- Uses the source MAC and source IP from the punted packet.

Aliveness monitor

After ODL learns a mac that is associated with an ip address, ODL schedule an arp monitor task, with the purpose of verifying that the device is still alive and responding. This is done by periodically sending arp requests to the device.

Current limitation: Aliveness monitor was not designed for monitoring devices behind flat/VLAN provider network ports.

Use Cases

- **L3 connectivity of tenant network and VLAN provider network, between VMs and PNFs.**
  - VMs in a private network, PNFs in external network
- L3 connectivity between VMs and PNFs in a two diffrent tenant networks.

Proposed change

Subnet-route

- Upon OpenStack configuration of a Subnet in a provider network, a new vrf entry with subnet-route augmentation will be created.
- Upon associataion of neutron router with a subnet in a tenant network, a new vrf entry with subnet-route augmentation will be created.
- Upon receiving ARP reply, install exact IP match flow in FIB table to direct all further traffic towards the newly resolved PNF, on all relevant computes nodes, which will be discussed later
- Packets that had been punted to controller will be resubmitted to the openflow pipeline after installation of exact match flow.

Communication between VMs in tenant networks and PNFs in provider networks.

In this scenario a VM in a private tenant network wants to communicate with a PNF in the (external) provider network

- The controller will hold the packets, and initiate an ARP request towards the PNF IP. an ARP request will have source MAC and IP the router gateway and will be sent from the NAPT switch.
- ARP packets will be punted from the NAPT switch only.
- Upon receiving ARP reply, install exact IP match flow in FIB table to direct all further traffic towards the newly resolved PNF, on all compute nodes that are associated with the external network.
- leveraging Aliveness monitor feature to monitor PNFs. The controller will send ARP requests from the NAPT switch.

Communication between VMs and PNFs in different tenant networks.

In this scenario a VM and a PNF, in different private networks of the same tenant, wants to communicate. For each subnet prefix, a designated switch will be chosen to communicate directly with the PNFs in that subnet prefix. That means sending ARP requests to the PNFs and receiving their traffic.

Note: IP traffic from VM instances will retain the src MAC of the VM instance, instead of replacing it with the router-interface-mac, in order to prevent MAC momvements in the underlay switches. This is a limitation until NetVirt supports a MAC per hypervisor implementation.

- A subnet flow will be installed in the FIB table, matching the subnet prefix and vpn-id of the router.
- ARP request will have a source MAC and IP of the router interface, and will be sent via the provider port in the designated switch.
- ARP packets will be punted from the designated switch only.
- Upon receiving an ARP reply, install exact IP match flow in FIB table to direct all further traffic towards the newly resolved PNF, on all computes related to the router
- ARP responder flow: a new ARP responder flow will be installed in the designated switch This flow will response for ARP requests from a PNF and the response MAC will be the router interface MAC. This flow will use the LPort-tag of the provider port.
- Split Horizon protection disabling: traffic from PNFs, arrives to the primary switch(via a provider port) due to the ARP responder rule described above, and will need to be directed to the proper compute of the designated VM (via a provider port). This require disabling the split horizon protection. In order to protects against infinite loops, the packet TTL will be decreased.
- leveraging Aliveness monitor, the controller will send ARP requests from the designated switch.

ARP messages

ARP messages in the Flat/Vlan provider and tenant networks will be punted from a designated switch, in order to avoid a performance issue in the controller, of dealing with broadcast packets that may be received in multiple provider ports. In external networks this switch is the NAPT switch.

Pipeline changes

First use-case depends on hairpinning spec [2], the flows presented here reflects that dependency.

Egress traffic from VM with floating IP to an unresolved PNF in external network

- Packets in FIB table after translation to FIP, will match on subnet flow and will be punted to controller from Subnet Route table. Then, ARP request will be generated and be sent to the PNF. No flow changes are required in this part.

  Classifier table (0) =>
  Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
  GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
  FIB table (21) `match: vpn-id=router-id` =>
  Pre SNAT table (26) `match: vpn-id=router-id,src-ip=vm-ip set vpn-id=ext-subnet-id,src-ip=fip` =>
  SNAT table (28) `match: vpn-id=ext-subnet-id,src-ip=fip set src-mac=fip-mac` =>
  FIB table (21) `match: vpn-id=ext-subnet-id, dst-ip=ext-subnet-ip` =>
  Subnet Route table (22): => Output to Controller

- After receiving ARP response from the PNF a new exact IP flow will be installed in table 21. No other flow changes are required.

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id` =>
Pre SNAT table (26) `match: vpn-id=router-id,src-ip=vm-ip set vpn-id=ext-subnet-id,src-ip=fip` =>
SNAT table (28) `match: vpn-id=ext-subnet-id,src-ip=fip set src-mac=fip-mac` =>
FIB table (21) `match: vpn-id=ext-subnet-id, dst-ip=pnf-ip, set dst-mac=pnf-mac, reg6=provider-lport-tag` =>
Egress table (220) output to provider port

Egress traffic from VM using NAPT to an unresolved PNF in external network

- Ingress-DPN is not the NAPT switch, no changes required. Traffic will be directed to NAPT switch and directed to the outbound NAPT table straight from the internal tunnel table

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id` =>
Pre SNAT table (26) `match: vpn-id=router-id` =>
NAPT Group `output to tunnel port of NAPT switch`

- Ingress-DPN is the NAPT switch. Packets in FIB table after translation to NAPT, will match on subnet flow and will be punted to controller from Subnet Route table. Then, ARP request will be generated and be sent to the PNF. No flow changes are required.

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id` =>
Pre SNAT table (26) `match: vpn-id=router-id` =>
Outbound NAPT table (46) `match: src-ip=vm-ip,port=int-port set src-ip=router-gw-ip,vpn-id=router-gw-subnet-id,port=ext-port` =>
NAPT PFIB tabl (47) `match: vpn-id=router-gw-subnet-id` =>
FIB table (21) `match: vpn-id=ext-subnet-id, dst-ip=ext-subnet-ip` =>
Subnet Route table (22) => Output to Controller

- After receiving ARP response from the PNF a new exact IP flow will be installed in table 21. No other changes required.

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id` =>
Pre SNAT table (26) `match: vpn-id=router-id` =>
Outbound NAPT table (46) `match: vpn-id=router-id TBD set vpn-id=external-net-id` =>
NAPT PFIB table (47) `match: vpn-id=external-net-id` =>
FIB table (21) `match: vpn-id=ext-network-id, dst-ip=pnf-ip set dst-mac=pnf-mac, reg6=provider-lport-tag` =>
Egress table (220) output to provider port

Egress traffic from VM in private network to an unresolved PNF in another private network

- Packet from a VM is punted to the controller, no flow changes are required.

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id dst-ip=subnet-ip` =>
Subnet Route table (22): => Output to Controller

- After receiving ARP response from the PNF a new exact IP flow will be installed in table 21.

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id dst-ip=pnf-ip set dst-mac=pnf-mac, reg6=provider-lport-tag` =>
Egress table (220) output to provider port

Ingress traffic to VM in private network from a PNF in another private network

- New flow in table 19, to distinguish our new use-case, in which we want to decrease the TTL of the packet

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>

GW Mac table (19) `match: lport-tag=provider-port, vpn-id=router-id, dst-mac=router-interface-mac, set split-horizon-bit = 0, decrease-ttl` =>
FIB table (21) `match: vpn-id=router-id dst-ip=vm-ip set dst-mac=vm-mac reg6=provider-lport-tag` =>
Egress table (220) output to provider port

Yang changes

In odl-l3vpn module, adjacency-list grouping will be enhanced with the following field

```
 grouping adjacency-list {
  list adjacency {
    key "ip_address";
    ...
    leaf phys-network-func {
         type boolean;
         default false;
         description "Value of True indicates this is an adjacency of a device in a␣
↪provider network";
    }
  }
}
```

An adjacency that is added as a result of a PNF discovery, is a primary adjacency with an empty next-hop-ip list. This is not enough to distinguish PNF at all times. This new field will help us identify this use-case in a more robust way.

Configuration impact

A configuration mode will be available to turn this feature ON/OFF.

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

All traffic of PNFs in each subnet-prefix sends their traffic to a designated switch.

Targeted Release

Carbon

Alternatives

None

Usage

Create external network with a subnet

```
neutron net-create public-net -- --router:external --is-default --provider:network_
↪type=flat
--provider:physical_network=physnet1
neutron subnet-create --ip_version 4 --gateway 10.64.0.1 --name public-subnet1
↪<public-net-uuid> 10.64.0.0/16
-- --enable_dhcp=False
```

Create internal networks with subnets

```
neutron net-create private-net1
neutron subnet-create --ip_version 4 --gateway 10.0.123.1 --name private-subnet1
↪<private-net1-uuid>
10.0.123.0/24
neutron net-create private-net2
neutron subnet-create --ip_version 4 --gateway 10.0.124.1 --name private-subnet2
↪<private-net2-uuid>
10.0.124.0/24
```

Create a router instance and connect it to an internal subnet and an external subnet

This will allow communication with PNFs in provider network

```
neutron router-create router1
neutron router-interface-add <router1-uuid> <private-subnet1-uuid>
neutron router-gateway-set --fixed-ip subnet_id=<public-subnet1-uuid> <router1-uuid>
↪<public-net-uuid>
```

Create a router instance and connect to it to two internal subnets

This will allow East/West communication between VMs and PNFs

```
neutron router-create router1
neutron router-interface-add <router1-uuid> <private-subnet1-uuid>
neutron router-interface-add <router1-uuid> <private-subnet2-uuid>
```

Features to Install

odl-netvirt-openstack

REST API

CLI

Implementation

Assignee(s)

Primary assignee: Tomer Pearl <tomer.pearl@hpe.com>

Other contributors: Yakir Dorani <yakir.dorani@hpe.com>

Work Items

- Configure subnet-route flows upon ext-net configuration / router association
- Solve traffic lost issues of punted packets from table 22
- Enable aliveness monitoring on external interfaces.
- Add ARP responder flow for L3-PNF
- Add ARP packet-in from primary switch only
- Disable split-horizon and enable TTL decrease for L3-PNF

Dependencies

This feature depends on hairpinning feature [2]

Testing

Unit Tests

Unit tests will be added for the new functionality

Integration Tests

CSIT

Will need to see if a PNF could be simulated in CSIT

Documentation Impact

References

[1] https://docs.google.com/presentation/d/1ByvEQXUtIyH-H7Bin6OBJNrHjOv-3hpHYzU6Sf6hDbA/edit#slide=id.g11657174d1_0_31 [2] http://docs.opendaylight.org/en/latest/submodules/netvirt/docs/specs/hairpinning-flat-vlan.html

Table of Contents

ECMP Support for BGP based L3VPN

https://git.opendaylight.org/gerrit/#/q/topic:l3vpn_ecmp

This Feature is needed for load balancing of traffic in a cloud and also redundancy of paths for resiliency in cloud.

Problem description

The current L3VPN implementation for BGP VPN doesn't support load balancing behavior for `external routes` through multiple DC-GWs and reaching starting route behind Nova VMs through multiple compute nodes.

This spec provides implementation details about providing traffic load balancing using ECMP for L3 routing and forwarding. The load balancing of traffic can be across virtual machines with each connected to the different compute nodes, DC-Gateways. ECMP also enables fast failover of traffic The ECMP forwarding is required for both inter-DC and intra-DC data traffic types. For inter-DC traffic, spraying from DC-GW to compute nodes & VMs for the traffic entering DC and spraying from compute node to DC-GWs for the traffic exiting DC is needed. For intra-DC traffic, spraying of traffic within DC across multiple compute nodes & VMs is needed. There should be tunnel monitoring (e.g. GRE-KA or BFD) logic implemented to monitor DC-GW /compute node GRE tunnels which helps to determine available ECMP paths to forward the traffic.

Use Cases

- ECMP forwarding of traffic entering a DC (i.e. Spraying of DC-GW -> OVS traffic across multiple Compute Nodes & VMs). In this case, DC-GW can load balance the traffic if a `static route` can be reachable through multiple NOVA VMs (say VM1 and VM2 connected on different compute nodes) running some networking application (example: vRouter).
- ECMP forwarding of traffic exiting a DC (i.e. Spraying of OVS -> DC-GW traffic across multiple DC Gateways). In this case, a Compute Node can LB the traffic if `external route` can be reachable from multiple DC-GWs.

- ECMP forwarding of intra-DC traffic (i.e. Spraying of traffic within DC across multiple Compute Nodes & VMs) This is similar to UC1, but load balancing behavior is applied on remote Compute Node for intra-DC communication.
- OVS -> DC-GW tunnel status based ECMP for inter and intra-DC traffic. Tunnel status based on monitoring (BFD) is considered in ECMP path set determination.

High-Level Components:

The following components of the Openstack - ODL solution need to be enhanced to provide ECMP support:

- OpenStack Neutron BGPVPN Driver (for supporting multiple RDs)
- OpenDaylight Controller (NetVirt VpnService)

We will review enhancements that will be made to each of the above components in following sections.

Proposed change

The following components within OpenDaylight Controller needs to be enhanced:

- NeutronvpnManager
- VPN Engine (VPN Manager and VPN Interface Manager)
- FIB Manager

Pipeline changes

Local FIB entry/Nexthop Group programming:

A `static route` (example: 100.0.0.0/24) reachable through two VMs connected with same compute node.

Table 0=>Table 17=>Table 19=>Table 21=>LB Group=>Local VM Group=>Table 220

```
cookie=0x8000003, duration=46.020s, table=21, n_packets=0, n_bytes=0, priority=34,ip,
↪metadata=0x222e4/0xfffffffe, nw_dst=100.0.0.0/24 actions=write_actions(group:150002)
group_id=150002,type=select,bucket=weight:50,actions=group:150001,bucket=weight:50,
↪actions=group:150000
group_id=150001,type=all,bucket=actions=set_field:fa:16:3e:34:ff:58->eth_dst,
↪load:0x200->NXM_NX_REG6[],resubmit(,220)
group_id=150000,type=all,bucket=actions=set_field:fa:16:3e:eb:61:39->eth_dst,
↪load:0x100->NXM_NX_REG6[],resubmit(,220)
```

Remote FIB entry/Nexthop Group programming:

- A `static route` (example: 10.0.0.1/32) reachable through two VMs connected with different compute node.

  on remote compute node,

  Table 0=>Table 17=>Table 19=>Table 21=>LB Group=>VxLAN port

```
cookie=0x8000003, duration=46.020s, table=21, n_packets=0, n_bytes=0, priority=34,
↪ip,metadata=0x222e4/0xfffffffe, nw_dst=10.0.0.1 actions=set_field:0xEF->tun_id,␣
↪group:150003
group_id=150003,type=select,bucket=weight:50,actions=output:1,bucket=weight:50,
↪actions=output:2
```

on local compute node,

Here, From LB group, packets flow through local VM and VxLAN port

Table 0=>Table 17=>Table 19=>Table 21=>LB Group=>Local VM Group=>Table 220
.....=> VxLAN port

```
cookie=0x8000003, duration=46.020s, table=21, n_packets=0, n_bytes=0,␣
↪priority=34,ip,metadata=0x222e4/0xfffffffe, nw_dst=10.0.0.1␣
↪actions=group:150003
group_id=150003,type=select,bucket=weight:50,group=150001,bucket=weight:50,
↪actions=set_field:0xEF->tun_id, output:2
group_id=150001,type=all,bucket=actions=set_field:fa:16:3e:34:ff:58->eth_dst,
↪load:0x200->NXM_NX_REG6[],resubmit(,220)
```

- An `external route` (example: 20.0.0.1/32) reachable through two DC-GWs.

  Table 0=>Table 17=>Table 19=>Table 21=>LB Group=>GRE port

```
cookie=0x8000003, duration=13.044s, table=21, n_packets=0, n_bytes=0,priority=42,
↪ip,metadata=0x222ec/0xfffffffe,nw_dst=20.0.0.1 actions=load:0x64->NXM_NX_REG0[0.
↪.19],load:0xc8->NXM_NX_REG1[0..19],group:150111
group_id=150111,type=select,bucket=weight:50,actions=push_mpls:0x8847, move:NXM_
↪NX_REG0[0..19]->OXM_OF_MPLS_LABEL[],output:3, bucket=weight:50,actions=push_
↪mpls:0x8847,move:NXM_NX_REG1[0..19]->OXM_OF_MPLS_LABEL[],output:4
```

YANG changes

Changes will be needed in `l3vpn.yang` , `odl-l3vpn.yang` and `odl-fib.yang` to support ECMP functionality.

L3VPN YANG changes

route-distinguisher type is changed from leaf to leaf-list in vpn-af-config grouping in l3vpn.yang.

Listing 1.5: l3vpn.yang

```
 grouping vpn-af-config {
     description "A set of configuration parameters that is applicable to both IPv4␣
↪and
     IPv6 address family for a VPN instance .";

     leaf-list route-distinguisher {
         description "The route-distinguisher command configures a route␣
↪distinguisher (RD)
         for the IPv4 or IPv6 address family of a VPN instance.
         Format is ASN:nn or IP-address:nn.";
```

```
        config "true";
        type string{
            length "3..21";
        }
    }
}
```

ODL-L3VPN YANG changes

- Add `vrf-id` (RD) in adjacency list in odl-l3vpn.yang.

Listing 1.6: odl-l3vpn.yang

```
grouping adjacency-list {
    list adjacency{
        key "ip_address";
        leaf-list next-hop-ip-list { type string; }
        leaf ip_address {type string;}
        leaf primary-adjacency {
            type boolean;
            default false;
            description "Value of True indicates this is a primary adjacency";
        }

        leaf label { type uint32; config "false"; }     /*optional*/
        leaf mac_address {type string;}     /*optional*/
        leaf vrf-id {type string;}
        }
}
```

- vpn-to-extraroute have to be updated with multiple RDs `(vrf-id)` when extra route from VMs connected with different compute node and when connected on same compute node, just use same RD and update nexthop-ip-list with new VM IP address like below.

Listing 1.7: odl-l3vpn.yang

```
    container vpn-to-extraroutes {
        config false;
        list vpn-extraroutes {
            key "vpn-name";
            leaf vpn-name {
                type uint32;
            }

            list extra-routes {
                key "vrf-id";
                leaf vrf-id {
                    description "The vrf-id command configures a route␣
↪distinguisher (RD) for the IPv4
                    or IPv6 address family of a VPN instance or vpn instance name␣
↪for
                    internal vpn case.";
                    type string;
                }

               list route-paths {
```

```
                key "prefix";
                leaf prefix {type string;}
                leaf-list nexthop-ip-list {
                    type string;
                }
            }
        }
    }
}
```

- To manage RDs for extra with multiple next hops, the following YANG model is required to advertise (or) withdraw the extra routes with unique NLRI accordingly.

Listing 1.8: odl-l3vpn.yang

```
    container extraroute-routedistinguishers-map {
        config true;
        list extraroute-routedistingueshers {
            key "vpnid";
            leaf vpnid {
                type uint32;
            }

            list dest-prefixes {
                key "dest-prefix";
                leaf dest-prefix {
                    type string;
                    mandatory true;
                }

                leaf-list route-distinguishers {
                    type string;
                }
            }
        }
    }
```

ODL-FIB YANG changes

- When Quagga BGP announces route with multiple paths, then it is ODL responsibility to program Fib entries in all compute nodes where VPN instance blueprint is present, so that traffic can be load balanced between these two DC gateways. It requires changes in existing odl-fib.yang model (like below) to support multiple routes for same destination IP prefix.

Listing 1.9: odl-fib.yang

```
grouping vrfEntries {
    list vrfEntry {
        key  "destPrefix";
        leaf destPrefix {
            type string;
            mandatory true;
        }

        leaf origin {
            type string;
```

```
            mandatory true;
        }

            list route-paths {
                key "nexthop-address";
                leaf nexthop-address {
                    type string;
                    mandatory true;
                }

                leaf label {
                    type uint32;
                }
            }
    }
}
```

- New YANG model to update load balancing next hop group buckets according to VxLAN/GRE tunnel status [Note that these changes are required only if watch_port in group bucket is not working based on tunnel port liveness monitoring affected by the BFD status]. When one of the VxLAN/GRE tunnel is going down, then retrieve nexthop-key from dpid-l3vpn-lb-nexthops by providing tep-device-ids from `src-info` and `dst-info` of StateTunnelList while handling its update DCN. After retrieving next hop key, fetch target-device-id list from l3vpn-lb-nexthops and reprogram VxLAN/GRE load balancing group in each remote Compute Node based on tunnel state between source and destination Compute Node. Similarly, when tunnel comes up, then logic have to be rerun to add its bucket back into Load balancing group.

Listing 1.10: odl-fib.yang

```
    container l3vpn-lb-nexthops {
        config false;
        list nexthops {
            key "nexthop-key";
            leaf group-id { type string; }
            leaf nexhop-key { type string; }
            leaf-list target-device-id { type string;
            //dpId or ip-address }
         }
    }

    container dpid-l3vpn-lb-nexthops {
        config false;
        list dpn-lb-nexthops {
            key "src-dp-id dst-device-id";
            leaf src-dp-id { type uint64; }
            leaf dst-device-id { type string;
            //dpId or ip-address }
            leaf-list nexthop-keys { type string; }
        }
    }
```

ECMP forwarding through multiple Compute Node and VMs

In some cases, extra route can be added which can have reachability through multiple Nova VMs. These VMs can be either connected on same compute node (or) different Compute Nodes. When VMs are in different compute nodes, DC-GW should learn all the route paths such that ECMP behavior can be applied for these multi path routes. When

VMs are co-located in same compute node, DC-GW will not perform ECMP and compute node performs traffic splitting instead.

ECMP forwarding for dispersed VMs

When configured extra route are reached through nova VMs which are connected with different compute node, then it is ODL responsibility to advertise these multiple route paths (but with same `MPLS label`) to Quagga BGP which in turn sends these routes into DC-GW. But DC-GW replaces the existing route with a new route received from the peer if the NLRI (prefix) is same in the two routes.

This is true even when multipath is enabled on the DC-GW and it is as per standard BGP RFC 4271, Section 9 UPDATE Message Handling. Hence the route is lost in DC-GW even before path computation for multipath is applied.This scenario is solved by adding multiple route distinguisher (RDs) for the vpn instance and let ODL uses the list of RDs to advertise the same prefix with different BGP NHs. Multiple RDs will be supported only for BGP VPNs.

ECMP forwarding for co-located VMs

When extra routes on VM interfaces are connected with same compute node, LFIB/FIB and Terminating service table flow entries should be programmed so that traffic can be load balanced between local VMs. This can be done by creating load balancing next hop group for each vpn-to-extraroute (if nexthop-ip-list size is greater than 1) with buckets pointing to the actual VMs next hop group on source Compute Node. Even for the co-located VMs, VPN interface manager should assign separate RDs for each adjacency of same dest IP prefix and let route can be advertised again to Quagga BGP with same next hop (TEP IP address). This will enable DC-Gateway to realize ECMP behavior when an IP prefix can be reachable through multiple co located VMs on one Compute Node and an another VM connected on different Compute Node.

To create load balancing next hop group, the dest IP prefix is used as the key to generate group id. When any of next hop is removed, then adjust load balancing nexthop group so that traffic can be sent through active next hops.

ECMP forwarding through two DC-Gateways

The current ITM implementation provides support for creating multiple GRE tunnels for the provided list of DC-GW IP addresses from compute node. This should help in creating corresponding load balancing group whenever Quagga BGP is advertising two routes on same IP prefix pointing to multiple DC GWs. The group id of this load balancing group can be derived from sorted order of `DC GW TEP IP addresses` with the following format dc_gw_tep_ip _address_1: dc_gw_tep_ip_address_2. This will be useful when multiple external IP prefixes share the same next hops. The load balancing next hop group buckets is programmed according to sorted remote end point DC-Gateway IP address. The support of action move:NXM_NX_REG0(1) -> `MPLS label` is not supported in ODL openflowplugin. It has to be implemented. Since there are two DC gateways present for the data center, it is possible that multiple equal cost routes are supplied to ODL by Quagga BGP like Fig 2. The current Quagga BGP doesn't have multipath support and it will be done. When Quagga BGP announces route with multiple paths, then it is ODL responsibility to program Fib entries in all compute nodes where VPN instance blueprint is present, so that traffic can be load balanced between these two DC gateways. It requires changes in existing odl-fib.yang model (like below) to support multiple routes for same destination IP prefix.

BGPManager should be able to create vrf entry for the advertised IP prefix with multiple route paths. VrfEntryListener listens to DCN on these vrf entries and program Fib entries (21) based on number route paths available for given IP prefix. For the given (external) destination IP prefix, if there is only one route path exists, use the existing approach to program FIB table flow entry matches on (vpnid, ipv4_dst) and actions with push `MPLS label` and output to gre tunnel port. For the given (external) destination IP prefix, if there are two route paths exist, then retrieve next hop ip address from routes list in the same sorted order (i.e. using same logic which is used to create buckets for load balancing next hop group for DC- Gateway IP addresses), then program FIB table flow entry with an instruction like

Fig 3. It should have two set field actions where first action sets `MPLS label` to NX_REG0 for first sorted DC-GW IP address and second action sets `MPLS label` to NX_REG1 for the second sorted DC-GW IP address. When more than two DC Gateways are used, then more number of NXM Registries have to be used to push appropriate `MPLS label` before sending it to next hop group. It needs operational DS container to have mapping between DC Gateway IP address and NXM_REG. When one of the route is withdrawn for the IP prefix, then modify the FIB table flow entry with with push `MPLS label` and output to the available gre tunnel port.

ECMP for Intra-DC L3VPN communication

ECMP within data center is required to load balance the data traffic when extra route can be reached through multiple next hops (i.e. Nova VMs) when these are connected with different compute nodes. It mainly deals with how Compute Nodes can spray the traffic when dest IP prefix can be reached through two or more VMs (next hops) which are connected with multiple compute nodes.

When there are multiple RDs (if VPN is of type BGP VPN) assigned to VPN instance so that VPN engine can be advertise IP route with different RDs to achieve ECMP behavior in DC-GW as mentioned before. But for intra-DC, this doesn't make any more sense since it's all about programming remote FIB entries on computes nodes to achieve data traffic spray behavior.

Irrespective of RDs, when multiple next hops (which are from different Compute Nodes) are present for the extra-route adjacency, then FIB Manager has to create load balancing next hop group in remote compute node with buckets pointing with targeted Compute Node VxLAN tunnel ports.

To allocate group id for this load balancing next hop, the same destination IP prefix is used as the group key. The remote FIB table flow should point to this next hop group after writing prefix `label` into tunnel_id. The bucket weight of remote next hop is adjusted according to number of VMs associated to given extra route and on which compute node the VMs are connected. For example, two compute node having one VM each, then bucket weight is 50 each. One compute node having two VMs and another compute node having one VM, then bucket weight is 66 and 34 each. The hop-count property in vrfEntry data store helps to decide what is the bucket weight for each bucket.

ECMP Path decision based on Internal/External Tunnel Monitoring

ODL will use GRE-KA or BFD protocol to implement monitoring of GRE external tunnels. This implementation detail is out of scope in this document. Based on the tunnel state, GRE Load Balancing Group is adjusted accordingly as mentioned like below.

GRE tunnel state handling

As soon as GRE tunnel interface is created in ODL, interface manager uses alivenessmonitor to monitor the GRE tunnels for its liveness using GRE Keep-alive protocol. When tunnel state changes, it has to handled accordingly to adjust above load balancing group so that data traffic is sent to only active DC-GW tunnel. This can be done with listening to update StateTunnelList DCN.

When one GRE tunnel is operationally going down, then retrieve the corresponding bucket from the load balancing group and delete it. When GRE tunnel comes up again, then add bucket back into load balancing group and reprogram it.

When both GRE tunnels are going down, then just recreate load balancing group with empty. Withdraw the routes from that particular DC-GW. With the above implementation, there is no need of modifying Fib entries for GRE tunnel state changes.

But when BGP Quagga withdrawing one of the route for external IP prefix, then reprogram FIB flow entry (21) by directly pointing to output=<gre_port> after pushing `MPLS label`.

VxLAN tunnel state handling

Similarly, when VxLAN tunnel state changes, the Load Balancing Groups in Compute Nodes have to be updated accordingly so that traffic can flow through active VxLAN tunnels. It can be done by having config mapping between target `data-path-id` to `next hop group Ids` and vice versa.

For both GRE and VxLAN tunnel monitoring, L3VPN has to implement the following YANG model to update load balancing next hop group buckets according to tunnel status.

When one of the VxLAN/GRE tunnel is going down, then retrieve nexthop-key from dpid-l3vpn-lb-nexthops by providing tep-device-ids from `src-info` and `dst-info` of StateTunnelList while handling its update DCN.

After retrieving next hop key, fetch target-device-id list from l3vpn-lb-nexthops and reprogram VxLAN/GRE load balancing group in each remote Compute Node based on tunnel state between source and destination Compute Node. Similarly, when tunnel comes up, then logic have to be rerun to add its bucket back into Load balancing group.

Assumptions

The support for action move:NXM_NX_REG0(1) -> `MPLS label` is already available in Compute Node.

Reboot Scenarios

This feature support all the following Reboot Scenarios for EVPN:

- Entire Cluster Reboot
- Leader PL reboot
- Candidate PL reboot
- OVS Datapath reboots
- Multiple PL reboots
- Multiple Cluster reboots
- Multiple reboots of the same OVS Datapath.
- Openstack Controller reboots

Clustering considerations

The feature should operate in ODL Clustered environment reliably.

Other Infra considerations

N.A.

Security considerations

N.A.

Scale and Performance Impact

Not covered by this Design Document.

Targeted Release

Carbon.

Alternatives

Alternatives considered and why they were not selected.

Usage

Features to Install

This feature doesn't add any new karaf feature.

REST API

Implementation

Assignee(s)

Primary assignee(s):

- Manu B <manu.b@ericsson.com>
- Kency Kurian <kency.kurian@ericsson.com>
- Gobinath <gobinath@ericsson.com>
- P Govinda Rajulu <p.govinda.rajulu@ericsson.com>

Other contributors:

- Periyasamy Palanisamy <periyasamy.palanisamy@ericsson.com>

Work Items

The Trello cards have already been raised for this feature under l3vpn_ecmp.

Link for the Trello Card: https://trello.com/c/8E3LWIkq/121-ecmp-support-for-bgp-based-l3vpn-l3vpn-ecmp

Dependencies

Quagga BGP multipath support and APIs. This is needed to support when two DC-GW advertises routes for same external prefix with different route `labels` GRE tunnel monitoring. This is need to implement ECMP forwarding based on MPLSoGRE tunnel state. Support for action move:NXM_NX_REG0(1) -> `MPLS label` in ODL openflowplugin

Testing

Capture details of testing that will need to be added.

Unit Tests

Appropriate UTs will be added for the new code coming in once framework is in place.

Integration Tests

There won't be any Integration tests provided for this feature.

CSIT

CSIT will be enhanced to cover this feature by providing new CSIT tests.

Documentation Impact

This will require changes to User Guide and Developer Guide.

References

- https://docs.google.com/document/d/1KRxrIGCLCBuz2D8f8IhU2I84VrM5EMa1Y7Scjb6qEKw

Table of Contents

Element Counters

https://git.opendaylight.org/gerrit/#/q/element-counters

This feature depends on the Netvirt statistics feature.

This feature enables collecting statistics on filtered traffic passed from/to a network element. For example: traffic outgoing/incoming from a specific IP, tcp traffic, udp traffic, incoming/outgoing traffic only.

Problem description

Collecting statistics on filtered traffic sent to/from a VM is currently not possible.

Use Cases

- Tracking East/West communication between local VMs.
- Tracking East/West communication between VMs that are located in different compute nodes.
- Tracking communication between a local VM and an IP located in an external network.
- Tracking TCP/UDP traffic sent from/to a VM.
- Tracking dropped packets between 2 VMs.

Proposed change

The Netvirt Statistics Plugin will receive requests regarding element filtered counters. A new service will be implemented ("CounterService"), and will be associated with the relevant interfaces (either ingress side, egress sides or both of them).

- Ingress traffic: The service will be the first one in the pipeline after the Ingress ACL service.

- Egress traffic: The service will be the last one after the Egress ACL service.
- The input for counters request regarding VM A, and incoming and outgoing traffic from VM B, will be VM A interface uuid and VM B IP.
- The input can also include other filters like TCP only traffic, UDP only traffic, incoming/outgoing traffic.
- In order to track dropped traffic between VM A and VM B, the feature should be activated on both VMS (either in the same compute node or in different compute nodes). service binding will be done on both VMs relevant interfaces.
- If the counters request involves an external IP, service binding will be done only on the VM interface.
- Adding/Removing the "CounterService" should be dynamic and triggered by requesting element counters.

The Statistics Plugin will use OpenFlow flow statistic requests for these new rules, allowing it to gather statistics regarding the traffic between the 2 elements. It will be responsible to validate and filter the counters results.

Pipeline changes

Two new tables will be used: table 219 for outgoing traffic from the VM, and table 249 for incoming traffic from the VM. In both ingress and egress pipelines, the counter service will be just after the appropriate ACL service. The default rule will resubmit traffic to the appropriate dispatcher table.

Assuming we want statistics on VM A traffic, received or sent from VM B.

VM A Outgoing Traffic (vm interface)

In table 219 traffic will be matched against dst-ip and lport tag.

> Ingress dispatcher table (17): `match: lport-tag=vmA-interface, actions: go to table 219` =>
> Ingress counters table (219): `match: dst-ip=vmB-ip, lport-tag=vmA-interface, actions: resubmit to table 17` =>

VM A Incoming Traffic (vm interface)

In table 249 traffic will be matched against src-ip and lport tag.

> Egress dispatcher table (220): `match: lport-tag=vmA-interface, actions: go to table 249` =>
> Egress counters table (249): `match: lport-tag=vmA-interface, src-ip=vmB-ip, actions: resubmit to table 220` =>

Assuming we want statistics on VM A incoming TCP traffic.

VM A Outgoing Traffic (vm interface)

> Egress dispatcher table (220): `match: lport-tag=vmA-interface, actions: go to table 249` =>
> Egress counters table (249): `match: lport-tag=vmA-interface, tcp, actions: resubmit to table 220` =>

Assuming we want statistics on VM A outgoing UDP traffic.

VM A Incoming traffic (vm interface)

Ingress dispatcher table (17): `match: lport-tag=vmA-interface, actions: go to table 219` =>
Ingress counters table (219): `match: lport-tag=vmA-interface, udp, actions: resubmit to table 17` =>

Assuming we want statistics on all traffic sent to VM A port.

VM A Incoming traffic (vm interface)

Ingress dispatcher table (17): `match: lport-tag=vmA-interface, actions: go to table 219` =>
Ingress counters table (219): `match: lport-tag=vmA-interface, actions: resubmit to table 17` =>

Yang changes

Netvirt Statistics module will be enhanced with the following RPC:

```
grouping result {
    list counterResult {
        key id;
        leaf id {
            type string;
        }
        list groups {
            key name;
            leaf name {
                type string;
            }
            list counters {
                key name;
                leaf name {
                    type string;
                }
                leaf value {
                    type uint64;
                }
            }
        }
    }
}

grouping filters {
    leaf-list groupFilters {
        type string;
    }
    leaf-list counterFilter {
        type string;
    }
}

grouping elementRequestData {
    container filters {
```

```
            container tcpFilter {
            leaf on {
                type boolean;
            }
            leaf srcPort {
                type int32;
                default -1;
            }
            leaf dstPort {
                type int32;
                default -1;
            }
        }

        container udpFilter {
            leaf on {
                type boolean;
            }
            leaf dstPort {
                type int32;
                default -1;
            }
            leaf srcPort {
                type int32;
                default -1;
            }
        }

        container ipFilter {
            leaf ip {
                type string;
                default "";
            }
        }
    }
}

container elementCountersRequestConfig {
    list counterRequests {
        key "requestId";
        leaf requestId {
            type string;
        }
        leaf lportTag {
            type int32;
        }
        leaf dpn {
            type uint64;
        }
        leaf portId {
            type string;
        }
        leaf trafficDirection {
            type string;
        }
        uses elementRequestData;
    }
}
```

```
rpc acquireElementCountersRequestHandler {
    input {
        leaf portId {
            type string;
        }
        container incomingTraffic {
            uses elementRequestData;
        }
        container outgoingTraffic {
            uses elementRequestData;
        }
        uses filters;
    }
    output {
        leaf incomingTrafficHandler {
            type string;
        }
        leaf outcoingTrafficHandler {
            type string;
        }
    }
}

rpc releaseElementCountersRequestHandler {
    input {
        leaf handler {
            type string;
        }
    }
    output {
    }
}

rpc getElementCountersByHandler {
    input {
        leaf handler {
            type string;
        }
    }
    output {
        uses result;
    }
}
```

Configuration impact

The described above YANG model will be saved in the data store.

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

Since adding the new service is done by a request (as well as removing it), not all packets will be sent to the new tables described above.

Targeted Release

Carbon

Alternatives

None

Usage

- Create router, network, 2 VMS, VXLAN tunnel.
- Connect to each one of the VMs and send ping to the other VM.
- Use REST to get the statistics.

Run the following to get interface ids:

```
http://10.0.77.135:8181/restconf/operational/ietf-interfaces:interfaces-state/
```

Choose VM B interface and use the following REST in order to get the statistics: Assuming VM A IP = 1.1.1.1, VM B IP = 2.2.2.2

Acquire counter request handler:

```
10.0.77.135:8181/restconf/operations/statistics-
↪plugin:acquireElementCountersRequestHandler, {"input":{"portId":"4073b4fe-a3d5-47c0-
↪b37d-4fb9db4be9b1", "incomingTraffic":{"filters":{"ipFilter":{"ip":"1.1.3.9"}}}}},␣
↪headers={Authorization=Basic YWRtaW46YWRtaW4=, Cache-Control=no-cache, Content-
↪Type=application/json}]
```

Release handler:

```
10.0.77.135:8181/restconf/operations/statistics-
↪plugin:releaseElementCountersRequestHandler, input={"input":{"handler":"1"}},␣
↪headers={Authorization=Basic YWRtaW46YWRtaW4=, Cache-Control=no-cache, Content-
↪Type=application/json}]
```

Get counters:

```
10.0.77.135:8181/restconf/operations/statistics-plugin:getElementCountersByHandler,␣
↪input={"input":{"handler":"1"}}, headers={Authorization=Basic YWRtaW46YWRtaW4=,␣
↪Cache-Control=no-cache, Content-Type=application/json}]
```

Example counters output:


```
  {
"output": {
  "counterResult": [
    {
      "id": "SOME UNIQUE ID",
      "groups": [
        {
          "name": "Duration",
          "counters": [
            {
              "name": "durationNanoSecondCount",
              "value": 298000000
            },
            {
              "name": "durationSecondCount",
              "value": 10369
            }
          ]
        },
        {
          "name": "Bytes",
          "counters": [
            {
              "name": "bytesTransmittedCount",
              "value": 648
            },
            {
              "name": "bytesReceivedCount",
              "value": 0
            }
          ]
        },
        {
          "name": "Packets",
          "counters": [
            {
              "name": "packetsTransmittedCount",
              "value": 8
            },
            {
              "name": "packetsReceivedCount",
              "value": 0
            }
          ]
        }
      ]
    }
  ]
}
```

Features to Install

odl-netvirt-openstack

REST API

CLI

Implementation

Assignee(s)

Primary assignee: Guy Regev <guy.regev@hpe.com>

Other contributors: TBD

Work Items

https://trello.com/c/88MnwGwb/129-element-to-element-counters

- Add new service in Genius.
- Implement new rules installation.
- Update Netvirt Statistics module to support the new counters request.

Dependencies

None

Testing

Unit Tests

Integration Tests

CSIT

Documentation Impact

References

Netvirt statistics feature: https://git.opendaylight.org/gerrit/#/c/50164/8

Table of Contents

Hairpinning of floating IPs in flat/VLAN provider networks

https://git.opendaylight.org/gerrit/#/q/topic:hairpinning

This feature enables VM instances connected to the same router to communicate with each other using their floating ip addresses directly without traversing via the external gateway.

Problem description

Local and East/West communication between VMs using floating ips for flat/VLAN provider types is not handled internally by the pipeline currently. As a result, this type of traffic is mistakenly classified as North/South and routed to the external network gateway.

Today, SNATted traffic to flat/VLAN network is routed directly to the external gateway after traversing the SNAT/outbound NAPT pipeline using OF group per external network subnet. The group itself sets the destination mac as the mac address of the external gw associated with the floating ip/ router gw and output to the provider network port via the egress table. This workflow would be changed to align with the VxLAN provider type and direct SNATted traffic back to the FIB where the destination can then resolved to be floating ip on local or remote compute node.

Use Cases

- Local and East/West communication between VMs co-located on the same compute node using associated floating ip.
- Local and East/West communication between VMs located on different compute nodes using associated floating ip.

Proposed change

- The vpn-id used for classification of floating ips and router gateway external addresses in flat/VLAN provider networks is based on the external network id. It will be changed to reflect the subnet id associated with the floating ip/router gateway. This will allow traffic from the SNAT/outbound NAPT table to be resubmitted back to the FIB while preserving the subnet id.
- Each floating ip already has VRF entry in the fib table. The vpn-id of this entry will also be based on the subnet id of the floating ip instead of the external network id. If the VM associated with the floating ip is located on remote compute node, the traffic will be routed to the remote compute based on the provider network of the subnet from which the floating ip was allocated e.g. if the private network is VxLAN and the external network is VLAN provider, traffic to floating ip on remote compute node will be routed to the provider port associated with the VLAN provider and not the tunnel associated with the VxLAN provider.
- In the FIB table of the egress node, the destination mac will be replaced with the mac address of the floating ip in case of routing to remote compute node. This will allow traffic from flat/VLAN provider enter the L3 pipeline for DNAT of the floating ip.
- Default flow will be added to the FIB table for each external subnet-id. If no floating ip match was found in the FIB table for the subnet id, the traffic will be sent to the group of the external subnet. Each group entry will perform the following: (a) replace the destination mac address to the external gateway mac address (b) send the traffic to the provider network via the egress table.
- Ingress traffic from flat/VLAN provider network is bounded to L3VPN service using vpn-id of the external network id. To allow traffic classification based on subnet id for floating ips and router gateway ips, the GW MAC table will replace the vpn-id of the external network with the vpn-id of the subnet id of the floating ip. For

ingress traffic to router gateway mac, the vpn-id of the correct subnet will be deterined at the FIB table based on the router gateway fixed ip.

- A new model will be introduced to contain the new vpn/subnet associations - `odl-nat:subnets-networks`. This model will be filled only for external flat/VLAN provider networks and will take precedence over `odl-nat:external-networks` model for selection of vpn-id. BGPVPN use cases won't be affected by these changes as this model will not be applicable for these scenarios.

Pipeline changes

Egress traffic from VM with floating IP to the internet

- For Pre SNAT, SNAT, FIB tables the vpn-id will be based on the subnet-id of the floating ip
- Packets from SNAT table resubmitted back to the FIB rather than straight to the external network subnet-id group. In the FIB table it should be matched against a new flow with lower priority than any other flow containing dst-ip match. Traffic will be redirected based on the vpn-id of the floating ip subnet to the external network subnet-id group.

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id` =>
Pre SNAT table (26) `match: vpn-id=router-id,src-ip=vm-ip set vpn-id=fip-subnet-id,src-ip=fip` =>
SNAT table (28) `match: vpn-id=fip-subnet-id,src-ip=fip set src-mac=fip-mac` =>
FIB table (21) `match: vpn-id=fip-subnet-id` =>
Subnet-id group: `set dst-mac=ext-subnet-gw-mac, reg6=provider-lport-tag` =>
Egress table (220) output to provider network

Ingress traffic from the internet to VM with floating IP

- For GW MAC, FIB table the vpn-id will be based on the subnet-id of the floating ip

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=ext-net-id` =>
GW Mac table (19) `match: vpn-id=ext-net-id,dst-mac=floating-ip-mac set vpn-id=fip-subnet-id` =>
FIB table (21) `match: vpn-id=fip-subnet-id,dst-ip=fip` =>
Pre DNAT table (25) `match: dst-ip=fip set vpn-id=router-id,dst-ip=vm-ip` =>
DNAT table (27) `match: vpn-id=router-id,dst-ip=vm-ip` =>
FIB table (21) `match: vpn-id=router-id,dst-ip=vm-ip` =>
Local Next-Hop group: `set dst-mac=vm-mac, reg6=vm-lport-tag` =>
Egress table (220) output to VM port

Egress traffic from VM with no associated floating IP to the internet - NAPT switch

- For Outbound NAPT, NAPT PFIB and FIB tables the vpn-id will be based on the subnet-id of the router gateway
- Packets from NAPT PFIB table resubmitted back to the FIB rather than straight to the external network subnet-id group. In the FIB table it should be matched against a new flow with lower priority than any other flow containing dst-ip match. Traffic will be redirected based on the vpn-id of the router gateway subnet to the external network subnet-id group.

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id` =>
Pre SNAT table (26) `match: vpn-id=router-id` =>
Outbound NAPT table (46) `match: src-ip=vm-ip,port=int-port set src-ip=router-gw-ip,vpn-id=router-gw-subnet-id,port=ext-port` =>
NAPT PFIB table (47) `match: vpn-id=router-gw-subnet-id` =>
FIB table (21) `match: vpn-id=router-gw-subnet-id` =>
Subnet-id group: `set dst-mac=ext-subnet-gw-mac, reg6=provider-lport-tag` =>
Egress table (220) output to provider network

Ingress traffic from the internet to VM with no associated floating IP - NAPT switch

- For FIB table the vpn-id will be based on the subnet-id of the router gateway

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=ext-net-id` =>
GW Mac table (19) `match vpn-id=ext-net-id,dst-mac=router-gw mac` =>
FIB table (21) `match: vpn-id=ext-net-id,dst-ip=router-gw set vpn-id=router-gw-subnet-id` =>
Inbound NAPT table (44) `match: dst-ip=router-gw,port=ext-port set dst-ip=vm-ip,vpn-id=router-id,port=int-port` =>
PFIB table (47) `match: vpn-id=router-id` =>
FIB table (21) `match: vpn-id=router-id,dst-ip=vm-ip` =>
Local Next-Hop group: `set dst-mac=vm-mac,reg6=vm-lport-tag` =>
Egress table (220) output to VM port

Hairpinning - VM traffic to floating ip on the same compute node

- For Pre SNAT, SNAT, FIB tables the vpn-id will be based on the subnet-id of the floating ips

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id` =>

Pre SNAT table (26) `match: vpn-id=router-id,src-ip=src-vm-ip set vpn-id=fip-subnet-id,src-ip=src-fip` =>
SNAT table (28) `match: vpn-id=fip-subnet-id,src-ip=src-fip set src-mac=src-fip-mac` =>
FIB table (21) `match: vpn-id=fip-subnet-id,dst-ip=dst-fip` =>
Pre DNAT table (25) `match: dst-ip=dst-fip set vpn-id=router-id,dst-ip=dst-vm-ip` =>
DNAT table (27) `match: vpn-id=router-id,dst-ip=dst-vm-ip` =>
FIB table (21) `match: vpn-id=router-id,dst-ip=dst-vm-ip` =>
Local Next-Hop group: `set dst-mac=dst-vm-mac,reg6=dst-vm-lport-tag` =>
Egress table (220) output to VM port

Hairpinning - VM traffic to floating ip on remote compute node

VM originating the traffic (Ingress DPN):

- For Pre SNAT, SNAT, FIB tables the vpn-id will be based on the subnet-id of the floating ip
- The destination mac is updated by the FIB table to be the floating ip mac. Traffic is sent to the egress DPN over the port of the flat/VLAN provider network.

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id` =>
Pre SNAT table (26) `match: vpn-id=router-id,src-ip=src-vm-ip set vpn-id=fip-subnet-id,src-ip=src-fip` =>
SNAT table (28) `match: vpn-id=fip-subnet-id,src-ip=src-fip set src-mac=src-fip-mac` =>
FIB table (21) `match: vpn-id=fip-subnet-id,dst-ip=dst-fip set dst-mac=dst-fip-mac, reg6=provider-lport-tag` =>
Egress table (220) output to provider network

VM receiving the traffic (Egress DPN):

- For GW MAC, FIB table the vpn-id will be based on the subnet-id of the floating ip

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=ext-net-id` =>
GW Mac table (19) `match: vpn-id=ext-net-id,dst-mac=dst-fip-mac set vpn-id=fip-subnet-id` =>
FIB table (21) `match: vpn-id=fip-subnet-id,dst-ip=dst-fip` =>
Pre DNAT table (25) `match: dst-ip=dst-fip set vpn-id=router-id,dst-ip=dst-vm-ip` =>
DNAT table (27) `match: vpn-id=router-id,dst-ip=dst-vm-ip` =>
FIB table (21) `match: vpn-id=router-id,dst-ip=dst-vm-ip` =>

Local Next-Hop group: `set dst-mac=dst-vm-mac,lport-tag=dst-vm-lport-tag` =>
Egress table (220) output to VM port

Hairpinning - traffic from VM with no associated floating IP to floating ip on remote compute node

VM originating the traffic (Ingress DPN) is non-NAPT switch:

- No flow changes required. Traffic will be directed to NAPT switch and directed to the outbound NAPT table straight from the internal tunnel table

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id` =>
Pre SNAT table (26) `match: vpn-id=router-id` =>
NAPT Group `output to tunnel port of NAPT switch` =>

VM originating the traffic (Ingress DPN) is the NAPT switch:

- For Outbound NAPT, NAPT PFIB, Pre DNAT, DNAT and FIB tables the vpn-id will be based on the common subnet-id of the router gateway and the floating-ip.
- Packets from NAPT PFIB table resubmitted back to the FIB where they will be matched against the destnation floating ip.
- The destination mac is updated by the FIB table to be the floating ip mac. Traffic is sent to the egress DPN over the port of the flat/VLAN provider network.

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id` =>
Pre SNAT table (26) `match: vpn-id=router-id` =>
Outbound NAPT table (46) `match: src-ip=vm-ip,port=int-port set src-ip=router-gw-ip,vpn-id=router-gw-subnet-id,port=ext-port` =>
NAPT PFIB table (47) `match: vpn-id=router-gw-subnet-id` =>
FIB table (21) `match: vpn-id=router-gw-subnet-id dst-ip=dst-fip set dst-mac=dst-fip-mac, reg6=provider-lport-tag` =>
Egress table (220) output to provider network

VM receiving the traffic (Egress DPN):

- For GW MAC, FIB table the vpn-id will be based on the subnet-id of the floating ip

Classifier table (0) =>

Dispatcher table (17) `l3vpn service: set vpn-id=ext-net-id` =>
GW Mac table (19) `match: vpn-id=ext-net-id,dst-mac=dst-fip-mac set vpn-id=fip-subnet-id` =>
FIB table (21) `match: vpn-id=fip-subnet-id,dst-ip=dst-fip` =>
Pre DNAT table (25) `match: dst-ip=dst-fip set vpn-id=router-id,dst-ip=dst-vm-ip` =>
DNAT table (27) `match: vpn-id=router-id,dst-ip=dst-vm-ip` =>
FIB table (21) `match: vpn-id=router-id,dst-ip=dst-vm-ip` =>
Local Next-Hop group: `set dst-mac=dst-vm-mac,lport-tag=dst-vm-lport-tag` =>
Egress table (220) output to VM port

Yang changes

odl-nat module will be enhanced with the following container

```
container external-subnets {
  list subnets  {
    key id;
    leaf id {
       type yang:uuid;
    }
    leaf vpnid {
       type yang:uuid;
    }
    leaf-list router-ids {
       type yang:uuid;
    }
    leaf external-network-id {
       type yang:uuid;
    }
  }
}
```

This model will be filled out only for flat/VLAN external network provider types. If this model is missing, vpn-id will be taken from `odl-nat:external-networks` model to maintain compatibility with BGPVPN models.

`odl-nat:ext-routers` container will be enhanced with the list of the external subnet-ids associated with the router.

```
container ext-routers {
  list routers {
    key router-name;
    leaf router-name {
      type string;
    }
    ...

    leaf-list external-subnet-id {
      type yang:uuid; }
    }
  }
}
```

Configuration impact

None

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

None

Targeted Release

Carbon

Alternatives

None

Usage

Create external network with two subnets

```
neutron net-create public-net -- --router:external --is-default --provider:network_
↪type=flat
--provider:physical_network=physnet1
neutron subnet-create --ip_version 4 --gateway 10.64.0.1 --name public-subnet1
↪<public-net-uuid> 10.64.0.0/16
-- --enable_dhcp=False
neutron subnet-create --ip_version 4 --gateway 10.65.0.1 --name public-subnet2
↪<public-net-uuid> 10.65.0.0/16
-- --enable_dhcp=False
```

Create internal networks with subnets

```
neutron net-create private-net1
neutron subnet-create --ip_version 4 --gateway 10.0.123.1 --name private-subnet1
↪<private-net1-uuid>
10.0.123.0/24
neutron net-create private-net2
neutron subnet-create --ip_version 4 --gateway 10.0.124.1 --name private-subnet2
↪<private-net2-uuid>
10.0.124.0/24
neutron net-create private-net3
neutron subnet-create --ip_version 4 --gateway 10.0.125.1 --name private-subnet3
↪<private-net3-uuid>
10.0.125.0/24
neutron net-create private-net4
neutron subnet-create --ip_version 4 --gateway 10.0.126.1 --name private-subnet4
↪<private-net4-uuid>
10.0.126.0/24
```

Create two router instances and connect each router to one internal subnet and one external subnet

```
neutron router-create router1
neutron router-interface-add <router1-uuid> <private-subnet1-uuid>
neutron router-gateway-set --fixed-ip subnet_id=<public-subnet1-uuid> <router1-uuid>
↪<public-net-uuid>
neutron router-create router2
neutron router-interface-add <router2-uuid> <private-subnet2-uuid>
neutron router-gateway-set --fixed-ip subnet_id=<public-subnet2-uuid> <router2-uuid>
↪<public-net-uuid>
```

Create router instance connected to both external subnets and the remaining internal subnets

```
neutron router-create router3
neutron router-interface-add <router3-uuid> <private-subnet3-uuid>
neutron router-interface-add <router3-uuid> <private-subnet4-uuid>
neutron router-gateway-set --fixed-ip subnet_id=<public-subnet1-uuid> --fixed-ip␣
↪subnet_id=<public-subnet2-uuid>
<router3-uuid> <public-net-uuid>
```

Create floating ips from both subnets

```
neutron floatingip-create --subnet <public-subnet1-uuid> public-net
neutron floatingip-create --subnet <public-subnet1-uuid> public-net
neutron floatingip-create --subnet <public-subnet2-uuid> public-net
```

Create 2 VM instance in each subnet and associate with floating ips

```
nova boot --image <image-id> --flavor <flavor-id> --nic net-id=<private-net1-uuid> VM1
nova floating-ip-associate VM1 <fip1-public-subnet1>
```

```
nova boot --image <image-id> --flavor <flavor-id> --nic net-id=<private-net1-uuid> VM2
nova floating-ip-associate VM2 <fip2-public-subnet1>
nova boot --image <image-id> --flavor <flavor-id> --nic net-id=<private-net2-uuid> VM3
nova floating-ip-associate VM3 <fip1-public-subnet2>
nova boot --image <image-id> --flavor <flavor-id> --nic net-id=<private-net2-uuid> VM4
nova boot --image <image-id> --flavor <flavor-id> --nic net-id=<private-net3-uuid> VM5
nova boot --image <image-id> --flavor <flavor-id> --nic net-id=<private-net4-uuid> VM6
```

Connectivity tests

- Connect to the internet from all VMs. `VM1` and `VM2` will route traffic through external gateway 10.64.0.1 `VM3` and `VM4` route traffic through external gateway 10.65.0.1.
- Connect to the internet from `VM5` and `VM6`. Each connection will be routed to different external gateway with the corresponding subnet router-gateway ip.
- Hairpinning when source VM is associated with floating ip - ping between `VM1` and `VM2` using their floating ips.
- Hairpinning when source VM is not associated with floating ip - ping from `VM4` to `VM3` using floating ip. Since `VM4` has no associated floating ip a NAPT entry will be allocated using the router-gateway ip.

Features to Install

odl-netvirt-openstack

REST API

N/A

CLI

N/A

Implementation

Assignee(s)

Primary assignee: Yair Zinger <yair.zinger@hpe.com>

Other contributors: Tali Ben-Meir <tali@hpe.com>

Work Items

https://trello.com/c/uDcQw95v/104-pipeline-changes-fip-w-multiple-subnets-in-ext-net-hairpinning

- Add external-subnets model
- Add vpn-instances for external flat/VLAN sunbets
- Change pipeline to prefer vpn-id from external-subnets over vpn-id from external-networks

- Add write metadata to GW MAC table for floating ip/router gw mac addresses
- Add default subnet-id match in FIB table to external subnet group entry
- **Changes in remote next-hop flow for floating ip in FIB table**
  - Set destination mac to floating ip mac
  - Set egress actions to provider port of the network attached to the floating ip subnet
- Resubmit SNAT + Outbound NAPT flows to FIB table

Dependencies

None

Testing

Unit Tests

Integration Tests

CSIT

- Hairpinning between VMs in the same subnet
- Hairpinning between VMs in different subnets connected to the same router
- Hairpinning with NAPT - source VM is not associated with floating ip
- Traffic to external network with multiple subnets

Documentation Impact

None

References

[1] OpenDaylight Documentation Guide

Table of Contents

IPv6 DC-Internet L3 North-South connectivity using L3VPN provider network types.

https://git.opendaylight.org/gerrit/#/q/topic:ipv6-l3vpn-internet

In this specification we will be discussing the high level design of IPv6 Datacenter to Internet North-South connectivity support in OpenDaylight using L3VPN provider network type use-case.

Problem description

Provide IPv6 connectivity to virtual machines located in different subnets spread over multiple sites or Data center can be achieved through use of Globally Unique Addresses and capacity to update enough routing tables to forge a path between the two. Even if IPv6 is made to interconnect hosts without the help of any NAT mechanisms, routing with the best efficienty (shortest path) or policy (route weight, commercial relationships) must be configured using only few parameters, automatically updating routes for each VM spawned in new network.

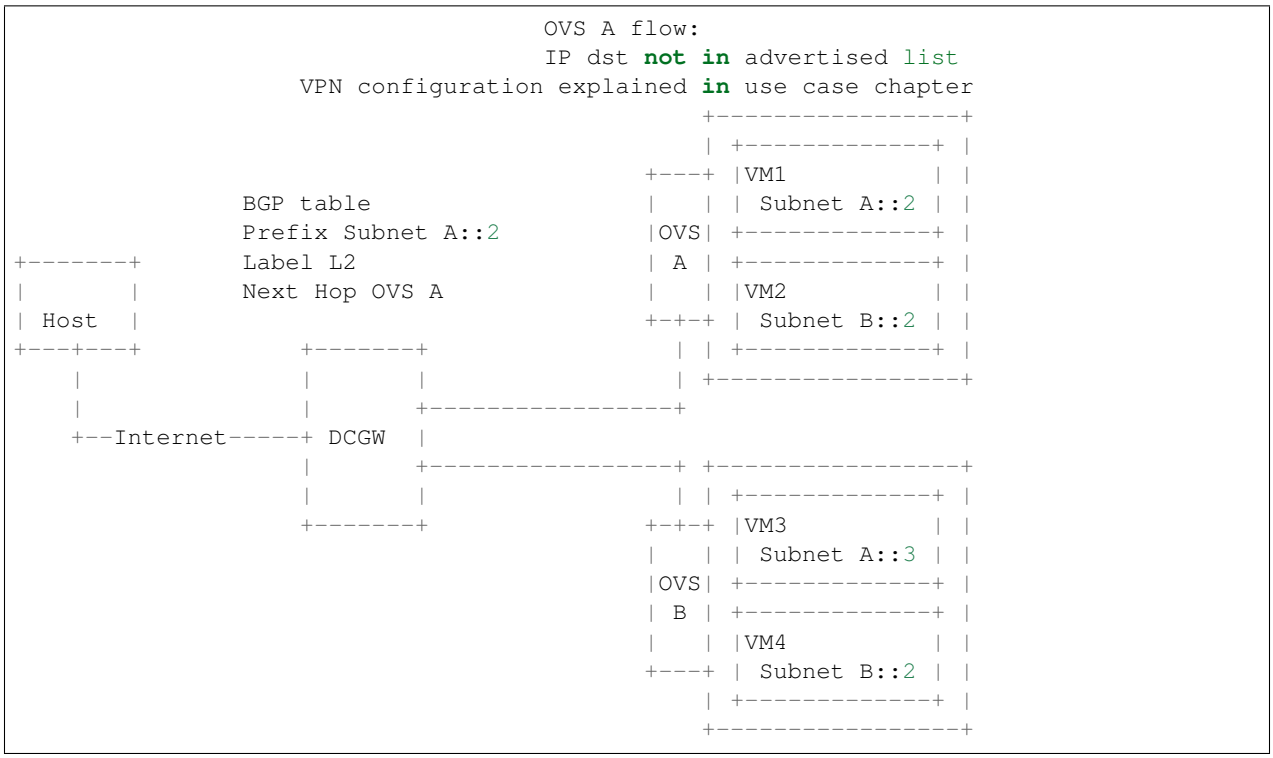
Keep in mind that key aspects of L3VPN connectivity is Route Targets and VPN-IPv6 address family. Assuming an operator can configure data center gateways with a Route Distinguisher dedicated to Internet connectivity and a set of imported Route Targets, each time a virtual machine is spawned within a data center subnet associated with that Route Distinguisher, it will trigger the send of a BGP UPDATE message containing MP-BGP attributes required for reaching the VM outside the datacenter. In the same manner, adding extra-route or declaring subnetworks will trigger

the same. Such behavior can be achieved by configuring a neutron router an internet public VPN address. For the following of the document, we focus to GUA/128 addresses that are advertised, when one VM start. Indeed, most of the requirements are dealing with VM access to internet.

Only IPv6 Globally Unique Address (eg /128) are advertised, this is not a scaling architecture since it implies as much routes to process as the number of spawned VMs, but with such BGP routing information base, DCGW can select the Compute Node to which a packet coming from the WAN should be forwarded to.

The following covers the case where a VM connects to a host located in the internet, and the destination ip address of packets is not part of the list of advertised prefixes (see spec [6]).

Following schema could help :



Use Cases

Datacenter IPv6 external connectivity to/from Internet for VMs spawned on tenant networks.

There are several techniques for VPNs to access the Internet. Those methods are described in [8], on section 11. Also a note describes in [8] the different techniques that could be applied to the DC-GW case. Note that not all solutions are compliant with the RFC. Also, we make the hypothesis of using GUA.

The method that will be described more in detail below is the option 2. Option 2 is external network connectivity option 2 from [8]). That method implies 2 VPNs. One VPN will be dedicated to Internet access, and will contain the Internet Routes, but also the VPNs routes. The Internet VPN can also contain default route to a gateway. Having a separated VPN brings some advantages: - the VPN that do not need to get Internet access get the private characteristic

of VPNs.

- using a VPN internet, instead of default forwarding table is enabling flexibility, since it coud permit creating more than one internet VPN. As consequence, it could permit applying different rules (different gateway for example).

Having 2 VPNs implies the following for one packet going from VPN to the internet. The FIB table will be used for that. If the packet's destination address does no match any route in the first VPN, then it may be matched against the internet VPN forwarding table. Reversely, in order for traffic to flow natively in the opposite direction, some of the routes from the VPN will be exported to the internet VPN.

Configuration steps in a datacenter:

- Configure ODL and Devstack networking-odl for BGP VPN.
- Create a tenant network with IPv6 subnet using GUA prefix or an

admin-created-shared-ipv6-subnet-pool. - This tenant network is connected to an external network where the DCGW is

connected. Separation between both networks is done by DPN located on compute nodes. The subnet on this external network is using the same tenant as an IPv4 subnet used for MPLS over GRE tunnels endpoints between DCGW and DPN on Compute nodes. Configure one GRE tunnel between DPN on compute node and DCGW.

- Create a Neutron Router and connect its ports to all internal subnets
- Create a transport zone to declare that a tunneling method is planned to reach an external IP:

the IPv6 interface of the DC-GW

- The neutron router subnetworks will be associated to two L3 BGPVPN instance.

  The step create the L3VPN instances and associate the instances to the router. Especially, two VPN instances will be created, one for the VPN, and one for the internetVPN.

  operations:neutronvpn:createL3VPN ("route-distinguisher" = "vpn1" "import-RT" = ["vpn1","internetvpn"] "export-RT" = ["vpn1","internetvpn"])

  operations:neutronvpn:createL3VPN ("route-distinguisher" = "internetvpn" "import-RT" = "internetvpn" "export-RT" = "internetvpn")

- The DC-GW configuration will also include 2 BGP VPN instances. Below is a configuration from QBGP using vty command interface.

vrf rd "internetvpn" vrf rt both "internetvpn" vrf rd "vpn1" vrf rt both "vpn1" "internetvpn"

- Spawn VM and bind its network interface to a subnet, L3 connectivty between

VM in datacenter and a host on WAN must be successful. More precisely, a route belonging to VPN1 will be associated to VM GUA. and will be sent to remote DC-GW. DC-GW will import the entry to both "vpn1" and "internetvpn" so that the route will be known on both vpns. Reversely, because DC-GW knows internet routes in "internetvpn", those routes will be sent to QBGP. ODL will get those internet routes, only in the "internetvpn" vpn. For example, when a VM will try to reach a remote, a first lookup will be done in "vpn1" FIB table. If none is found, a second lookup will be found in the "internetvpn" FIB table. The second lookup should be successfull, thus trigerring the encapsulation of packet to the DC-GW.

When the data centers is set up, there are 2 use cases:

- Traffic from Local DPN to DC-Gateway
- Traffic from DC-Gateway to Local DPN

The use cases are slightly different from [6], on the Tx side.

Proposed change

Similar as with [6], plus a specific processing on Tx side. An additionnal processing in DPN is required. When a packet is received by a neutron router associated with L3VPN, with destination mac address is the subnet gateway mac address, and the destination ip is not in the FIB (default gateway) of local DPN, then the packet should do a second lookup in the second VPN configured. So that the packet can enter the L3VPN netvirt pipeline. The MPLS label pushed on the IPv6 packet is the one configured to provide access to Internet at DCGW level.

Pipeline changes

No pipeline changes, compared with [6]. However, FIB Manager will be modified so as to implement the fallback mechanism. The FIB tables of the import-RTs VPNs from the default VPN created will be parsed. In our case, a match will be found in the "internetVPN" FIB table. If not match is found, the drop rule will be applied.

Regarding the pipeline changes, we can use the same BGPVPNv4 pipeline (Tables Dispatcher (17), DMAC (19), LFIB (20), L3FIB (21), and NextHop Group tables) and enhance those tables to support IPv6 North-South communication through MPLS/GRE. For understanding, the pipeline is written below: l3vpn-id is the ID associated to the initial VPN, while l3vpn-internet-id is the ID associated to the internet VPN.

Traffic from DC-Gateway to Local DPN (SYMMETRIC IRB)

When a packet is coming from DC-Gateway, the label will help finding out the associated VPN. The first one is l3vpn-id.

Classifier Table (0) =>
LFIB Table (20) `match: tun-id=mpls_label set vpn-id=l3vpn-id, pop_mpls label, set output to nexthopgroup-dst-vm` =>
NextHopGroup-dst-vm: `set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

Traffic from Local DPN to DC-Gateway (SYMMETRIC IRB)

When a packet is going out from a dedicated VM, the l3vpn-id attached to that subnetwork will be used. Theorically, in L3 FIB, there will be no match for dst IP with this l3vpn-id. However, because ODL know the relationship between both VPNs, then the dst IP will be attached with the first l3vpn-id.

However, since the gateway IP for inter-DC and external access is the same, the same MPLS label will be used for both VPNs.

Classifier Table (0) =>
Lport Dispatcher Table (17) ‘‘match: LportTag l3vpn service: set vpn-id=l3vpn-id‘ =>
DMAC Service Filter (19) `match: dst-mac=router-internal-interface-mac l3vpn service: set vpn-id=internet-l3vpn-id` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=<alternate-ip> set tun-id=mpls_label output to MPLSoGRE tunnel port` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=ext-ip-address set tun-id=mpls_label output to MPLSoGRE tunnel port` =>

Fib Manager changes

Ingress traffic from internet to VM

The FIB Manager is being configured with 2 entries for different RDs : l3vpn-id and internetvpn-id. The LFIB will be matched first. In our case, label NH and prefix are the same, whereas we have 2 VPN instances. So, proposed change is to prevent LFIB from adding entries if a label is already registered for that compute node.

Egress traffic from VM to internet

The FIB Manager is being configured with the internet routes on one RD only : internetvpn-id. As packets that are emitted from the VM with vpn=l3vpn-id, the internet route will not be matched in l3vpn, if implementation remains as it is. In FIB Manager, solution is the following: - The internetvpn is not attached to any local subnetwork. so, any eligible VPNs are looked up in the list of VPN instances. for each VPN instance, for each RD, if an imported RT matches the internetvpnID, then a new rule will be appended.

Yang changes

None

Configuration impact

The configuration will require to create 2 VPN instances.

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

The number of entries will be duplicated, compared with [6]. This is the cost in order to keep some VPNs private, and others kind of public. Another impact is the double lookup that may result, when emitting a packet. This is due to the fact that the whole fib should be parsed to fallback to the next VPN, in order to make an other search, so that the packet can enter in the L3VPN flow.

Targeted Release

Carbon

Alternatives

None

Usage

- Configure MPLS/GRE tunnel endpoint on DCGW connected to public-net network
- Configure neutron networking-odl plugin
- Configure BGP speaker in charge of retrieving prefixes for/from data center gateway in ODL through the set of vpnservice.bgpspeaker.host.name in etc/custom.properties. No REST API can configure that parameter. Use config/ebgp:bgp REST api to start BGP stack and configure VRF, address family and neighboring. In our case, as example, following values will be used:
  - rd="100:2" # internet VPN - import-rts="100:2" - export-rts="100:2"
  - rd="100:1" # vpn1 - import-rts="100:1 100:2" - export-rts="100:1 100:2"

```
POST config/ebgp:bgp
{
    "ebgp:as-id": {
          "ebgp:stalepath-time": "360",
          "ebgp:router-id": "<ip-bgp-stack>",
          "ebgp:announce-fbit": "true",
          "ebgp:local-as": "<as>"
    },
   "ebgp:neighbors": [
     {
       "ebgp:remote-as": "<as>",
       "ebgp:address-families": [
         {
           "ebgp:afi": "2",
           "ebgp:peer-ip": "<neighbor-ip-address>",
           "ebgp:safi": "128"
         }
       ],
       "ebgp:address": "<neighbor-ip-address>"
     }
   ],
}


* Configure BGP speaker on DCGW to exchange prefixes with ODL BGP stack. Since
 DCGW should be a vendor solution, the configuration of such equipment is out of
 the scope of this specification.
```

- Create an internal tenant network with an IPv6 (or dual-stack) subnet.

```
neutron net-create private-net
neutron subnet-create --name ipv6-int-subnet --ip-version 6
--ipv6-ra-mode slaac --ipv6-address-mode slaac private-net 2001:db8:0:2::/64
```

- Use neutronvpn:createL3VPN REST api to create L3VPN

```
POST /restconf/operations/neutronvpn:createL3VPN

{
```

```
   "input": {
      "l3vpn":[
         {
            "id":"vpnid_uuid_1",
            "name":"internetvpn",
            "route-distinguisher": [100:2],
            "export-RT": [100:2],
            "import-RT": [100:2],
            "tenant-id":"tenant_uuid"
         }
      ]
   }
}

POST /restconf/operations/neutronvpn:createL3VPN

{
   "input": {
      "l3vpn":[
         {
            "id":"vpnid_uuid_2",
            "name":"vpn1",
            "route-distinguisher": [100:1],
            "export-RT": [100:1, 100:2],
            "import-RT": [100:1, 100:2],
            "tenant-id":"tenant_uuid"
         }
      ]
   }
}
```

- Associate L3VPN To Network

```
POST /restconf/operations/neutronvpn:associateNetworks

{
   "input":{
     "vpn-id":"vpnid_uuid_1",
     "network-id":"network_uuid"
   }
}
```

- Spawn a VM in the tenant network

```
nova boot --image <image-id> --flavor <flavor-id> --nic net-id=<private-net> VM1
```

- Dump ODL BGP FIB

```
GET /restconf/config/odl-fib:fibEntries

{
  "fibEntries": {
    "vrfTables": [
      {
        "routeDistinguisher": <rd-uuid_1>
      },
      {
        "routeDistinguisher": <rd_vpn1>,
```

```
        "vrfEntry": [
          {
            "destPrefix": <IPv6_VM1/128>,
            "label": <label>,
            "nextHopAddressList": [
              <DPN_IPv4>
            ],
            "origin": "l"
          },
        ]
      }
      {
        "routeDistinguisher": <rd-uuid_2>
      },
      {
        "routeDistinguisher": <rd_vpninternet>,
        "vrfEntry": [
          {
            "destPrefix": <IPv6_VM1/128>,
            "label": <label>,
            "nextHopAddressList": [
              <DPN_IPv4>
            ],
            "origin": "l"
          },
        ]
      }
    ]
  }
}
```

Features to Install

odl-netvirt-openstack

REST API

CLI

Implementation

Assignee(s)

Primary assignee: Julien Courtat <julien.courtat@6wind.com>

Other contributors: Noel de Prandieres <prandieres@6wind.com> Valentina Krasnobaeva <valentina.krasnobaeva@6wind.com> Philippe Guibert <philippe.guibert@6wind.com>

Work Items

- Validate proposed setup so that each VM entry is duplicated in 2 VPN instances

- Implement FIB-Manager fallback mechanism for output packets

Dependencies

[6]

Testing

Unit Tests

Unit tests related to fallback mechanism when setting up 2 VPN instances configured as above.

Integration Tests

TBD

CSIT

CSIT provided for the BGPVPNv6 versions will be enhanced to also support connectivity to Internet.

Documentation Impact

Necessary documentation would be added on how to use this feature.

References

[1] OpenDaylight Documentation Guide

[2] https://specs.openstack.org/openstack/nova-specs/specs/kilo/template.html

[3] http://docs.openstack.org/developer/networking-bgpvpn/overview.html

[4] IPv6 Distributed Router for Flat/VLAN based Provider Networks.

[5] BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN

[6] Spec to support IPv6 Inter DC L3VPN connectivity using BGPVPN.

[7] Spec to support IPv6 North-South support for Flat/VLAN Provider Network.

[8] External Network connectivity in IPv6 networks.

[9] BGP/MPLS IP Virtual Private Networks (VPNs)

Table of Contents

IPv6 Inter-DC L3 North-South connectivity using L3VPN provider network types.

https://git.opendaylight.org/gerrit/#/q/topic:ipv6-interdc-l3vpn

In this specification we will be discussing the high level design of IPv6 Inter-Datacenter North-South connectivity support in OpenDaylight using L3VPN provider network type use-case.

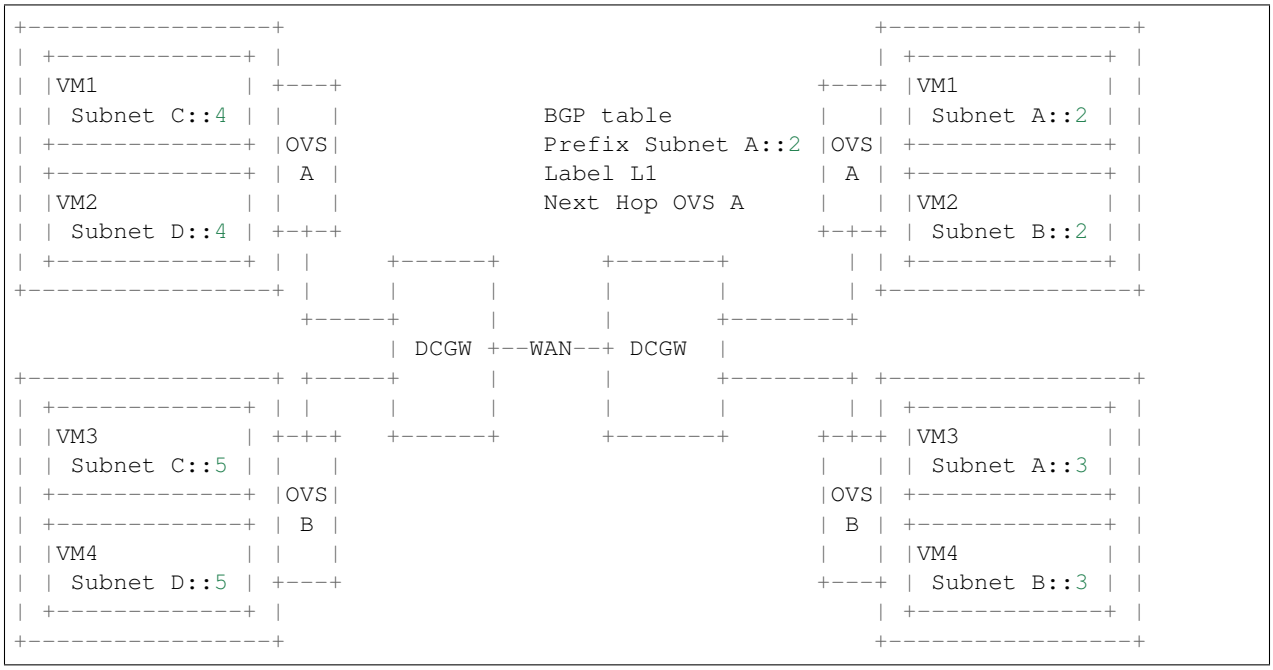
Problem description

Provide IPv6 connectivity to virtual machines located in different subnets spread over multiple sites or Data center can be achieved through use of Globally Unique Addresses and capacity to update enough routing tables to forge a path between the two. Even if IPv6 is made to interconnect hosts without the help of any NAT mechanisms, routing with the best efficienty (shortest path) or policy (route weight, commercial relationships) must be configured using only few parameters, automatically updating routes for each VM spawned in new network.

Keep in mind that key aspects of L3VPN connectivity is Route Targets and VPN-IPv6 address family. Assuming an operator can configure both data center gateways with same Route Distinguisher or set of imported Route Targets, each time a virtual machine is spawned within a new subnet, it will trigger the send of a BGP UPDATE message containing MP-BGP attributes required for reaching the VM. Such behavior can be achieved by configuring a neutron router a default gateway.

Only IPv6 Globally Unique Address (eg /128) are advertised, this is not a scaling architecture since it implies as much routes to process as the number of spawned VMs, but with such BGP routing information base, DCGW can select the Compute Node to which a packet coming from the WAN should be forwarded to.

Following schema could help :



BGP protocol and its MP-BGP extension would do the job as long as all BGP speakers are capable of processing UPDATE messages containing VPN-IPv6 address family, which AFI value is 2 and SAFI is 128. It is not required that BGP speakers peers using IPv6 LLA or GUA, IPv4 will be used to peer speakers together.

Opendaylight is already able to support the VPN-IPv4 address family (AFI=1, SAFI=128), and this blueprint focuses on specific requirements to VPN-IPv6.

One big question concerns the underlying transport IP version used with MPLS/GRE tunnels established between Data center Gateway (DCGW), and compute nodes (CNs). There is one MPLS/GRE tunnel setup from DCGW to each Compute Node involved in the L3VPN topology. Please note that this spec doesn't covers the case of VxLAN tunnels between DCGW and Compute Nodes.

According to RFC 4659 §3.2.1, the encoding of the nexthop attribute in MP-BGP UPDATE message differs if the tunneling transport version required is IPv4 or IPv6. In this blueprint spec, the assumption of transport IP version of IPv4 is prefered. This implies that any nexthop set for a prefix in FIB will be IPv4.

Within BGP RIB table, for each L3VPN entry, the nexthop and label are key elements for creating MPLS/GRE tunnel endpoints, and the prefix is used for programming netvirt pipeline. When a VM is spawned, the prefix advertised by BGP is 128 bits long and the nexthop carried along within UPDATE message is the ip address of the DPN interface used for DCGW connection. Since DCGW can be proprietary device, it may not support MPLS/GRE tunnel endpoint setup according to its internal BGP table. A static configuration of such tunnel endpoint may be required.

Use Cases

Inter Datacenter IPv6 external connectivity for VMs spawned on tenant networks, routes exchanged between BGP speakers using same Route Distinguisher.

Steps in both data centers :

- Configure ODL and Devstack networking-odl for BGP VPN.
- Create a tenant network with IPv6 subnet using GUA prefix or an admin-created-shared-ipv6-subnet-pool.
- This tenant network is separated to an external network where the DCGW is connected. Separation between both networks is done by DPN located on compute nodes. The subnet on this external network is using the same tenant as an IPv4 subnet used for MPLS over GRE tunnels endpoints between DCGW and DPN on Compute nodes. Configure one GRE tunnel between DPN on compute node and DCGW.
- Create a Neutron Router and connect its ports to all internal subnets that will belong to the same L3 BGPVPN identified by a Route Distinguisher.
- Start BGP stack managed by ODL, possibly on same host as ODL.
- Create L3VPN instance.
- Associate the Router with the L3VPN instance.
- Spawn VM on the tenant network, L3 connectivity between VMs located on different datacenter sharing same Route Distinguisher must be successful.

When both data centers are set up, there are 2 use cases per data center:

- Traffic from DC-Gateway to Local DPN (VMS on compute node)
- Traffic from Local DPN to DC-Gateway

Proposed change

ODL Controller would program the necessary pipeline flows to support IPv6 North South communication through MPLS/GRE tunnels out of compute node.

BGP manager would be updated to process BGP RIB when entries are IPv6 prefixes.

FIB manager would be updated to take into acount IPv6 prefixes.

Thrift interface between ODL and BGP implementation (Quagga BGP) must be enhanced to support new AFI=2. Thrift interface will still carry IPv4 Nexthops, and it will be the Quagga duty to transform this IPv4 Nexthop address into an IPv4-mapped IPv6 address in every NLRI fields. Here is the new api proposed :

```
enum af_afi {
    AFI_IP = 1,
    AFI_IPV6 = 2,
}
i32 pushRoute(1:string prefix, 2:string nexthop, 3:string rd, 4:i32 label,
              5:af_afi afi)
i32 withdrawRoute(1:string prefix, 2:string rd, 3:af_afi afi)
oneway void onUpdatePushRoute(1:string rd, 2:string prefix,
                              3:i32 prefixlen, 4:string nexthop,
                              5:i32 label, 6:af_afi afi)
oneway void onUpdateWithdrawRoute(1:string rd, 2:string prefix,
                                  3:i32 prefixlen, 4:string nexthop,
                                  5:af_afi afi)
Routes getRoutes(1:i32 optype, 2:i32 winSize, 3:af_afi afi)
```

BGP implementation (Quagga BGP) announcing (AFI=2,SAFI=128) capability as well as processing UPDATE messages with such address family. Note that the required changes in Quagga is not part of the design task covered by this blueprint.

Pipeline changes

Regarding the pipeline changes, we can use the same BGPVPNv4 pipeline (Tables Dispatcher (17), DMAC (19), LFIB (20), L3FIB (21), and NextHop Group tables) and enhance those tables to support IPv6 North-South communication through MPLS/GRE.

Traffic from DC-Gateway to Local DPN (SYMMETRIC IRB)

Classifier Table (0) =>
LFIB Table (20) `match: tun-id=mpls_label set vpn-id=l3vpn-id, pop_mpls label, set output to nexthopgroup-dst-vm` =>
NextHopGroup-dst-vm: `set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

Please note that `vpn-subnet-gateway-mac-address` stands for MAC address of the neutron port of the internal subnet gateway router.

Traffic from Local DPN to DC-Gateway (SYMMETRIC IRB)

Classifier Table (0) =>
Lport Dispatcher Table (17) `match: LportTag l3vpn service: set vpn-id=l3vpn-id` =>
DMAC Service Filter (19) `match: dst-mac=router-internal-interface-mac l3vpn service: set vpn-id=l3vpn-id` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=ext-ip-address set tun-id=mpls_label output to MPLSoGRE tunnel port` =>

Please note that `router-internal-interface-mac` stands for MAC address of the neutron port of the internal subnet gateway router.

Yang changes

Changes will be needed in `ebgp.yang` to start supporting IPv6 networks advertisements.

A new leaf afi will be added to container `networks`

Listing 1.11: ebgp.yang

```
list networks {
    key "rd prefix-len";

    leaf rd {
          type string;
    }
```

```
    leaf prefix-len {
          type string;
    }

    leaf afi {
          type uint32;
          mandatory "false";
    }

    leaf nexthop {
          type inet:ipv4-address;
          mandatory "false";
    }

    leaf label {
          type uint32;
          mandatory "false";
    }
}
```

Configuration impact

None

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

Impact on scaling inside datacenter essentially grow with the number of VM connected to subnets associated with the L3VPN. Since Globally Unique Address are used and there is no NAT involved in the datapath, it implies prefixes advertised are all /128. At the end, it means that every prefix advertised will have its entry in BGP RIB of all ODL controllers and DCGW involved in L3VPN (ie all bgp aware equipment will handle all prefixes advertised wihtin a Route Distinguisher).

This may imply BGP table with very high number of entries. This also implies a high number of entries in ODL routing table and equivalent number of flows inserted in OVS, since prefix advertised add matching ip destination in OVS tables.

This fact also impact the scaling of the BGP speaker implementation (Quagga BGP) with many thousands of BGPVPNv4 and BGPVPNv6 prefixes (as much as number of spawned VMs) with best path selection algorithm on route updates, graceful restart procedure, and multipath.

Targeted Release

Carbon

Alternatives

None

Usage

- Configure MPLS/GRE tunnel endpoint on DCGW connected to public-net network
- Configure neutron networking-odl plugin
- Configure BGP speaker in charge of retrieving prefixes for/from data center gateway in ODL through the set of vpnservice.bgpspeaker.host.name in etc/custom.properties. No REST API can configure that parameter. Use config/ebgp:bgp REST api to start BGP stack and configure VRF, address family and neighboring

```
POST config/ebgp:bgp
{
    "ebgp:as-id": {
          "ebgp:stalepath-time": "360",
          "ebgp:router-id": "<ip-bgp-stack>",
          "ebgp:announce-fbit": "true",
          "ebgp:local-as": "<as>"
    },
    "ebgp:vrfs": [
     {
       "ebgp:export-rts": [
         "<export-rts>"
       ],
       "ebgp:rd": "<RD>",
       "ebgp:import-rts": [
         "<import-rts>"
       ]
     }
   ],
   "ebgp:neighbors": [
     {
       "ebgp:remote-as": "<as>",
       "ebgp:address-families": [
         {
           "ebgp:afi": "2",
           "ebgp:peer-ip": "<neighbor-ip-address>",
           "ebgp:safi": "128"
         }
       ],
       "ebgp:address": "<neighbor-ip-address>"
     }
   ],
}
```

- Configure BGP speaker on DCGW to exchange prefixes with ODL BGP stack. Since DCGW should be a vendor solution, the configuration of such equipment is out of the scope of this specification.
- Create an internal tenant network with an IPv6 (or dual-stack) subnet and connect ports.

```
neutron net-create private-net
neutron subnet-create private-net 2001:db8:0:2::/64 --name ipv6-int-subnet
--ip-version 6 --ipv6-ra-mode slaac --ipv6-address-mode slaac
neutron port-create private-net --name port1_private1
```

- Create a router and associate it to internal subnets.

```
neutron router-create router1
neutron router-interface-add router1 ipv6-int-subnet
```

- Use neutronvpn:createL3VPN REST api to create L3VPN

```
POST /restconf/operations/neutronvpn:createL3VPN
{
   "input": {
      "l3vpn":[
         {
            "id":"vpnid_uuid",
            "name":"vpn1",
            "route-distinguisher": [100:1],
            "export-RT": [100:1],
            "import-RT": [100:1],
            "tenant-id":"tenant_uuid"
         }
      ]
   }
}
```

- Associate L3VPN To Routers

```
POST /restconf/operations/neutronvpn:associateRouter
{
  "input":{
     "vpn-id":"vpnid_uuid",
     "router-id":[ "router_uuid" ]
   }
}
```

- Create MPLSoGRE tunnel between DPN and DCGW

```
POST /restconf/operations/itm-rpc:add-external-tunnel-endpoint
{
  "itm-rpc:input": {
    "itm-rpc:destination-ip": "dcgw_ip",
    "itm-rpc:tunnel-type": "odl-interface:tunnel-type-mpls-over-gre"
  }
}
```

- Spawn a VM in the tenant network

```
nova boot --image <image-id> --flavor <flavor-id> \
     --nic net-id=port1_private1_uuid VM1
```

- Dump ODL BGP FIB

```
GET /restconf/config/odl-fib:fibEntries
{
  "fibEntries": {
```

```
    "vrfTables": [
      {
        "routeDistinguisher": <rd-uuid>
      },
      {
        "routeDistinguisher": <rd>,
        "vrfEntry": [
          {
            "destPrefix": <IPv6_VM1/128>,
            "label": <label>,
            "nextHopAddressList": [
              <DPN_IPv4>
            ],
            "origin": "l"
          },
        ]
      }
    ]
  }
}
```

Features to Install

odl-netvirt-openstack

REST API

CLI

A new option `--afi` will be added to command `odl:bgp-network`:

```
opendaylight-user@root>
odl:bgp-network --prefix 2001:db8::1/128 --rd 100:1 --nexthop 192.168.0.2
                --label 700 --afi 2 add/del
```

Implementation

Assignee(s)

Primary assignee: Julien Courtat <julien.courtat@6wind.com>

Other contributors: Noel de Prandieres <prandieres@6wind.com> Valentina Krasnobaeva <valentina.krasnobaeva@6wind.com> Philippe Guibert <philippe.guibert@6wind.com>

Work Items

- Implement necessary APIs to allocate a transport over IPv6 requirement configuration for a given Route Target as the primary key.
- Support of BGPVPNv6 prefixes within MD-SAL. Enhance RIB-manager to support routes learned from other bgp speakers, [un]set static routes.

- BGP speaker implementation, Quagga BGP, to support BGPVPN6 prefixes exchanges with other BGP speakers (interoperability), and thrift interface updates.
- Program necessary pipeline flows to support IPv6 to MPLS/GRE (IPv4) communication.

Dependencies

Quagga from 6WIND is publicly available at the following url

- https://github.com/6WIND/quagga
- https://github.com/6WIND/zrpcd

Testing

Unit Tests

Unit tests provided for the BGPVPNv4 versions will be enhanced to also support BGPVPNv6. No additional unit tests will be proposed.

Integration Tests

TBD

CSIT

CSIT provided for the BGPVPNv4 versions will be enhanced to also support BGPVPNv6. No additional CSIT will be proposed.

Documentation Impact

Necessary documentation would be added on how to use this feature.

References

[1] OpenDaylight Documentation Guide

[2] https://specs.openstack.org/openstack/nova-specs/specs/kilo/template.html

[3] http://docs.openstack.org/developer/networking-bgpvpn/overview.html

[4] Spec to support IPv6 North-South support for Flat/VLAN Provider Network.

[5] BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN

Note: This template was derived from [2], and has been modified to support our project.

Table of Contents

IPv6 L3 North-South support for Flat/VLAN Provider Networks.

https://git.opendaylight.org/gerrit/#/q/topic:ipv6-cvr-north-south

In this specification we will be discussing the high level design of IPv6 North-South support in OpenDaylight for VLAN/FLAT provider network use-case.

Problem description

OpenDaylight currently supports IPv6 IPAM (IP Address Management) and a fully distributed east-west router. IPv6 external connectivity is not yet supported. This SPEC captures the implementation details of IPv6 external connectivity for VLAN/FLAT provider network use-cases.

We have a separate SPEC [3] that captures external connectivity for L3VPN use-case.

The expectation in OpenStack is that Tenant IPv6 subnets are created with Globally Unique Addresses (GUA) that are routable by the external physical IPv6 gateway in the datacenter for external connectivity. So, there is no concept of NAT or Floating-IPs for IPv6 addresses in Neutron. An IPv6 router is hence expected to do a plain forwarding.

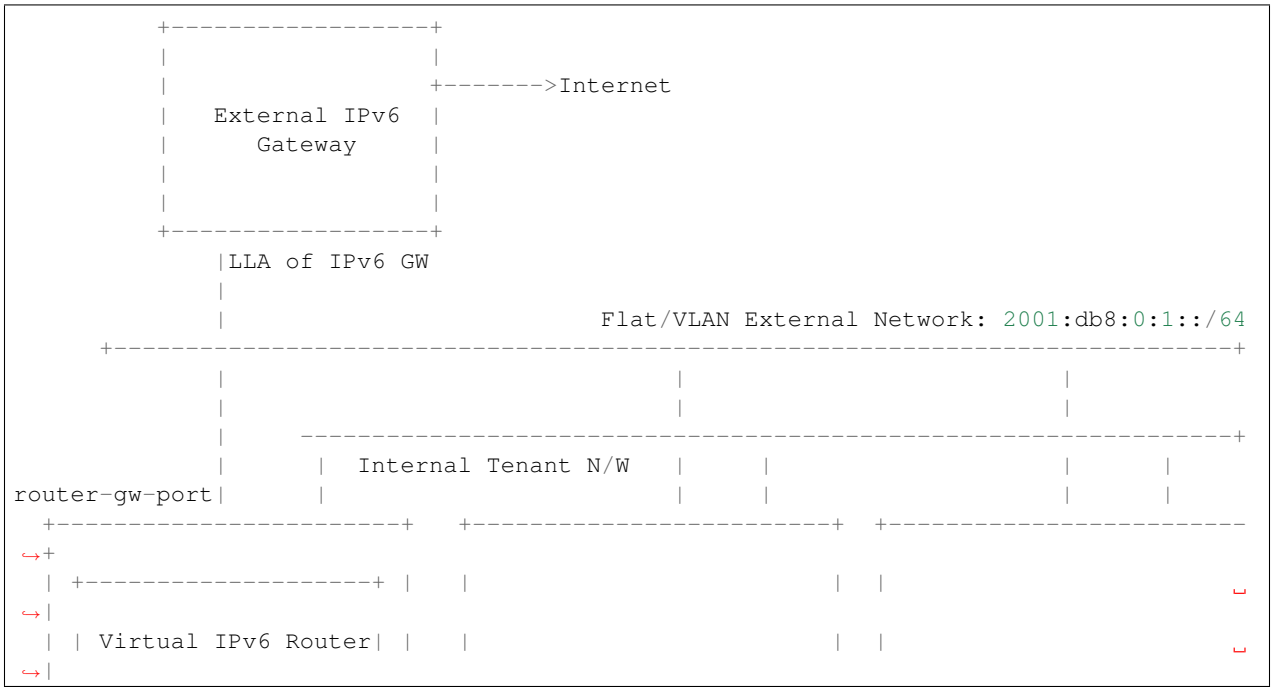
Initially, we would like to pursue a Centralized IPv6 router (CVR) use-case and look into a fully distributed router via a future spec. One of the main reasons for pursuing the CVR over DVR is that OpenStack Neutron creates only a single router gateway port (i.e., port with device owner as network:router_gateway) when the router is associated with the external network. When implementing a distributed router, we cannot use the same router gateway port MAC address from multiple Compute nodes as it could create issues in the underlying physical switches. In order to implement a fully distributed router, we would ideally require a router-gateway-port per compute node. We will be addressing the distributed router in a future spec taking into consideration both IPv4 and IPv6 use-cases.

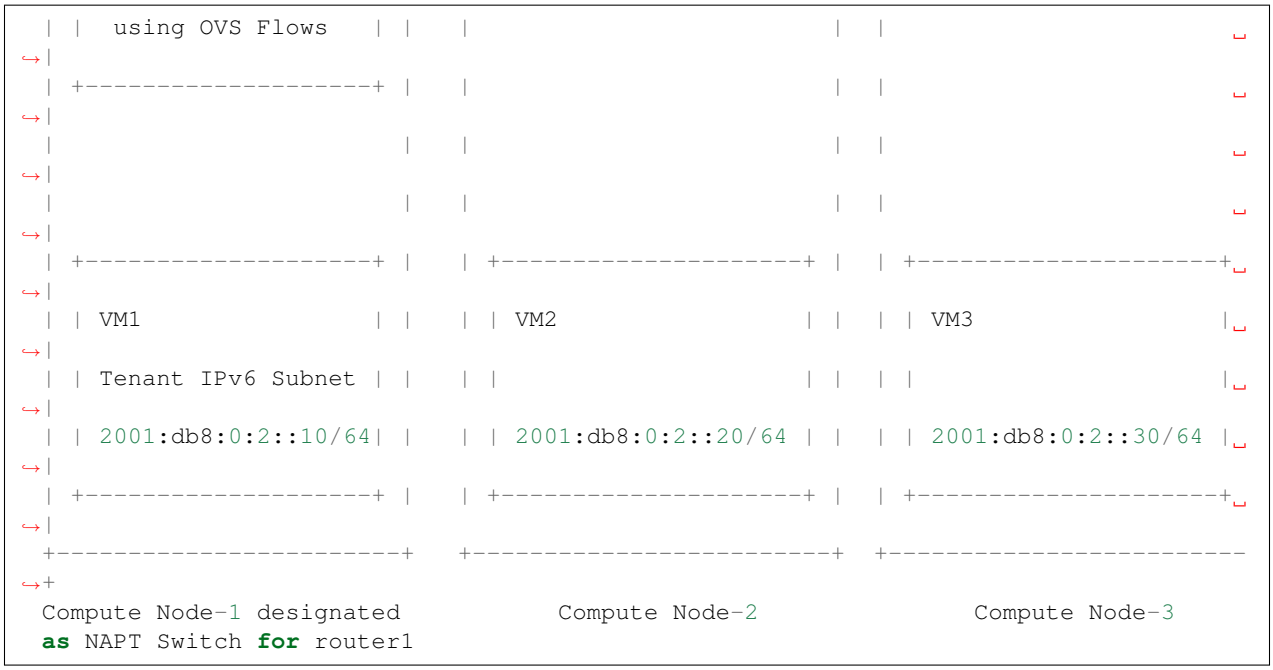
Use Cases

IPv6 external connectivity (north-south) for VMs spawned on tenant networks, when the external network is of type FLAT/VLAN based.

Steps:

- Create a tenant network with IPv6 subnet using GUA/ULA prefix or an admin-created-shared-ipv6-subnet-pool.
- Create an external network of type FLAT/VLAN with an IPv6 subnet where the gateway_ip points to the Link Local Address (LLA) of external/physical IPv6 gateway.
- Create a Neutron Router and associate it with the internal subnets and external network.
- Spawn VMs on the tenant network.





Proposed change

ODL Controller would implement the following.

- Program the necessary pipeline flows to support IPv6 forwarding
- Support Neighbor Discovery for Router Gateway port-ips on the external network. i.e., When the upstream/external IPv6 Gateway does a Neighbor Solicitation for the router-gateway-ip, ODL-Controller/ipv6service would respond with a Neighbor Advertisement providing the target link layer address.
- Enhance IPv6Service to learn the MAC-address of external-subnet-gateway-ip by framing the necessary Neighbor Solicitation messages and parsing the corresponding response. The APIs in IPv6Service would be triggered from Gateway MAC resolver code and the information obtained will be used while programming the Provider-NetworkGroup entries.

The implementation would be aligned with the existing IPv4 SNAT support we have in Netvirt. ODL controller would designate one of the compute nodes (also referred as NAPT Switch), one per router, to act as an IPv6/IPv4-SNAT router, from where the tenant traffic is routed to the external network. External traffic from VMs hosted on the NAPT switch is forwarded directly, whereas traffic from VMs hosted on other compute nodes would have to do an extra hop to NAPT switch before hitting the external network. If a router has both IPv4 and IPv6 subnets, the same NAPT Switch for the router will be used for IPv4-SNAT and IPV6 external-packet forwarding.

Pipeline changes

Flows on NAPT Switch for Egress traffic from VM to the internet

Classifier Table (0) =>
LPORT_DISPATCHER_TABLE (17) `l3vpn service: set: vpn-id=router-id` =>
L3_GW_MAC_TABLE (19) `priority=20, match: vpn-id=router-id, dst-mac=router-internal-interface-mac` =>

L3_FIB_TABLE (21) `priority=10, match: ipv6, vpn-id=router-id, default-route-flow` =>
PSNAT_TABLE (26) `priority=5, match: ipv6, vpn-id=router-id, unknown-sip` =>
OUTBOUND_NAPT_TABLE (46) `priority=10, match: ipv6, vpn-id=router-id, ip-src=vm-ip set: src-mac=external-router-gateway-mac-address, vpn-id=external-net-id,` =>
NAPT_PFIB_TABLE (47) `priority=6, match: ipv6, vpn-id=external-net-id, src-ip=vm-ip` =>
ProviderNetworkGroup: `set dst-mac=ext-subnet-gw-mac, reg6=provider-lport-tag` =>
EGRESS_LPORT_DISPATCHER_TABLE (220) output to provider network

Flows on NAPT Switch for Ingress traffic from internet to VM

Classifier Table (0) =>
LPORT_DISPATCHER_TABLE (17) `l3vpn service: set: vpn-id=ext-net-id` =>
L3_GW_MAC_TABLE (19) `priority=20, match: vpn-id=ext-net-id, dst-mac=router-gateway-mac` =>
L3_FIB_TABLE (21) `priority=138, match: ipv6, vpn-id=ext-net-id, dst-ip=vm-ip` =>
INBOUND_NAPT_TABLE (44) `priority=10, match: ipv6, vpn-id=ext-net-id, dst-ip=vm-ip set: vpn-id=router-id` =>
NAPT_PFIB_TABLE (47) `priority=5, match: ipv6, vpn-id=router-id set: in_port=0` =>
L3_FIB_TABLE (21) `priority=138, match: ipv6, vpn-id=router-id, dst-ip=vm-ip` =>
Local Next-Hop group: `set: src-mac=router-intf-mac, dst-mac=vm-mac,reg6=vm-lport-tag` =>
Egress table (220) output to VM port

Flows for VMs hosted on Compute node that is not acting as an NAPT Switch

Same egress pipeline flows as above until L3_FIB_TABLE (21).
PSNAT_TABLE (26) `priority=5, match: ipv6, vpn-id=router-id set: tun_id=<tunnel-id>` =>
TunnelOutputGroup: `output to tunnel-port` =>
OnNAPTSwitch (for Egress Traffic from VM)

> INTERNAL_TUNNEL_TABLE (36): `priority=10, match: ipv6, tun_id=<tunnel-id-set-on-compute-node> set: vpn-id=router-id, goto_table:46`
> Rest of the flows are common.

OnNAPTSwitch (for Ingress Traffic from Internet to VM)

> Same flows in ingress pipeline shown above until NAPT_PFIB_TABLE (47) =>
> L3_FIB_TABLE (21) `priority=138, match: ipv6, vpn-id=router-id, dst-ip=vm-ip set: tun_id=<tunnel-id>, dst-mac=vm-mac, output: <tunnel-port>` =>

Yang changes

IPv6Service would implement the following YANG model.

```
module ipv6-ndutil {
  yang-version 1;
  namespace "urn:opendaylight:netvirt:ipv6service:ipv6util";
  prefix "ipv6-ndutil";

  import ietf-interfaces {
      prefix if;
  }

  import ietf-inet-types {
      prefix inet; revision-date 2013-07-15;
  }

  import ietf-yang-types {
      prefix yang;
  }

  revision "2017-02-10" {
      description "IPv6 Neighbor Discovery Util module";
  }

  grouping interfaces {
      list interface-address {
         key interface;
         leaf interface {
              type leafref {
                  path "/if:interfaces/if:interface/if:name";
              }
         }
         leaf src-ip-address {
              type inet:ipv6-address;
         }
         leaf src-mac-address {
              type yang:phys-address;
          }
      }
  }

  rpc send-neighbor-solicitation {
      input {
          leaf target-ip-address {
              type inet:ipv6-address;
          }
          uses interfaces;
      }
  }
}
```

neighbor-solicitation-packet container in neighbor-discovery.yang would be enhanced with Source Link Layer optional header.

```
container neighbor-solicitation-packet {
  uses ethernet-header;
  uses ipv6-header;
```

```
  uses icmp6-header;
  leaf reserved {
      type uint32;
  }
  leaf target-ip-address {
      type inet:ipv6-address;
  }
  leaf option-type {
      type uint8;
  }
  leaf source-addr-length {
      type uint8;
  }
  leaf source-ll-address {
      type yang:mac-address;
  }
}
```

Configuration impact

None

Clustering considerations

None

Other Infra considerations

None

Security considerations

Scale and Performance Impact

- In the proposed implementation, we have to configure a static route on the external IPv6 Gateway with next-hop as the router-gateway-ip. In a future patch, we would enhance the implementation to use BGP for advertising the necessary routes.
- When the external IPv6 Gateway wants to contact the tenant VMs, it forwards all the traffic to the router-gateway-port on the designated NAPT Switch. To know the target-link-layer address of the router-gw-port, the external IPv6 Gateway would send out a Neighbor Solicitation for the router-gateway-port-ip. This request would be punted to the Controller and ipv6service would respond with the corresponding Neighbor Advertisement. In large deployments this can become a bottleneck. Note: Currently, OpenFlow does not have support to auto-respond to Neighbor Solicitation packets like IPv4 ARP. When the corresponding support is added in OpenFlow, we would program the necessary ovs flows to auto-respond to the Neighbor Soliciation requests for router-gateway-ports.

Targeted Release

Carbon

Alternatives

An alternate solution is to implement a fully distributed IPv6 router and would be pursued in a future SPEC.

Usage

- Create an external FLAT/VLAN network with an IPv6 (or dual-stack) subnet.

```
neutron net-create public-net -- --router:external --is-default
--provider:network_type=flat --provider:physical_network=public

neutron subnet-create --ip_version 6 --name ipv6-public-subnet
--gateway <LLA-of-external-ipv6-gateway> <public-net-uuid> 2001:db8:0:1::/64
```

- Create an internal tenant network with an IPv6 (or dual-stack) subnet.

```
neutron net-create private-net
neutron subnet-create --name ipv6-int-subnet --ip-version 6
--ipv6-ra-mode slaac --ipv6-address-mode slaac private-net 2001:db8:0:2::/64
```

- Create a router and associate the external and internal subnets. Explicitly specify the fixed_ip of router-gateway-port, as it would help us when manually configuring the downstream route on the external IPv6 Gateway.

```
neutron router-create router1
neutron router-gateway-set --fixed-ip subnet_id=<ipv6-public-subnet-id>,ip_
↪address=2001:db8:0:10 router1 public-net
neutron router-interface-add router1 ipv6-int-subnet
```

- Manually configure a downstream route in the external IPv6 gateway for the IPv6 subnet "2001:db8:0:2::/64" with next hop address as the router-gateway-ip.

```
Example (on Linux host acting as an external IPv6 gateway):
ip -6 route add 2001:db8:0:2::/64 via 2001:db8:0:10
```

- Spawn a VM in the tenant network

```
nova boot --image <image-id> --flavor <flavor-id> --nic net-id=<private-net> VM1
```

Features to Install

odl-netvirt-openstack

REST API

CLI

Implementation

Assignee(s)

Primary assignee: Sridhar Gaddam <sgaddam@redhat.com>

Other contributors: TBD

Work Items

https://trello.com/c/cqjOFmow/147-ipv6-centralized-router-l3-north-south-support-for-flat-vlan-provider-networks

- Program necessary pipeline flows to support IPv6 North-South communication.
- Enhance ipv6service to send out Neighbor Solicitation requests for the external/physical IPv6 gateway-ip and parse the response.
- Support controller based Neighbor Advertisement for router-gateway-ports on the external network.
- Implement Unit and Integration tests to validate the use-case.

Dependencies

None

Testing

Unit Tests

Necessary Unit tests would be added to validate the use-case.

Integration Tests

Necessary Integration tests would be added to validate the use-case.

CSIT

We shall explore the possibility to validate this use-case in CSIT.

Documentation Impact

Necessary documentation would be added on how to use this feature.

References

[1] OpenDaylight Documentation Guide

[2] https://specs.openstack.org/openstack/nova-specs/specs/kilo/template.html

[3] Spec to support IPv6 Inter DC L3VPN connectivity using BGPVPN

Note: This template was derived from [2], and has been modified to support our project.

Table of Contents

Dual Stack VM support in OpenDaylight

https://git.opendaylight.org/gerrit/#/q/topic:l3vpn-dual-stack-vms

In this specification we will introduce a support of basic L3 forwarding for dualstack VMs connectivity over L3 in NetVirt. Dualstack VM is a virtual machine that has at least two IP addresses with different ethertypes: IPv4 address and IPv6 address.

In addition to this, the specification ensures initial support of dualstack VMs inside L3 BGPVPN. L3 forwarding for dualstack VMs connectivity inside L3 BGPVPN will be provided for the following variations of L3 BGPVPN:

1. L3 BGPVPN constructed purely using networks;
2. L3 BGPVPN constructed purely using a router;
3. L3 BGPVPN constructed using multiple networks and a router.

Problem description

As a dualstack VM, we assume a VM which has one Neutron Port, i.e. one VNIC, that inherits two IPs addresses with different ethertypes: one IPv4 address and one IPv6 address. We also will use in this document a term singlestack VM to describe a VM, which VNIC possesses either IPv4 or IPv6 address, but not both simultaneously.

So, dualstack VM has two IP addresses with different ethertypes. This could be achieved by two ways:

1. VM was initially created with one VNIC, i.e. one Neutron Port from network with IPv4 subnet. Second VNIC, corresponded to a Neutron Port from another network with IPv6 subnet, was added to this machine after its creation.

2. VM has one Neutron Port from a network, which contains 2 subnets: IPv4 subnet and IPv6 subnet.

OpenDaylight has already provided a support for the first way, so this use-case is not in the scope of the specification. For the second way the specification doesn't intend to cover a use-case when, Neutron Port will possess several IPv4 and several IPv6 addresses. More specifically this specification covers only the use-case, when Neutron Port has only one IPv4 and one IPv6 address.

Since there are more and more services that use IPv6 by default, support of dualstack VMs is important. Usage of IPv6 GUA addresses has increased during the last couple years. Administrators want to deploy services, which will be accessible from traditional IPv4 infrastructures and from new IPv6 networks as well.

Dualstack VM should be able to connect to other VMs, be they are of IPv4 (or) IPv6 ethertypes. So in this document we can handle following use cases:

- Intra DC, Inter-Subnet basic L3 Forwarding support for dualstack VMs;
- Intra DC, Inter-Subnet L3 Forwarding support for dualstack VMs within L3 BGPVPN.

Current L3 BGPVPN allocation scheme picks up only the first IP address of dualstack VM Neutron Port. That means that the L3 BGPVPN allocation scheme will not apply both IPv4 and IPv6 network configurations for a port. For example, if the first allocated IP address is IPv4 address, then L3 BGPVPN allocation scheme will only apply to IPv4 network configuration. The second IPv6 address will be ignored.

Separate VPN connectivity for singlestack VMs within IPv4 subnetworks and within IPv6 subnetworks is already achieved by using distinct L3 BGPVPN instances. What we want is to support a case, when the same L3 BGPVPN instance will handle both IPV4 and IPv6 VM connectivity.

Regarding the problem description above, we would propose to implement in OpenDaylight two following solutions, applying to two setups

1. **two-router** setup solution

One router belongs to IPv4 subnetwork, another one belongs to IPv6 subnetwork. This setup brings flexibility to manage access to external networks. More specifically, by having two routers, where one is holding IPv4 subnet and another is holding IPv6 subnet, customer can tear-down access to external network for IPv4 subnet ONLY or for IPv6 subnet ONLY by doing a router-gateway-clear on a respective router.

Now this kind of orchestration step entail us to put a Single VPN Interface (representing the VNIC of DualStack VM) in two different Internal-VPNs, where each VPN represents one of the routers. To achive this we will use L3 BGPVPN concept. We will extend existing L3 BGPVPN instance implementation to give it an ability to be associated with two routers. As consequence, IPv4 and IPv6 subnetworks, added as ports in associated routers and, hence, IPv4 and IPv6 FIB entries, would be gathered in one L3 BGPVPN instance.

L3 BGPVPN concept is the easiest solution to federate two routers in a single L3 BGPVPN entity. From the orchestration point of view and from the networking point of view, there is no any reason to provide IPv4 L3VPN and IPv6 L3VPN access separately for dualstack VMs. It makes sense to have the same L3 BGPVPN entity that can handle both IPv4 and IPv6 subnetworks.

The external network connectivity using L3 BGPVPN is not in scope of this specification. Please, find more details about this in [6]. Right now, this configuration will be useful for inter-subnet and intra-dc routing.

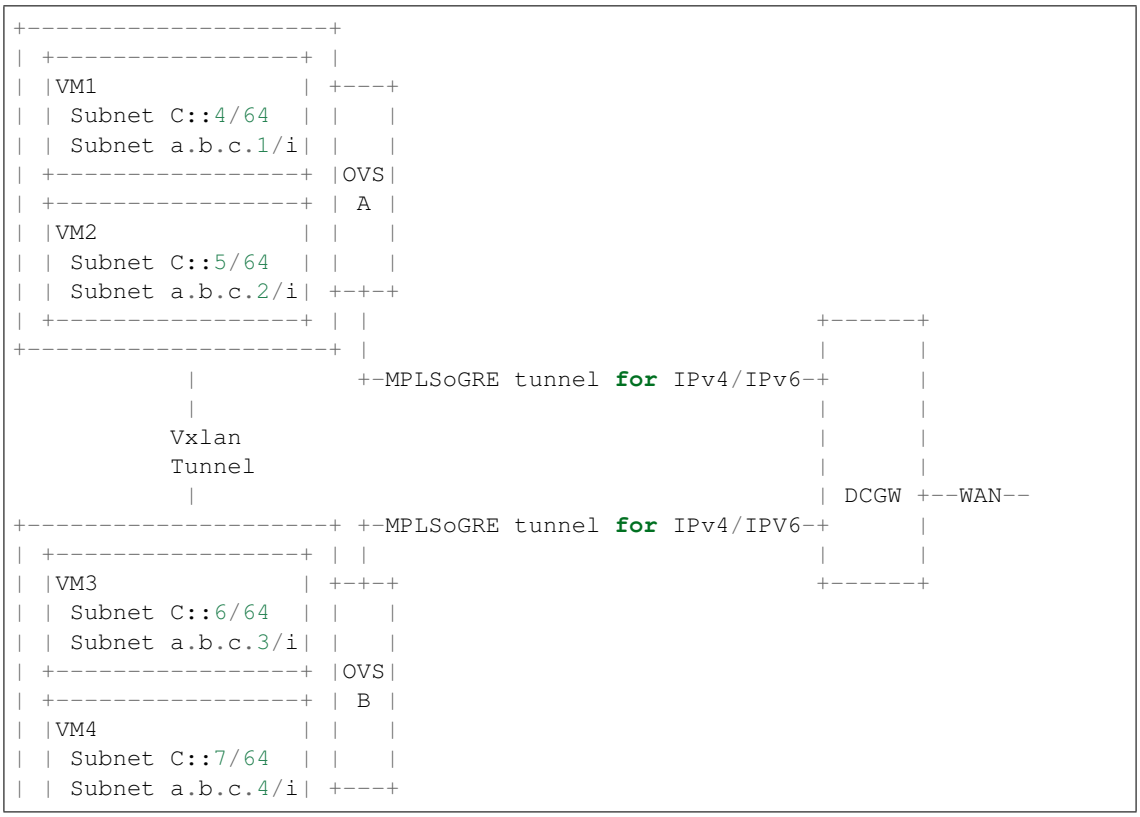
2. **dualstack-router** setup solution

The router with 2 ports (one port for IPv4 subnet and another one for IPv6 subnet) is attached to a L3 BGPVPN instance.

The external network connectivity using L3 BGPVPN is not in the scope of this specification.

Setup Presentation

Following drawing could help :



```
| +-----------------+ |
+---------------------+
```

We identify there 2 subnets:

- IPv4 subnet: a.b.c.x/i
- IPv6 subnet: C::x/64

Each VM will receive IPs from these two defined subnets.

Following schemes stand for conceptual representation of used neutron configurations for each proposed solution.

```
setup 1: two singlestack routers, associated with one BGPVPN
         ("two-router" solution)

                                      +---------------+
                                      | Network N3    |
                                      +---------------+
      +-----+     +----------------+  | Subnet C IPv4 |
      | VM1 |-----| Network N      |  +---------------+
      +-----+  +--|                |          |
               |  +----------------+  +---------------+
               |  | Subnet A IPv4 |----| Router 1      |-----+
               |  +----------------+  +---------------+     |
               |  | Subnet B IPv6 |          |              |   +--------+
               |  +----------------+  +---------------+     |   |        |
               |          |           | Subnet E IPv4 |     |---+ BGPVPN |
               |          |           +---------------+     |   |        |
               |          |           | Network N2    |     |   +--------+
               |          |           +---------------+     |
               |  +----------------+                        |
               |  | Router 2      |------------------------+
      +-----+  |  +----------------+
      | VM2 |--+          |
      +-----+     +----------------+
                  | Subnet D IPv6 |
                  +----------------+
                  | Network N1    |
                  +----------------+
```

Network N gathers 2 subnetworks, subnet A IPv4 and subnet B IPv6. This makes possible to create Neutron Ports, which will have 2 IP addresses and whose attributes will inherit information (extraroutes, etc) from these 2 subnets A and B.

Router1 and Router2 are connected to Subnet A and Subnet B respectively and will be attached to a same L3 BGPVPN instance. Routers 1 and 2 can also have other ports, but they always should stay singlestack routers, otherwise this configuration will not be still supported. See the chapter "Configuration impact" for more details.

```
setup 2: one dualstack router associated with one BGPVPN
         ("dualstack-router" solution)

       +-----+     +----------------+
       | VM1 |-----| Network N      |
       +-----+  +--|                |
                |  +----------------+          +----------+   +--------+
                |  | Subnet A IPv4 |----------|          |   |        |
                |  +----------------+          | Router 1 |---+ BGPVPN |
                |  | Subnet B IPv6 |----------|          |   |        |
```

```
                |  +---------------+          +----------+    +--------+
        +-----+ |
        | VM2 |--+
        +-----+
```

Network N gathers 2 subnetworks, subnet A IPv4 and subnet B IPv6. This makes possible to create Neutron Ports, which will have 2 IP addresses and whose attributes will inherit information (extraroutes, etc) from these 2 subnets A and B.

Router 1 is connected to Subnet A and Subnet B, and it will be attached to a L3 BGPVPN instance X. Other subnets can be added to Router 1, but this configurations will not be still supported. See the chapter "Configuration impact" for more details.

```
setup 3: networks associated with one BGPVPN

        +-----+     +-------------------+      +--------+
        | VM1 |-----| Network N1        |------| BGPVPN |
        +-----+  +--|                   |      |        |
                 |  +-------------------+      +--------+
                 |  | Subnet A IPv4 (1)|           |
        +-----+  |  +-------------------+           |
        | VM2 |--+  | Subnet B IPv6 (2)|           |
        +-----+     +-------------------+           |
                                                    |
                                                    |
        +-----+     +-------------------+           |
        | VM3 |-----+ Network N2        |-----------+
        +-----+     |                   |
                    +-------------------+
                    | Subnet C IPv4 (3)|
                    +-------------------+
                    | Subnet D IPv6 (4)|
                    +-------------------+
```

Network N1 gathers 2 subnets, subnet A with IPv4 ethertype and subnet B with IPv6 ethertype. When Neutron Port was created in the network N1, it has 1 IPv4 address and 1 IPv6 address. If user lately will add others subnets to the Network N1 and will create the second Neutron Port, anyway the second VPN port, constructed for a new Neutron Port will keep only IP addresses from subnets (1) and (2). So valid network configuration in this case is a network with only 2 subnets: IPv4 and IPv6. See the chapter "Configuration impact" for more details. Second dualstack network N2 can be added to the same L3 BGPVPN instance.

It is valid for all schemes: in dependency of chosen ODL configuration, either ODL, or Neutron Dhcp Agent will provide IPv4 addresses for launched VMs. Please note, that currently DHCPv6 is supported only by Neutron Dhcp Agent. ODL provides only SLAAC GUA IPv6 address allocation for VMs launched in IPv6 private subnets attached to a Neutron router.

It is to be noted that today, setup 3 can not be executed for VPNv6 with the above allocation scheme previously illustrated. Indeed, only a neutron router is able to send router advertisements, which is the corner stone for DHCPv6 allocation. Either IPv6 fixed IPs will have to be used for this setup, or an extra enhancement for providing router advertisements for such a configuration will have to be done. The setup 3 will be revisited in future.

Known Limitations

Currently, from Openstack-based Opendaylight Bgpvpn driver point-of-view, there is a check, where it does not allow more than one router to be associated to a single L3 BGPVPN. This was done in Openstack, because actually entire ODL modeling and enforcement supported only one router per L3 BGPVPN by design.

From Netvirt point of view, there are some limitations as well:

- We can not associate VPN port with both IPv4 and IPv6 Neutron Port addresses at the same time. Currently, any first Neutron Port IP address is using to create a VPN interface. If a Neutron Port possesses multiple IP Addresses, regardless of ethertype, this port might not work properly with ODL.
- It is not possible to associate a single L3 BGPVPN instance with two different routers.

Use Cases

There is no change in the use cases described in [6] and [7], except that the single L3 BGPVPN instance serves both IPv4 and IPv6 subnets.

Inter DC Access

1. **two-router** solution

IPv4 subnet Subnet A is added as a port in Router 1, IPv6 subnet Subnet B is added as a port in Router 2. The same L3 BGPVPN instance will be associated with both Router 1 and Router 2.

The L3 BGPVPN instance will distinguish ethertype of router ports and will create appropriate FIB entries associated to its own VPN entry, so IPv4 and IPv6 enries will be gathered in the same L3 BGPVPN.

2. **dualstack-router** solution

IPv4 subnet Subnet A is added as a port in Router 1, IPv6 subnet Subnet B is added as a port in Router 1 as well. L3 BGPVPN instance will be associated with Router 1.

The L3 BGPVPN instance will distinguish ethertype of routers ports and will create appropriate FIB entries associated to its own VPN entry as well. Appropriate BGP VRF context for IPv4 or IPv6 subnets will be also created.

External Internet Connectivity

External Internet Connectivity is not in the scope of this specification.

Proposed changes

All changes we can split in two main parts.

1. Distinguish IPv4 and IPv6 VRF tables with the same RD/iRT/eRT

    1.1 Changes in neutronvpn

    To support a pair of IPv4 and IPv6 prefixes for each launched dualstack VM we need to obtain information about subnets, where dualstack VM was spawned and information about extraroutes, enabled for these subnets. Obtained information will be stored in vmAdj and erAdjList objects respectively. These objects are attributes of created for new dualstack VM VPN interface. Created VPN port instance will be stored as part of already existed L3 BGPVPN node instance in MDSAL DataStore.

    When we update L3 BGPVPN instance node (associate/dissociated router or network), we need to provide information about ethertype of new attached/detached subnets, hence, Neutron Ports. New argument flags **ipv4On** and **ipv6On** will be introduced for that in **NeutronvpnManager** function API, called to update current L3 BGPVPN instance (*updateVpnInstanceNode()* method). *UpdateVpnInstanceNode()* method is also called, when we create

a new L3 BGPVPN instance. So, to provide appropriate values for **ipv4On**, **ipv6On** flags we need to parse subnets list. Then in dependency of these flags values we will set either **Ipv4Family** attribute for the new L3 BGPVPN instance or **Ipv6Family** attribute, or both attributes. **Ipv4Family**, **Ipv6Family** attributes allow to create ipv4 or/and ipv6 VRF context for underlayed vpnmanager and bgpmanager APIs.

1.2. Changes in vpnmanager

When L3 BGPVPN instance is created or updated, VRF tables must be created for QBGP as well. What we want, is to introduce separate VRF tables, created according to **IPv4Family/IPv6Family** VPN attributes, i.e. we want to distinguish IPv4 and IPv6 VRF tables, because this will bring flexibility in QBGP. For example, if QBGP receives an entry IPv6 MPLSVPN on a router, which is expecting to receive only IPv4 entries, this entry will be ignored. The same for IPv4 MPLSVPN entries respectively.

So, for creating **VrfEntry** objects, we need to provide information about L3 BGPVPN instance ethertype (**Ipv4Family/Ipv6Family** attribute), route distinguishers list, route imports list and route exports lists (**RD/iRT/eRT**). **RD/iRT/eRT** lists will be simply obtained from subnetworks, attached to the chosen L3 BGPVPN. Presence of **IPv4Family**, **IPv6Family** in VPN will be translated in following VpnInstanceListener class attributes: **afiIpv4**, **afiIpv6**, **safiMplsVpn**, **safiEvpn**, which will be passed to *addVrf()* and *deleteVrf()* bgpmanager methods for creating/deleting either **IPv4 VrfEntry** or **IPv6 VrfEntry** objects.

RD/iRT/eRT lists will be the same for both **IPv4 VrfEntry** and **IPv6 VrfEntry** in case, when IPv4 and IPv6 subnetworks are attached to the same L3 BGPVPN instance.

1.3 Changes in bgpmanager

In bgpmanager we need to change signatures of *addVrf()* and *deleteVrf()* methods, which will trigger signature changes of underlying API methods *addVrf()* and *delVrf()* from *BgpConfigurationManager* class.

This allows *BgpConfigurationManager* class to create needed IPv4 VrfEntry and IPv6 VrfEntry objects with appropriate **AFI** and **SAFI** values and finally pass this appropriate **AFI** and **SAFI** values to *BgpRouter*.

BgpRouter represents client interface for thrift API and will create needed IPv4 and IPv6 VRF tables in QBGP.

1.4 Changes in yang model

To support new attributes **AFI** and **SAFI** in bgpmanager classes, it should be added in ebgp.yang model:

```
list address-families {
 key "afi safi";
  leaf afi {
    type uint32;
    mandatory "true";
  }
  leaf safi {
    type uint32;
    mandatory "true";
  }
}
```

1.5 Changes in QBGP thrift interface

To support separate IPv4 and IPv6 VRF tables in QBGP we need to change signatures of underlying methods *addvrf()* and *delvrf()* in thrift API as well. They must include the address family and subsequent address families informations:

```
enum af_afi {
    AFI_IP = 1,
    AFI_IPV6 = 2,
}

i32 addVrf(1:layer_type l_type, 2:string rd, 3:list<string>␣
↪irts, 4:list<string> erts,
           5:af_afi afi, 6:af_safi afi),
i32 delVrf(1:string rd, 2:af_afi afi, 3:af_safi safi)
```

2. Support of two routers, attached to the same L3 BGPVPN

2.1 Changes in neutronvpn

two-router solution assumes, that all methods, which are using to create, update, delete VPN interface or/and VPN instance must be adapted to a case, when we have a list of subnetworks and/or list of router IDs to attach. Due to this, appropriate changes need to be done in nvpnManager method APIs.

To support **two-router** solution properly, we also should check, that we do not try to associate to L2 BGPVPN a router, that was already associated to that VPN instance. Attached to L3 BGPVPN router list must contain maximum 2 router IDs. Routers, which IDs are in the list must be only singlestack routers. More information about supported router configurations is available below in chapter "Configuration Impact".

For each created in dualstack network Neutron Port we take only the last received IPv4 address and the last received IPv6 address. So we also limit a length of subnets list, which could be attached to a L3 BGPVPN instance, to two elements. (More detailed information about supported network configurations is available below in chapter "Configuration Impact".) Two corresponding **Subnetmap** objects will be created in *NeutronPortChangeListener* class for attached subnets. A list with created subnetmaps will be passed as argument, when *createVpnInterface* method will be called.

2.2 Changes in vpnmanager

VpnMap structure must be changed to support a list with router IDs. This change triggers modifications in all methods, which retry router ID from *VpnMap* object.

VpnInterfaceManager structure must be also changed, to support a list of VPN instance name. So all methods, which gives VPN router ID from *VpnInterfaceManager* should be modified as well.

As consequence, in operDS, a *VpnInterfaceOpDataEntry* structure is created, inherited from *VpnInterface* in configDS. While the latter structure has a list of VPN instance name, the former will be instantiated in operDS as many times as there are VPN instances. The services that were handling *VPNInterface* in operDS, will be changed to handle *VPNInterfaceOpDataEntry*. That structure will be indexed by InterfaceName and by VPNName. The services include natservice, fibmanager, vpnmanager, cloud service chain.

Also, an augment structure will be done for *VPNInterfaceOpDataEntry* to contain the list of operational adjacencies. As for *VpnInterfaceOpDataEntry*, the new *AdjacenciesOp* structure will replace Adjacencies that are in operDS. Similarly, the services will be modified for that.

Also, *VPNInterfaceOpDataEntry* will contain a *VPNInterfaceState* that stands for the state of the VPN Interface. Code change will be done to reflect the state of the interface. For instance, if VPNInstance is not ready, associated VPNInterfaceOpDataEntries will have the state changed to INACTIVE. Reversely, the state will be changed to ACTIVE.

2.3 Changes in yang model

To provide change in *VpnMap* and in *VpnInterfaceManager* structures, described above, we need to modify following yang files.

2.3.1 neutronvpn.yang

- Currently, container *vpnMap* holds one router-id for each L3 BGPVPN instance ID. A change consists in replacing one router-id leaf by a leaf-list of router-ids. Obviously, no more than two router-ids will be used.
- Container *vpnMaps* is used internally for describing a L3 BGPVPN. Change router-id leaf by router-ids leaf-list in this container is also necessary.

```
--- a/vpnservice/neutronvpn/neutronvpn-api/src/main/yang/
↪neutronvpn.yang
+++ b/vpnservice/neutronvpn/neutronvpn-api/src/main/yang/
↪neutronvpn.yang
@@ -1,4 +1,3 @@
-
module neutronvpn {

namespace "urn:opendaylight:netvirt:neutronvpn";
@@ -120,7 +119,7 @@ module neutronvpn {
Format is ASN:nn or IP-address:nn.";
}

-         leaf router-id {
+         leaf-list router-ids {
          type    yang:uuid;
          description "UUID router list";
      }
@@ -173,7 +172,7 @@ module neutronvpn {
description "The UUID of the tenant that will own the subnet.";
}

-             leaf router-id {
+             leaf-list router_ids {
              type    yang:uuid;
              description "UUID router list";
          }
```

2.3.2 l3vpn.yang

- Currently, list vpn-interface holds a leaf vpn-instance-name, which is a container for VPN router ID. A change consists in replacing leaf vpn-instance-name by a leaf-list of VPN router IDs, because L3 BGPVPN instance can be associated with two routers. Obviously, no more than two VPN router-IDs will be stored in leaf-list vpn-instance-name.

```
--- a/vpnservice/vpnmanager/vpnmanager-api/src/main/yang/
↪l3vpn.yang
+++ b/vpnservice/vpnmanager/vpnmanager-api/src/main/yang/
↪l3vpn.yang
        @@ -795,21 +795,21 @@

      list vpn-interface  {
        key "name";
        max-elements "unbounded";
        min-elements "0";
        leaf name {
```

```
          type leafref {
            path "/if:interfaces/if:interface/if:name";
          }
        }
-       leaf vpn-instance-name {
+       leaf-list vpn-instance-name {
            type string {
                length "1..40";
            }
        }
        leaf dpn-id {
            type uint64;
        }
        leaf scheduled-for-remove {
            type boolean;
        }
```

2.3.3 odl-l3vpn.yang

```
augment "/odl-l3vpn:vpn-interface-op-data/odl-l3vpn:vpn-
↪interface-op-data-entry" {
   ext:augment-identifier "adjacencies-op";
   uses adjacency-list;
}

container vpn-interface-op-data {
   config false;
   list vpn-interface-op-data-entry {
      key "name vpn-instance-name";
      leaf name {
         type leafref {
           path "/if:interfaces/if:interface/if:name";
         }
       }
       leaf vpn-instance-name {
         type string {
           length "1..40";
         }
       }
       max-elements "unbounded";
       min-elements "0";
       leaf dpn-id {
         type uint64;
       }
       leaf scheduled-for-remove {
         type boolean;
       }
       leaf router-interface {
           type boolean;
       }
       leaf vpn-interface-state {
         description
          "This flag indicates the state of this interface␣
↪in the VPN identified by vpn-name.
           ACTIVE state indicates that this vpn-interface␣
↪is currently associated to vpn-name
           available as one of the keys.
           INACTIVE state indicates that this vpn-
↪interface has already been dis-associated
```

```
            from vpn-name available as one of the keys.";

            type enumeration {
             enum active {
                value "0";
                description
                "Active state";
             }
             enum inactive {
                value "1";
                description
                "Inactive state";
             }
            }
            default "active";
        }
    }
}
```

Pipeline changes

There is no change in the pipeline, regarding the changes already done in [6] and [7].

Traffic from DC-Gateway to Local DPN (SYMMETRIC IRB)

The DC-GW has the information, that permits to detect an underlay destination IP and MPLS label for a packet coming from the Internet or from anotherr DC-GW.

Classifier Table (0) =>
LFIB Table (20) `match: tun-id=mpls_label set vpn-id=l3vpn-id, pop_mpls label, set output to nexthopgroup-dst-vm` =>
NextHopGroup-dst-vm: `set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

Traffic from Local DPN to DC-Gateway (SYMMETRIC IRB)

Classifier Table (0) =>
Lport Dispatcher Table (17) `match: LportTag l3vpn service: set vpn-id=l3vpn-id` =>
DMAC Service Filter (19) `match: dst-mac=router-internal-interface-mac l3vpn service: set vpn-id=l3vpn-id` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=ext-ipv4-address set tun-id=mpls_label output to MPLSoGRE tunnel port` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=ext-ipv6-address set tun-id=mpls_label output to MPLSoGRE tunnel port` =>

Please, note that `router-internal-interface-mac` stands for MAC address of the internal subnet gateway router port.

Configuration impact

1. Limitations for router configurations

   1.1 Maximum number of singlestack routers that can be associated to a L3BGPVPN is limited to 2. Maximum number of dualstack routers that can be associated with a BGPVPN is limited to 1.

   1.2 If a L3 BGPVPN has already associated with a one singlestack router and we try to associate this VPN instance again with a dualstack router, exception will not be raised. But this configuration will not be valid.

   1.3 If a singlestack router is already associated to a L3 BGPVPN instance, and it has more than one port and we try to add a port to this router with another ethertype, i.e. we try to make this router dualstack, exception will not be raised. But this configuration will not be valid and supported.

   1.4 When a different ethertype port is added to a singlestack router, which already has only one port and which is already associated to a L3 BGPVPN instance, singlestack router in this case becomes dualstack router with only two ports. This router configuration is allowed by current specification.

2. Limitations for subnetworks configurations

   2.1 Maximum numbers of different ethertype subnetworks associated to a one L3 BGPVPN instance is limited to two. If a network contains more than two different ethertype subnetworks, exception won't be raised, but this configuration isn't supported.

   2.2 When we associate a network with a L3 BGPVPN instance, we do not care if subnetworks from this network are ports in some routers and these routers were associated with other VPNs. This configuration is not considered as supported as well.

3. Limitations for number of IP addresses for a Neutron Port

The specification only targets dual-stack networks, that is to say with 1 IPv4 address and one IPv6 address only. For other cases, that is to say, adding subnetworks IPv4 or IPv6, will lead to undefined or untested use cases. The multiple subnets test case would be handled in a future spec.

ECMP impact

ECMP - Equal Cost multiple path.

ECMP feature is currently provided for Neutron BGPVPN networks and described in the specification [10]. 3 cases have been cornered to use ECMP feature for BGPVPN usability.

- ECMP of traffic from DC-GW to OVS (inter-DC case)
- ECMP of traffic from OVS to DC-GW (inter-DC case)
- ECMP of traffic from OVS to OVS (intra-DC case)

In each case, traffic begins either at DC-GW or OVS node. Then it is sprayed to end either at OVS node or DC-GW.

ECMP feature for Neutron BGPVPN networks was successfully (OK) tested with IPv4 L3 BGPVPN and IPv6 L3 BGPVPN (OK). the dual stack VM connectivity should embrace ECMP

We've included this chapter to remind, that code changes for supporting dualstack VMs should be tested against ECMP scenario as well.

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

None

Targeted Release

Carbon

Alternatives

None

Usage

Assume, that in the same provider network we have OpenStack installed with 1 controller and 2 compute nodes, DC-GW node and OpenDaylight node.

- create private tenant networks and subnetworks
  - create Network N;
  - declare Subnet A IPv4 for Network N;
  - declare Subnet B IPv6 for Network N;
  - create two ports in Network N;
  - each port will inherit a dual IP configuration.
- create routers
  - **two-router** solution + create two routers A and B, each router will be respectively connected to

    IPv4 and IPv6 subnets;

    * add subnet A as a port to router A;
    * add subnet B as a port to router B.
  - **dualstack-router** solution + create router A; + add subnet A as a port to router A; + add subnet B as a port to router A.

- Create MPLSoGRE tunnel between DPN and DCGW

```
POST /restconf/operations/itm-rpc:add-external-tunnel-endpoint
{
  "itm-rpc:input": {
    "itm-rpc:destination-ip": "dcgw_ip",
    "itm-rpc:tunnel-type": "odl-interface:tunnel-type-mpls-over-gre"
  }
}
```

- create the DC-GW VPN settings
  - Create a L3 BGPVPN context. This context will have the same settings as in [7].In dualstack case both IPv4 and IPv6 prefixes will be injected in the same L3 BGPVPN.
- create the ODL L3 BGPVPN settings
  - Create a BGP context. This step permits to start QBGP module depicted in [8] and [9]. ODL has an API, that permits interfacing with that external software. The BGP creation context handles the following:
    - start of BGP protocol;
    - declaration of remote BGP neighbor with the AFI/SAFI affinities. In our case, VPNv4 and VPNv6 address families will be used.
  - Create a L3 BGPVPN, this L3 BGPVPN will have a name and will contain VRF settings.
- associate created L3 BGPVPN to router
  - **two-router** solution: associate routers A and B with a created L3 BGPVPN;
  - **dualstack-router** solution: associate router A with a created L3 BGPVPN.
- Spawn a VM in a created tenant network:

  The VM will possess IPv4 and IPv6 addresses from subnets A and B.
- Observation: dump ODL BGP FIB entries

  At ODL node, we can dump ODL BGP FIB entries and we should see entries for both IPv4 and IPv6 subnets prefixes:

```
GET /restconf/config/odl-fib:fibEntries
{
  "fibEntries": {
    "vrfTables": [
      {
        "routeDistinguisher": <rd-uuid>
      },
      {
        "routeDistinguisher": <rd>,
        "vrfEntry": [
          {
            "destPrefix": <IPv6_VM1/128>,
            "label": <label>,
            "nextHopAddressList": [
              <DPN_IPv4>
            ],
            "origin": "l"
          },
        ]
      }
    ]
```

```
  }
}
```

Features to Install

odl-netvirt-openstack

REST API

CLI

A new option `--afi` and `--safi` will be added to command `odl:bgp-vrf`:

```
odl:bgp-vrf --rd <> --import-rt <> --export-rt <> --afi <1|2> --safi <value> add|del
```

Implementation

Assignee(s)

Primary assignee: Philippe Guibert <philippe.guibert@6wind.com>

Other contributors:

- Valentina Krasnobaeva <valentina.krasnobaeva@6wind.com>
- Noel de Prandieres <prandieres@6wind.com>

Work Items

- QBGP Changes
- BGPManager changes
- VPNManager changes
- NeutronVpn changes

Dependencies

Quagga from 6WIND is available at the following urls:

- https://github.com/6WIND/quagga
- https://github.com/6WIND/zrpcd

Testing

Unit Tests

Some L3 BGPVPN testing may have be done. Complementary specification for other tests will be done.

Integration Tests

TBD

CSIT

Basically, IPv4 and IPv6 vpnservice functionality have to be validated by regression tests with a single BGPVRF.

CSIT specific testing will be done to check dualstack VMs connectivity with network configurations for **two-router** and **dualstack-router** solutions.

Two-router solution test suite:

1. Create 2 Neutron Networks NET_1_2RT and NET_2_2RT.

   1.1 Query ODL restconf API to check that both Neutron Network objects were successfully created in ODL.

   1.2 Update NET_1_2RT with a new description attribute.

2. In each Neutron Network create one Subnet IPv4 and one Subnet IPv6: SUBNET_V4_1_2RT, SUBNET_V6_1_2RT, SUBNET_V4_2_2RT, SUBNET_V6_2_2RT, respectively.

   2.1 Query ODL restconf API to check that all Subnetwork objects were successfully created in ODL.

   2.2 Update SUBNET_V4_2RT, SUBNET_V6_2RT with a new description attribute.

3. Create 2 Routers: ROUTER_1 and ROUTER_2.

   3.1 Query ODL restconf API to check that all Router objects were successfully created in ODL.

4. Add SUBNET_V4_1_2RT, SUBNET_V4_2_2RT to ROUTER_1 and SUBNET_V6_1_2RT, SUBNET_V6_2_2RT to ROUTER_2.

5. Create 2 security-groups: SG6_2RT and SG4_2RT. Add appropriate rules to allow IPv6 and IPv4 traffic from/to created subnets, respectively.

6. In network NET_1_2RT create Neutron Ports: PORT_11_2RT, PORT_12_2RT, attached with security groups SG6_2RT and SG4_2RT; in network NET_2_2RT: PORT_21_2RT, PORT_22_2RT, attached with security groups SG6_2RT and SG4_2RT.

   6.1 Query ODL restconf API to check, that all Neutron Port objects were successfully created in ODL.

   6.2 Update Name attribute of PORT_11_2RT.

7. Use each created Neutron Port to launch a VM with it, so we should have 4 VM instances: VM_11_2RT, VM_12_2RT, VM_21_2RT, VM_22_2RT.

   7.1 Connect to NET_1_2RT and NET_2_2RT dhcp-namespaces, check that subnet routes were successfully propagated.

   7.2 Check that all VMs have: one IPv4 address and one IPv6 addresses.

8. Check IPv4 and IPv6 VMs connectivity within NET_1_2RT and NET_2_2RT.

9. Check IPv4 and IPv6 VMs connectivity across NET_1_2RT and NET_2_2RT with ROUTER_1 and ROUTER_2.

   9.1 Check that FIB entries were created for spawned Neutron Ports.

   9.2 Check that all needed tables (19, 17, 81, 21) are presented in OVS pipelines and VMs IPs, gateways MAC and IP addresses are taken in account.

10. Connect to VM_11_2RT and VM_21_2RT and add extraroutes to other IPv4 and IPv6 subnets.

10.1 Check other IPv4 and IPv6 subnets reachability from VM_11_2RT and VM_21_2RT.

11. Delete created extraroutes.
12. Delete and recreate extraroutes and check its reachability again.
13. Create L3VPN and check with ODL REST API, that it was successfully created.
14. Associate ROUTER_1 and ROUTER_2 with created L3VPN and check the presence of router IDs in VPN instance with ODL REST API.
15. Check IPv4 and IPv6 connectivity accross NET_1_2RT and NET_2_2RT with associated to L3VPN routers.

    15.1 Check with ODL REST API, that VMs IP addresses are presented in VPN interfaces entries.

    15.2 Verify OVS pipelines at compute nodes.

    15.3 Check the presence of VMs IP addresses in vrfTables objects with ODL REST API query.
16. Dissociate L3VPN from ROUTER_1 and ROUTER_2.
17. Delete ROUTER_1 and ROUTER_2 and its interfaces from L3VPN.
18. Try to delete router with NonExistentRouter name.
19. Associate L3VPN to NET_1_2RT.
20. Dissociate L3VPN from NET_1_2RT.
21. Delete L3VPN.
22. Create multiple L3VPN.
23. Delete multiple L3VPN.

Documentation Impact

Necessary documentation would be added if needed.

References

[1] OpenDaylight Documentation Guide

[2] https://specs.openstack.org/openstack/nova-specs/specs/kilo/template.html

[3] http://docs.openstack.org/developer/networking-bgpvpn/overview.html

[4] Spec to support IPv6 North-South support for Flat/VLAN Provider Network.

[5] BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN

[6] Spec to support IPv6 DC to Internet L3VPN connectivity using BGPVPN

[7] Spec to support IPv6 Inter DC L3VPN connectivity using BGPVPN

[8] Zebra Remote Procedure Call

[9] Quagga BGP protocol

Listener Dependency Helper

https://git.opendaylight.org/gerrit/#/q/topic:ListenerDepedencyHelper

Listener Dependency Helper makes “Data Store Listeners” independent from dependency resolution.

Problem description

When a DataStore-Listener is fired with config add/update/delete event, as part of listener processing it may try to read the other data store objects, at times those datastore objects are not yet populated. In this scenario, listener event processing has to be delayed (or) discarded, as the required information is NOT entirely available. Later when the dependant data objects are available, this listener event will not be triggered again by DataStore.

This results in some events not getting processed resulting in possible data-path, bgp control and data plane failures.

Example: VpnInterface add() callback triggered by MD-SAL on vpnInterface add. While processing add() callback, the corresponding vpnInstance is expected to be present in MD-SAL operational DS; which means that vpnInstance creation is complete (updating the vpn-targets in Operational DS and BGP).

Information: vpnInstance Config-DS listener thread has to process vpnInstance creation and update vpnInstance in operational DS. vpnInstance creation listener callback is handled by different listener thread.

Use Cases

Use Case 1: VPNInterfaces may get triggered before VPNInstance Creation.

Current implementation: Delay based waits for handling VPNInterfaces that may get triggered before VPNInstance Creation(waitForVpnInstance()).

Use Case 2: VPNManager to handle successful deletion of VPN which has a large number of BGP Routes (internal/external):

Current implementation: Delay-based logic on VPNInstance delete in VPNManager (waitForOpRemoval()).

Use Case 3: VpnSubnetRouteHandler that may get triggered before VPNInstance Creation.

Current implementation: Delay based waits in VpnSubnetRouteHandler which may get triggered before VPNInstance Creation(waitForVpnInstance()).

Use Case 4: VPN Swaps (Internal to External and vice-versa)

Current implementation: Currently we support max of 100 VM's for swap (VpnInterfaceUpdateTimerTask, waitForFibToRemoveVpnPrefix()).

Proposed change

During Listener event call-back (AsyncDataTreeChangeListenerBase) from DataStore, check for pending events in "Listener-Dependent-Queue" with same InstanceIdentifier to avoid re-ordering.

Generic Queue Event Format:

key : Instance Identifier eventType : Type of event (ADD/UPDATE/DELETE) oldData : Data before modification (for Update event); newData : Newly populated data queuedTime : at which the event is queued to LDH. lastProcessedTime : latest time at which dependency list verified expiryTime : beyond which processing for event is useless waitBetweenDependencyCheckTime : wait time between each dependency check dependentIIDs : list of dependent InstanceIdentifiers retryCount : max retries allowed. databroker : data broker. deferTimerBased : flag to choose between (timer/listener based).

For Use Case - 1: deferTimerBased shall be set to TRUE (as per the specification).

During processing of events (either directly from DataStore or from "Listener-Dependent-Queue"), if there any dependent objects are yet to populated; queue them to "Listener-Dependent-Queue".

Expectations from Listener: Listener will push the callable instance to "Listener-Dependent-Queue" if it cannot proceed with processing of the event due to dependent objects/InstanceIdentifier and list of dependent IID's.

There are two approaches the Listener Dependency check can be verified.

approach-1 Get the list of dependent-IID's, query DataStore/Cache for

depenedency resolution at regular intervals using "timer-task-pool". Once all the dependent IID's are resolved, call respective listener for processing.

LDH-task-pool : pool of threads which query for dependency resolution READ ONLY operation in DataStore. These threads are part of LDH common for all listeners.

hasDependencyResolved(<InstanceIdentifier iid, Boolean shouldDataExist, DataStoreType DSType> List), this shall return either Null list (or) the list which has dependencies yet to be resolved. In case Listener has local-cache implemented for set of dependencies, it can look at cache and identify. This api will be called from LDH-task-pool of thread(s).

instanceIdentifier is the MD-SAL key value which need to be verified for existence/non-existence of data. Boolean shouldDataExist: shall be TRUE, if the Listener expects to have the information exists in MD-SAL; False otherwise.

approach-2 Register Listener for wild-card path of IID's.

When a Listener gets queued to ""Listener-Dependent-Queue", LDH shall register itself as Listener for the dependent IID's (using wild-card-path/parent-node). Once the listener gets fired, identify the dependent listeners waiting for the Data. Once the dependent Listener is identified, if the dependent-IID list is NULL. Trigger listener for processing the event. LDH-task-pool shall unregister itself from wild-card-path/parent-node once there are no dependent listeners on child-nodes.

Re-Ordering

The following scenario, when re-ordering can happen and avoidance of the same:

Example: Key1 and Value1 are present in MD-SAL Data Store under Tree1, SubTree1 (for say). Update-Listener for Key1 is dependent on Dependency1.

Key1 received UPDATE event (UPDATE-1) with value=x, at the time of processing UPDATE-1, dependency is not available. So Listener Queued 'UPDATE-1' event to "UnProcessed-EventQueue". same key1 received UPDATE event (UPDATE-2) with value=y, at the time of processing UPDATE-2, dependency is available (Dependency1 is resolved), so it goes and processes the event and updates value of Key1 to y.

After WaitTime, event Key1, UPDATE-1 is de-queued from "UnProcessed-EventQueue" and put for processing in Lister. Listener processes it and updates the Key1 value to x. (which is incorrect, happened due to re-ordering of events).

To avoid reordering of events within listener, every listener call back shall peek into "UnProcessed-EventQueue" to identify if there exists a pending event with same key value; if so, either suppress (or) queue the event. Below are event ordering expected from MD-SAL and respective actions:

what to consider before processing the event to avoid re-ordering of events:

Current Event| Queued Event| Action
ADD | ADD | NOT EXPECTED
ADD | REMOVE | QUEUE THE EVENT
ADD | UPDATE | NOT EXPECTED
UPDATE | ADD | QUEUE EVENT
UPDATE | UPDATE | QUEUE EVENT
UPDATE | REMOVE | NOT EXPECTED
REMOVE | ADD | SUPPRESS BOTH
REMOVE | UPDATE | EXECUTE REMOVE SUPPRESS UPDATE
REMOVE | REMOVE | NOT EXPECTED

Pipeline changes

none

Yang changes

none

Configuration impact

none

Clustering considerations

In the two approaches mentioned: 1 - Timer: polling MD-SAL for dependency resolution may incur in more number of reads.

2 - RegisterListener: RegisterListener may some impact at the time of registering listener after which a notification message to cluser nodes.

Predined List of Listeners

perational/odl-l3vpn:vpn-instance-op-data/vpn-instance-op-data-entry/* operational/odl-l3vpn:vpn-instance-op-data/vpn-instance-op-data-entry/

vpn-id/vpn-to-dpn-list/*

config/l3vpn:vpn-instances/*

Other Infra considerations

Security considerations

none

Scale and Performance Impact

this infra, shall improve scaling of application without having to wait for dependent data store gets populated. Performance shall remain intact.

Targeted Release

Alternatives

- use polling/wait mechanisms

Features to Install

REST API

CLI

CLI will be added for debugging purpose.

Implementation

Assignee(s)

Primary assignee: Siva Kumar Perumalla (sivakumar.perumalla@ericsson.com)

Other contributors: Suneelu Verma K.

Work Items

Dependencies

Testing

Unit Tests

Integration Tests

CSIT

Documentation Impact

References

Acronyms

IID: InstanceIdentifier

Table of Contents

New SFC Classifier

https://git.opendaylight.org/gerrit/#/q/topic:new-sfc-classifier

The current SFC Netvirt classifier only exists in the old Netvirt. This blueprint explains how to migrate the old Netvirt classifier to a new Netvirt classifier.

Terminology

- NSH - Network Service Headers, used as Service Chaining encapsulation. NSH RFC Draft [1]
- NSI - Network Service Index, a field in the NSH header used to indicate the next hop
- NSP - Network Service Path, a field in the NSH header used to indicate the service chain
- RSP - Rendered Service Path, a service chain.

- SFC - Service Function Chaining. SFC RFC [2] ODL SFC Wiki [3].
- SF - Service Function
- SFF - Service Function Forwarder
- VXGPE - VXLAN GPE (Generic Protocol Encapsulation) Used as transport for NSH. VXGPE uses the same header format as traditional VXLAN, but adds a Next Protocol field to indicate NSH will be the next header. Traditional VXLAN implicitly expects the next header to be ethernet. VXGPE RFC Draft [4].

Problem description

In the Boron release, an SFC classifier was implemented, but in the old Netvirt. This blueprint intends to explain how to migrate the old Netvirt classifier to a new Netvirt classifier, which includes integrating the classifier and SFC with Genius.

The classifier is an integral part of Service Function Chaining (SFC). The classifier maps client/tenant traffic to a service chain by matching the packets using an ACL, and once matched, the classifier encapsulates the packets using some sort of Service Chaining encapsulation. Currently, the only supported Service Chaining encapsulation is NSH using VXGPE as the transport. Very soon (possibly in the Carbon release) Vxlan will be added as another encapsulation/transport, in which case NSH is not used. The transport and encapsulation information to be used for the service chain is obtained by querying the Rendered Service Path (RSP) specified in the ACL action.

The transport and encapsulation used between the classifier and the SFF, and also between SFFs will be VXGPE+NSH. The transport and encapsulation used between the SFF and the SF will be Ethernet+NSH.

The following image details the packet headers used for Service Chaining encapsulation with VXGPE+NSH.

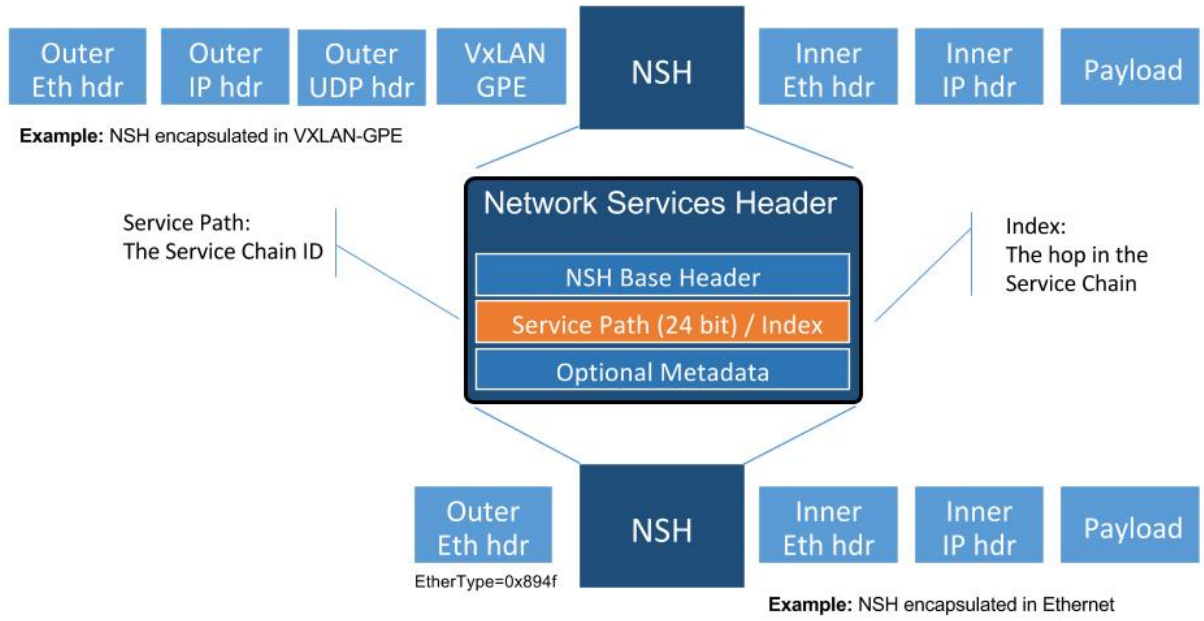


Diagram source [5].

The problem was originally discussed using the slides in this link [12] as a guideline. These slides are only intended for reference, and are not to be used for implementation.

Use Cases

The main use case addressed by adding an SFC classifier to Netvirt is to integrate SFC with Netvirt, thus allowing for Service Chaining to be used in an OpenStack virtual deployment, such as the OPNFV SFC project [6].

SFC works with both OVS and VPP virtual switches, and its even possible to have a hybrid setup whereby Netvirt is hosted on OVS and SFC is hosted on VPP switches. This blueprint only addresses the use of SFC with NetVirt and OVS.

As mentioned previously, currently SFC works with VXGPE+NSH and Eth+NSH transport/encapsulation, and soon SFC will work with VXLAN as the transport and encapsulation. The first version of this implementation will focus on VXGPE+NSH and Eth+NSH. In the future, when VXLAN is implemented in SFC, VXLAN can be added to the Netvirt SFC classifier. Changes in the transport and encapsulation used for service chains will have no affect on the Netvirt ACL model, since the transport and encapsulation information is obtained via the RSP specified in the RSP.

Proposed change

The existing old Netvirt SFC code can be found here:

- netvirt/openstack/net-virt-sfc/{api,impl}

Once the new Netvirt SFC classifier is implemented and working, the old Netvirt SFC classifier code will be left in place for at least one release cycle.

The new Netvirt SFC code base will be located here:

- netvirt/vpnservice/sfc/classifier/{api,impl}

The new Netvirt SFC classifier implementation will be new code. This implementation is not to be confused with the existing Netvirt aclservice, which is implemented for Security Groups. More details about the Genius integration can be found in the following section, but the Netvirt SFC classifier will be in a new Genius classifier service. The SFC implementation is already integrated with Genius and is managed via the Genius SFC service.

Integration with Genius

Genius [7], [8] is an OpenDaylight project that provides generic infrastructure services to other OpenDaylight projects. New Netvirt makes use of Genius and the new Netvirt classifier will also make use of Genius services. Among these services, the interface manager, tunnel manager and service binding services are of special relevance for the new Netvirt classifier.

Genius interface manager handles an overlay of logical interfaces on top of the data plane physical ports. Based on these logical interfaces, different services/applications may be bound to them with certain priority ensuring that there is no interference between them. Avoiding interference between services/applications is called Application Coexistence in Genius terminology. Typically, the effect of an application binding to a logical interface is that downstream traffic from that interface will be handed off to that application pipeline. Each application is then responsible to either perform a termination action with the packet (i.e output or drop action) or to return the packet back to Genius so that another application can handle the packet. There is a predefined set of types of services that can bind, and Classifier is one of them.

For OpenStack environments, Netvirt registers Neutron ports as logical interfaces in the Genius interface manager. Classifying traffic for a client/tenant ultimately relies on classifying traffic downstream from their corresponding Neutron ports. As such, the Netvirt classifier will bind on these interfaces as a newly defined Genius Classifier service through the Genius interface manager. It was considered integrating the Netvirt classifier with the existing Netvirt security groups, but the idea was discarded due to the possible conflicts and other complications this could cause.

Netvirt also keeps track of the physical location of these Neutron ports in the data plane and updates the corresponding Genius logical interface with this information. Services integrated with Genius may consume this information to be

aware of the physical location of a logical interface in the data plane and it's changes when a VM migrates from one location to another. New Netvirt classifier will install the classification rules based on the data plane location of the client/tenant Neutron ports whose traffic is to be classified. On VM migration, the classifier has to remove or modify the corresponding classification rules accounting for this location change, which can be a physical node change or a physical port change.

The classifier is responsible for forwarding packets to the first service function forwarder (SFF) in the chain. This SFF may or may not be on the same compute host as the classifier. If the classifier and SFF are located on the same compute host, then the encapsulated packet is sent to the SFF via the Genius Dispatcher and OpenFlow pipelines. The packets can be forwarded to the SFF locally via the ingress or egress classifier, and it will most likely be performed by the egress classifier, but this decision will be determined at implementation time.

In scenarios where the first SFF is on a different compute host than the client node, the encapsulated packet needs to be forwarded to that SFF through a tunnel port. Tunnels are handled by the Genius tunnel manager (ITM) with an entity called transport zone: all nodes in a transport zone will be connected through a tunnel mesh. Thus the netvirt classifier needs to ensure that the classifier and the SFF are included in a transport zone. The transport type is also specified at the transport zone level and for NSH it needs to be VXGPE. The classifier needs to make sure that this transport zone is handled for location changes of client VMs. Likewise, SFC needs to make sure the transport zone is handled for SF location changes.

The afore-mentioned Genius ITM is different than the tunnels currently used by Netvirt. SFC uses VXGPE tunnels, and requests they be created via the Genius ITM.

Classifier and SFC Genius Services

There will be 2 new Genius services created in Netvirt for the new Netvirt SFC classifier, namely an "Ingress SFC Classifier" and an "Egress SFC Classifier". There will also be a Genius service for the SFC SFF functionality that has already been created in the SFC project.

The priorites of the services will be as follows:

Ingress Dispatcher:

- SFC - P1
- IngressACL - P2
- Ingress SFC Classifier - P3
- IPv6, IPv4, L2 - P4...

Egress Dispatcher:

- EgressACL - P1
- Egress SFC Classifier - P2

The Ingress SFC classifier will bind on all the Neutron VM ports of the Neutron Network configured in the ACL. All packets received from these Neutron ports will be sent to the Ingress SFC classifier via the Genius Ingress Dispatcher, and will be subjected to ACL matching. If there is no match, then the packets will be returned to the Genius dispatcher so they can be sent down the rest of the Netvirt pipeline. If there is an ACL match, then the classifier will encapsulate NSH, set the NSP and NSI accordingly, initialize C1 and C2 to 0, and send the packet down the rest of the pipeline. Since the SFC service (SFF) will most likely not be bound to this same Neutron port, the packet wont be processed by the SFF on the ingress pipeline. If the classifier and first SFF are in the same node, when the packet is processed by the egress SFC classifier, it will be resubmitted back to the Ingress SFC service (SFC SFF) for SFC processing. If not, the packet will be sent to the first SFF.

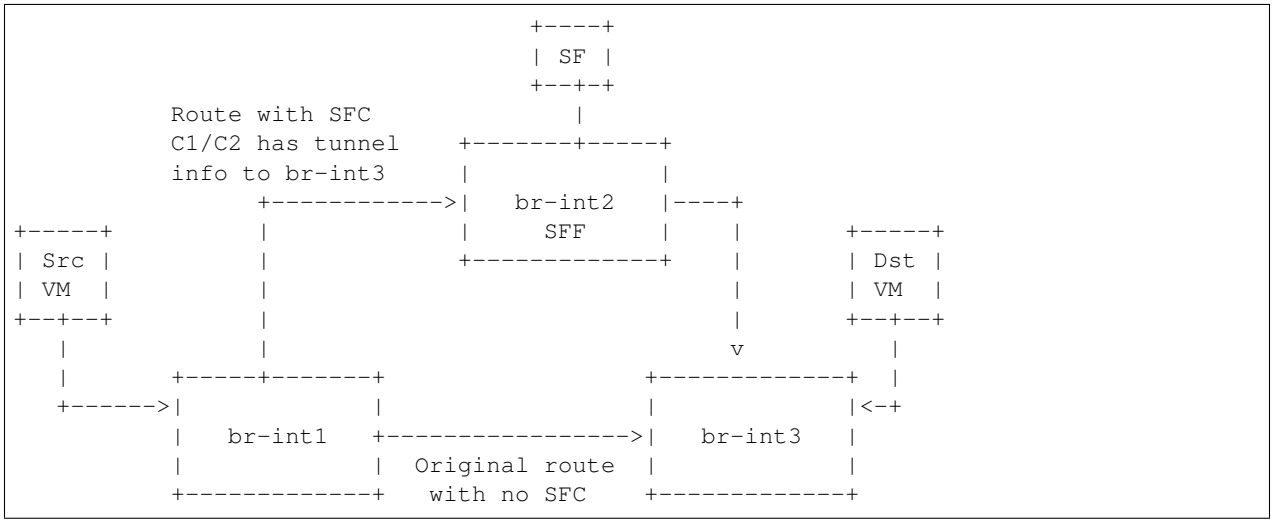
The Ingress SFC service (SFF) will bind on the Neutron ports for the Service Functions and on the VXGPE ports. The Ingress SFC service will receive packets from these Neutron and VXGPE ports, and also those that have been resubmitted from the Egress SFC Classifier. It may be possible that packets received from the SFs are not NSH

encapsulated, so any packets received by the Ingress SFC service that are not NSH encapsulated will not be processed and will be sent back to the Ingress Dispatcher. For the NSH packets that are received, the Ingress SFC service will calculate the Next-Hop and modify either the VXGPE header if the next hop is a different SFF, or modify the Ethernet encapsulation header if the next hop is an SF on this same SFF. Once NSH packets are processed by the Ingress SFC service, they will be sent to the Egress Dispatcher.

The Egress SFC classifier service is the final phase of what the Ingress SFC classifier service started when an ACL match happens. The packet needed to go down the rest of the pipeline so the original packet destination can be calculated. The Egress SFC classifier will take the information prepared by the rest of the Netvirt pipeline and store the TunIPv4Dst and VNID of the destination compute host in C1 and C2 respectively. If the packet is not NSH encapsulated, then it will be sent back to the Egress Dispatcher. If the packet does have NSH encapsulation, then if C1/C2 is 0, then the fields will be populated as explained above. If the C1/C2 fields are already set, the packet will be sent out to either the Next Hop SF or SFF.

At the last hop SFF, when the packet egresses the Service Chain, the SFF will pop the NSH encapsulation and use the NSH C1 and C2 fields to tunnel the packet to its destination compute host. If the destination compute host is the same as the last hop SFF, then the packet will be sent down the rest of the Netvirt pipeline so it can be sent to its destination VM on this compute host. When the destination is local, then the inport will probably have to be adjusted.

An example of how the last hop SFF routing works, imagine the following diagram where packet from the Src VM would go from br-int1 to br-int3 to reach the Dst VM when there is no service chaining employed. When the packets from the Src VM are subjected to service chaining, the pipeline in br-int1 need to calculate the the final destination is br-int3, and the appropriate information needs to be set in the NSH C1/C2 fields. Then the SFC SFF on br-int2, upon chain egress will use C1/C2 to send the packets to br-int3 so they can ultimately reach the Dst VM.



Pipeline changes

The existing Netvirt pipeline will not change as a result of adding the new classifier, other than the fact that the Ingress SFC classifier and Egress SFC classifier Genius Services will be added, which will change the Genius Service priorities as explained previously. The Genius pipelines can be found here [10].

Ingress Classifier Flows:

The following flows are an approximation of what the Ingress Classifier service pipeline will look like. Notice there are 2 tables defined as follows:

- **table 100: Ingress Classifier Filter table.**
  - Only allows Non-NSH packets to proceed in the classifier

- **table 101: Ingress Classifier ACL table.**
  - Performs the ACL classification, and sends packets to Ingress Dispatcher

The final table numbers may change depending on how they are assigned by Genius.

```
  // Pkt has NSH, send back to Ingress Dispatcher
cookie=0xf005ball00000101 table=100, n_packets=11, n_bytes=918,
    priority=550,nsp=42 actions=resubmit(,GENIUS_INGRESS_DISPATCHER_TABLE)

  // Pkt does NOT have NSH, send to GENIUS_INGRESS_DISPATCHER_TABLE
cookie=0xf005ball00000102 table=100, n_packets=11, n_bytes=918,
    priority=5 actions=goto_table:GENIUS_INGRESS_DISPATCHER_TABLE

  // ACL match: if TCP port=80
  // Action: encapsulate NSH and set NSH NSP, NSI, C1, C2, first SFF
  // IP in Reg0, and send back to Ingress Dispatcher to be sent down
  // the Netvirt pipeline. The in_port in the match is derived from
  // the Neutron Network specified in the ACL match and identifies
  // the tenant/Neutron Network the packet originates from
cookie=0xf005ball00000103, table=101, n_packets=11, n_bytes=918,
    tcp,tp_dst=80, in_port=10
    actions=push_nsh,
        load:0x1->NXM_NX_NSH_MDTYPE[],
        load:0x0->NXM_NX_NSH_C1[],
        load:0x0->NXM_NX_NSH_C2[],
        load:0x2a->NXM_NX_NSP[0..23],
        load:0xff->NXM_NX_NSI[],
        load:0x0a00010b->NXM_NX_REG0[],
        resubmit(,GENIUS_INGRESS_DISPATCHER_TABLE)
```

Egress Classifier Flows:

The following flows are an approximation of what the Egress Classifier service pipeline will look like. Notice there are 3 tables defined as follows:

- **table 221: Egress Classifier Filter table.**
  - Only allows NSH packets to proceed in the egress classifier
- **table 222: Egress Classifier NextHop table.**
  - Set C1/C2 accordingly
- **table 223: Egress Classifier TransportEgress table.**
  - Final egress processing and egress packets
  - Determines if the packet should go to a local or remote SFF

The final table numbers may change depending on how they are assigned by Genius.

```
  // If pkt has NSH, goto table 222 for more processing
cookie=0x14 table=221, n_packets=11, n_bytes=918,
    priority=260,md_type=1
    actions=goto_table:222

  // Pkt does not have NSH, send back to Egress Dispatcher
cookie=0x14 table=110, n_packets=0, n_bytes=0,
    priority=250
    actions=resubmit(,GENIUS_EGRESS_DISPATCHER_TABLE)
```

```
  // Pkt has NSH, if NSH C1/C2 = 0, Set C1/C2 and overwrite TunIpv4Dst
  // with SFF IP (Reg0) and send to table 223 for egress
cookie=0x14 table=222, n_packets=11, n_bytes=918,
    priority=260,nshc1=0,nshc2=0
    actions=load:NXM_NX_TUN_IPV4_DST[]->NXM_NX_NSH_C1[],
            load:NXM_NX_TUN_ID[]->NXM_NX_NSH_C2[],
            load:NXM_NX_REG0[]->NXM_NX_TUN_IPV4_DST[]
            goto_table:223

  // Pkt has NSH, but NSH C1/C2 aleady set,
  // send to table 223 for egress
cookie=0x14 table=222, n_packets=11, n_bytes=918,
    priority=250
    actions=goto_table:223


  // Checks if the first SFF (IP stored in reg0) is on this node,
  // if so resubmit to SFC SFF service
cookie=0x14 table=223, n_packets=0, n_bytes=0,
    priority=260,nsp=42,reg0=0x0a00010b
    actions=resubmit(, SFF_TRANSPORT_INGRESS_TABLE)

cookie=0x14 table=223, n_packets=0, n_bytes=0,
    priority=250,nsp=42
    actions=outport:6
```

Ingress SFC Service (SFF) Flows:

The following flows are an approximation of what the Ingress SFC service (SFF) pipeline will look like. Notice there are 3 tables defined as follows:

- **table 83: SFF TransportIngress table.**
  - Only allows NSH packets to proceed into the SFF
- tables 84 and 85 are not used for NSH
- **table 86: SFF NextHop table.**
  - Set the destination of the next SF
- **table 87: SFF TransportEgress table.**
  - Prepare the packet for egress

The final table numbers may change depending on how they are assigned by Genius.

```
  // Pkt has NSH, send to table 86 for further processing
cookie=0x14 table=83, n_packets=11, n_bytes=918,
    priority=250,nsp=42
    actions=goto_table:86
  // Pkt does NOT have NSH, send back to Ingress Dispatcher
cookie=0x14 table=83, n_packets=0, n_bytes=0,
    priority=5
    actions=resubmit(,GENIUS_INGRESS_DISPATCHER_TABLE)

  // Table not used for NSH, shown for completeness
cookie=0x14 table=84, n_packets=0, n_bytes=0,
    priority=250
    actions=goto_table:86
```

```
  // Table not used for NSH, shown for completeness
cookie=0x14 table=85, n_packets=0, n_bytes=0,
    priority=250
    actions=goto_table:86

  // Match on specific NSH NSI/NSP, Encapsulate outer Ethernet
  // transport. Send to table 87 for further processing.
cookie=0x14 table=86, n_packets=11, n_bytes=918,
    priority=550,nsi=255,nsp=42
    actions=load:0xb00000c->NXM_NX_TUN_IPV4_DST[],
    goto_table:87
  // The rest of the packets are sent to
  // table 87 for further processing
cookie=0x14 table=86, n_packets=8, n_bytes=836,
    priority=5
    actions=goto_table:87

  // Match on specific NSH NSI/NSP, C1/C2 set
  // prepare pkt for egress, send to Egress Dispatcher
cookie=0xba5eba1100000101 table=87, n_packets=11, n_bytes=918,
        priority=650,nsi=255,nsp=42
        actions=move:NXM_NX_NSH_MDTYPE[]->NXM_NX_NSH_MDTYPE[],
                move:NXM_NX_NSH_NP[]->NXM_NX_NSH_NP[],
                move:NXM_NX_TUN_ID[0..31]->NXM_NX_TUN_ID[0..31],
                load:0x4->NXM_NX_TUN_GPE_NP[],
                resubmit(,GENIUS_EGRESS_DISPATCHER_TABLE)
```

Yang changes

The api YANGs used for the classifier build on the ietf acl models from the mdsal models.

Multiple options can be taken, depending on the desired functionality. Depending on the option chosen, YANG changes *might be* required.

Assuming no YANG changes, SFC classification will be performed on all VMs in the same neutron-network - this attribute is already present in the YANG model. **This is the proposed route**, since it hits a sweet-spot in the trade-off between functionality and risk.

If classifying the traffic from specific interfaces is desired, then the YANG model would need to be updated, possibly by adding a list of interfaces on which to classify.

Configuration impact

None

Clustering considerations

None

Other Infra considerations

Since SFC uses NSH, and the new Netvirt Classifier will need to add NSH encapsulation, a version of OVS that supports NSH must be used. NSH has not been officially accepted into the OVS project, so a branched version of OVS is used. Details about the branched version of OVS can be found here [9].

Security considerations

None

Scale and Performance Impact

None

Targeted Release

This change is targeted for the ODL Carbon release.

Alternatives

None

Usage

The new Netvirt Classifier will be configured via the REST JSON configuration mentioned in the REST API section below.

Features to Install

The existing old Netvirt SFC classifier is implemented in the following Karaf feature:

odl-ovsdb-sfc

When the new Netvirt SFC classifier is implemented, the previous Karaf feature will no longer be needed, and the following will be used:

odl-netvirt-sfc

REST API

The classifier REST API wont change from the old to the new Netvirt. The following example is how the old Netvirt classifier is configured.

Defined in netvirt/openstack/net-virt-sfc/api/src/main/yang/netvirt-acl.yang

An ACL is created which specifies the matching criteria and the action, which is to send the packets to an SFC RSP. Notice the "network-uuid" is set. This is for binding the Netvirt classifier service to a logical port. The procedure will be to query Genius for all the logical ports in that network uuid, and bind the Netvirt classifier service to each of them.

If the RSP has not been created yet, then the classification can not be created, since there wont be any information available about the RSP. In this case, the ACL information will be buffered, and there will be a separate listener for RSPs. When the referenced RSP is created, then the classifier processing will continue.

```
URL: /restconf/config/ietf-access-control-list:access-lists/

{
  "access-lists": {
    "acl": [
      {
        "acl-name": "ACL1",
        "acl-type": "ietf-access-control-list:ipv4-acl",
        "access-list-entries": {
          "ace": [
            {
              "rule-name": "ACE1",
              "actions": {
                "netvirt-sfc-acl:rsp-name": "RSP1"
              },
              "matches": {
                "network-uuid" : "eccb57ae-5a2e-467f-823e-45d7bb2a6a9a",
                "source-ipv4-network": "192.168.2.0/24",
                "protocol": "6",
                "source-port-range": {
                    "lower-port": 0
                },
                "destination-port-range": {
                    "lower-port": 80
                }
              }
            }
          ]
        }
      }]}}
```

CLI

None.

Implementation

Assignee(s)

Primary assignee:

- <brady.allen.johnson@ericsson.com>

Other contributors:

- <brady.allen.johnson@ericsson.com>
- <david.suarez.fuentes@ericsson.com
- <jaime.camaano.ruiz@ericsson.com>
- <miguel.duarte.de.mora.barroso@ericsson.com>

Work Items

Simple scenario:

- Augment the provisioned ACL with the 'neutron-network' augmentation - [11]
- From the neutron-network, get a list of neutron-ports - the interfaces connecting the VMs to that particular neutron-network. For each interface, do as follows:
  - Extract the DPN-ID of the node hosting the VM having that neutron-port
  - Extract the DPN-ID of the node hosting the first SF of the RSP
  - The forwarding logic to implement depends on the co-location of the client's VM with the first SF in the chain.
    - When the VMs are co-located (i.e. located in the same host), the output actions are to forward the packet to the first table of the SFC pipeline.
    - When the VMs are **not** co-located (i.e. hosted on different nodes) it is necessary to:
      - Use genius RPCs to get the interface connecting 2 DPN-IDs. This will return the tunnel endpoint connecting the compute nodes.
      - Use genius RPCs to get the list of actions to reach the tunnel endpoint.

Enabling VM mobility:

1. Handle first SF mobility

   Listen to RSP updates, where the only relevant migration is when the first SF moves to another node (different DPN-IDs). In this scenario, we delete the flows from the *old* node, and install the newly calculated flows in the new one. This happens for **each** node having an interface to classify attached to the provisioned neutron-network.

2. Handle client VM mobility

   Listen to client's InterfaceState changes, re-evaluating the Forwarding logic, since the tunnel interface used to reach the target DPN-ID is different. This means the action list to implement it, will also be different. The interfaces to listen to will be ones attached to the provisioned neutron-network.

3. **Must** keep all the nodes having interfaces to classify (i.e. nodes having neutron-ports attached to the neutron-network) and the first SF host node within the same transport zone. By listening to InterfaceState changes of clients within the neutron-network & the first SF neutron ports, the transport zone rendering can be redone.

   TODO: *is there a better way to identify when the transport zone needs to be updated?*

Dependencies

No dependency changes will be introduced by this change.

Testing

Unit Tests

Unit tests for the new Netvirt classifier will be modeled on the existing old Netvirt classifier unit tests, and tests will be removed and/or added appropriately.

Integration Tests

The existing old Netvirt Classifier Integration tests will need to be migrated to use the new Netvirt classifier.

CSIT

The existing Netvirt CSIT tests for the old classifier will need to be migrated to use the new Netvirt classifier.

Documentation Impact

User Guide documentation will be added by one of the following contributors:

- <brady.allen.johnson@ericsson.com>
- <david.suarez.fuentes@ericsson.com
- <jaime.camaano.ruiz@ericsson.com>
- <miguel.duarte.de.mora.barroso@ericsson.com>

References

[1] https://datatracker.ietf.org/doc/draft-ietf-sfc-nsh/

[2] https://datatracker.ietf.org/doc/rfc7665/

[3] https://wiki.opendaylight.org/view/Service_Function_Chaining:Main

[4] https://datatracker.ietf.org/doc/draft-ietf-nvo3-vxlan-gpe/

[5] https://docs.google.com/presentation/d/1kBY5PKPETEtRA4KRQ-GvVUSLbJoojPsmJlvpKyfZ5dU/edit?usp=sharing

[6] https://wiki.opnfv.org/display/sfc/Service+Function+Chaining+Home

[7] http://docs.opendaylight.org/en/stable-boron/user-guide/genius-user-guide.html

[8] https://wiki.opendaylight.org/view/Genius:Design_doc

[9] https://wiki.opendaylight.org/view/Service_Function_Chaining:Main#Building_Open_vSwitch_with_VxLAN-GPE_and_NSH_support

[10] http://docs.opendaylight.org/en/latest/submodules/genius/docs/pipeline.html

[11] https://github.com/opendaylight/netvirt/blob/master/openstack/net-virt-sfc/api/src/main/yang/netvirt-acl.yang

[12] https://docs.google.com/presentation/d/1gN8GnpVGwku4mp1on7EBZiE41RI7lZ-FFmFS2QlUTKk/edit?usp=sharing

Table of Contents

Netvirt Statistics

https://git.opendaylight.org/gerrit/#/q/topic:netvirt-counters

The feature enables getting statistics on ports and switches.

Problem description

Being able to ask for statistics, given as input Netvirt identifiers. It will enable filtering the results and having aggregated result. In a later stage, it will be also used to get element to element counters. Examples for possible filters: RX only, TX only, port + VLAN counters...

Use Cases

- Getting port counters, given its interface id (ietf interface name).

- Getting node counters, given its node id.

Port counters can be useful also to get statistics on traffic going into tunnels when requesting it from the tunnel endpoint port. In addition, there will also be support in aggregated results. For example: Getting the total number of transmitted packets from a given switch.

Proposed change

Adding a new bundle named "statistics-plugin" to Netvirt. This bundle will be responsible for converting the Netvirt unique identifiers into OpenFlow ones, and will get the relevant statistics by using OpenFlowPlugin capabilities. It will also be responsible of validating and filtering the results. It will be able to provide a wide range of aggregated results in the future.

Work flow description: Once a port statistics request is received, it is translated to a port statistics request from openflow plugin. Once the transaction is received, the data is validated and translated to a user friendly data. The user will be notified if a timeout occurs. In case of a request for aggregated counters, the user will receive a single counter result divided to groups (such as "bits", "packets"...). The counters in each group will be the sum of all of the matching counters for all ports. Neither one of the counter request nor the counter response will not be stored in the configuration database. Moreover, requests are not periodic and they are on demand only.

Pipeline changes

None

Yang changes

The new plugin introduced will have the following models:

```
  grouping result {
    list counterResult {
        key id;
        leaf id {
            type string;
        }
        list groups {
            key name;
            leaf name {
                type string;
            }
            list counters {
                key name;
                leaf name {
                    type string;
                }
                leaf value {
                    type uint64;
                }
            }
        }
    }
}

grouping filters {
    leaf-list groupFilters {
```

```
        type string;
    }
    leaf-list counterFilter {
        type string;
    }
}

rpc getNodeConnectorCounters {
    input {
        leaf portId {
            type string;
        }
        uses filters;
    }
    output {
        uses result;
    }
}

rpc getNodeCounters {
    input {
        leaf nodeId {
            type uint64;
        }
    }
    output {
        uses result;
    }
}

rpc getNodeAggregatedCounters {
    input {
        leaf nodeId {
            type uint64;
        }
        uses filters;
    }
    output {
        uses result;
    }
}
```

Configuration impact

None

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

None

Targeted Release

Carbon

Alternatives

Getting the statistics from OpenFlow flows: it would be possible to target the appropriate rules in ingress/egress tables, and count the hits on these flows. The reason we decided to work with ports instead is because we don't want to be dependent on flow structure changes.

Usage

- Create router, network, VMS, VXLAN tunnel.
- Connect to one of the VMs, send ping ping to the other VM.
- Use REST to get the statistics.

Port statistics:

```
http://10.0.77.135:8181/restconf/operational/ietf-interfaces:interfaces-state/
```

Choose a port id and use the following REST in order to get the statistics:

```
10.0.77.135:8181/restconf/operations/statistics-plugin:getNodeConnectorCounters,␣
↪input={"input":{"portId":"b99a7352-1847-4185-ba24-9ecb4c1793d9"}}, headers=
↪{Authorization=Basic YWRtaW46YWRtaW4=, Cache-Control=no-cache, Content-
↪Type=application/json}]
```

Node statistics:

```
http://10.0.77.135:8181/restconf/config/odl-interface-meta:bridge-interface-info/
```

Choose a node dpId and use the following REST in order to get the statistics:

```
10.0.77.135:8181/restconf/operations/statistics-plugin:getNodeCounters, input=
       {"input": { "portId": "b99a7352-1847-4185-ba24-9ecb4c1793d9","groups": [{ "name
↪": "byte*",
                           "counters": [{
                                                   "name": "rec*",
                                               }, {
                                                   "name": "transmitted*",
                                               }]
                                   }]
```

```
        }},
headers={Authorization=Basic YWRtaW46YWRtaW4=, Cache-Control=no-cache, Content-
↪Type=application/json}]
```

```
10.0.77.135:8181/restconf/operations/statistics-plugin:getNodeAggregatedCounters,␣
↪input=
       {"input": { "portId": "b99a7352-1847-4185-ba24-9ecb4c1793d9","groups": [{ "name
↪": "byte*",
                            "counters": [{
                                                        "name": "rec*",
                                                    }, {
                                                        "name": "transmitted*",
                                                    }]
                                        }]
        }},
headers={Authorization=Basic YWRtaW46YWRtaW4=, Cache-Control=no-cache, Content-
↪Type=application/json}]
```

Example for a filtered request:

```
10.0.77.135:8181/restconf/operations/statistics-plugin:getPortCounters, input={"input
↪": {"portId":"b99a7352-1847-4185-ba24-9ecb4c1793d9"} }, headers=
↪{Authorization=Basic YWRtaW46YWRtaW4=, Cache-Control=no-cache, Content-
↪Type=application/json}]
```

An example for node connector counters result:

```
   {
 "output": {
   "counterResult": [
     {
       "id": "openflow:194097926788804:5",
       "groups": [
         {
           "name": "Duration",
           "counters": [
             {
               "name": "durationNanoSecondCount",
               "value": 471000000
             },
             {
               "name": "durationSecondCount",
               "value": 693554
             }
           ]
         },
         {
           "name": "Bytes",
           "counters": [
             {
               "name": "bytesReceivedCount",
               "value": 1455
             },
             {
               "name": "bytesTransmittedCount",
               "value": 14151299
             }
           ]
```

```
          },
          {
            "name": "Packets",
            "counters": [
              {
                "name": "packetsReceivedCount",
                "value": 9
              },
              {
                "name": "packetsTransmittedCount",
                "value": 9
              }
            ]
          }
        ]
      }
    ]
  }
}
```

An example for node counters result:

```
   {
 "output": {
   "counterResult": [
     {
       "id": "openflow:194097926788804:3",
       "groups": [
         {
           "name": "Duration",
           "counters": [
             {
               "name": "durationNanoSecondCount",
               "value": 43000000
             },
             {
               "name": "durationSecondCount",
               "value": 694674
             }
           ]
         },
         {
           "name": "Bytes",
           "counters": [
             {
               "name": "bytesReceivedCount",
               "value": 0
             },
             {
               "name": "bytesTransmittedCount",
               "value": 648
             }
           ]
         },
         {
           "name": "Packets",
           "counters": [
             {
```

```
            "name": "packetsReceivedCount",
            "value": 0
          },
          {
            "name": "packetsTransmittedCount",
            "value": 0
          }
        ]
      }
    ]
  },
  {
    "id": "openflow:194097926788804:2",
    "groups": [
      {
        "name": "Duration",
        "counters": [
          {
            "name": "durationNanoSecondCount",
            "value": 882000000
          },
          {
            "name": "durationSecondCount",
            "value": 698578
          }
        ]
      },
      {
        "name": "Bytes",
        "counters": [
          {
            "name": "bytesReceivedCount",
            "value": 0
          },
          {
            "name": "bytesTransmittedCount",
            "value": 648
          }
        ]
      },
      {
        "name": "Packets",
        "counters": [
          {
            "name": "packetsReceivedCount",
            "value": 0
          },
          {
            "name": "packetsTransmittedCount",
            "value": 0
          }
        ]
      }
    ]
  },
  {
    "id": "openflow:194097926788804:1",
    "groups": [
```

```
        {
          "name": "Duration",
          "counters": [
            {
              "name": "durationNanoSecondCount",
              "value": 978000000
            },
            {
              "name": "durationSecondCount",
              "value": 698627
            }
          ]
        },
        {
          "name": "Bytes",
          "counters": [
            {
              "name": "bytesReceivedCount",
              "value": 6896336558
            },
            {
              "name": "bytesTransmittedCount",
              "value": 161078765
            }
          ]
        },
        {
          "name": "Packets",
          "counters": [
            {
              "name": "packetsReceivedCount",
              "value": 35644913
            },
            {
              "name": "packetsTransmittedCount",
              "value": 35644913
            }
          ]
        }
      ]
    },
    {
      "id": "openflow:194097926788804:LOCAL",
      "groups": [
        {
          "name": "Duration",
          "counters": [
            {
              "name": "durationNanoSecondCount",
              "value": 339000000
            },
            {
              "name": "durationSecondCount",
              "value": 698628
            }
          ]
        },
        {
```

```
          "name": "Bytes",
          "counters": [
            {
              "name": "bytesReceivedCount",
              "value": 0
            },
            {
              "name": "bytesTransmittedCount",
              "value": 0
            }
          ]
        },
        {
          "name": "Packets",
          "counters": [
            {
              "name": "packetsReceivedCount",
              "value": 0
            },
            {
              "name": "packetsTransmittedCount",
              "value": 0
            }
          ]
        }
      ]
    },
    {
      "id": "openflow:194097926788804:5",
      "groups": [
        {
          "name": "Duration",
          "counters": [
            {
              "name": "durationNanoSecondCount",
              "value": 787000000
            },
            {
              "name": "durationSecondCount",
              "value": 693545
            }
          ]
        },
        {
          "name": "Bytes",
          "counters": [
            {
              "name": "bytesReceivedCount",
              "value": 1455
            },
            {
              "name": "bytesTransmittedCount",
              "value": 14151073
            }
          ]
        },
        {
          "name": "Packets",
```

```
            "counters": [
              {
                "name": "packetsReceivedCount",
                "value": 9
              },
              {
                "name": "packetsTransmittedCount",
                "value": 9
              }
            ]
          }
        ]
      }
    ]
  }
}
```

Features to Install

odl-netvirt-openflowplugin-genius-openstack

REST API

CLI

Implementation

Assignee(s)

Primary assignee: Guy Regev <guy.regev@hpe.com>

Other contributors: TBD

Work Items

https://trello.com/c/ZdoLQWoV/126-netvirt-statistics

- Support port counters.
- Support node counters.
- Support aggregated results.
- Support filters on results.

Dependencies

- Genius
- OpenFlow Plugin
- Infrautils

Testing

Capture details of testing that will need to be added.

Unit Tests

Integration Tests

CSIT

Documentation Impact

References

Note: This template was derived from [2], and has been modified to support our project.

Table of Contents

Policy based path selection for multiple VxLAN tunnels

https://git.opendaylight.org/gerrit/#/q/topic:policy-based-path-selection

The purpose of this feature is to allow selection of primary and backup VxLAN tunnels for different types of VxLAN encapsulated traffic between a pair of OVS nodes based on some predefined policy.

Egress traffic can be classified using different characteristics e.g. 5-tuple, ingress port+VLAN, service-name to determine the best available path when multiple VxLAN endpoints are configured for the same destination.

Problem description

Today, netvirt is not able to classify traffic and route it over different tunnel endpoints based on a set of predefined characteristics. This is an essential infrastructure for applications on top of netvirt offering premium and personalized services.

Use Cases

- Forwarding of VxLAN traffic between hypervisors with multiple physical/logical ports.

Proposed change

The current implementation of transport-zone creation generates vtep elements based on the `local_ip` definition in the `other-config` column of the Open_vSwitch schema where the `local_ip` value represents the tunnel interface ip. This feature will introduce a new `other-config` property `local_ips`. `local_ips` will express the association between multiple tunnel ip addresses and multiple underlay networks using the following format:

```
local_ips=<tun1-ip>:<underlay1-net>,<tun2-ip>:<underlay2-net>,..,<tunN-ip>:<underlayN-
↪net>
```

Upon transport-zone creation, if the `local_ips` configuration is present, full tunnel mesh will be created between all TEP ips in the same underlay network considering the existing transport-zone optimizations i.e. tunnels will be created only between compute nodes with at least one spawned VM in the same VxLAN network or between networks connected to the same router if at least one of the networks is VxLAN-based.

Note that configuration of multiple tunnel IPs for the same DPN in the same underlay network is not a supported as part of this feature and requires further enhancements in both ITM and the transport-zone model.

The underlay networks are logical entities that will be used to distigush between multiple uplinks for routing of egress VxLAN traffic. They have no relation to Openstack and neutron networks definition. A new yang module is introduced to model the association between different types of OVS egress VxLAN traffic and the selected underlay network paths to output the traffic.

Policy-based path selection will be defined as a new egress tunnel service and depends on tunnel service binding functionality detailed in [3].

The policy service will be bounded only for tunnels of type logical tunnel group defined in [2].

The service will classify different types of traffic based on a predefined set of policy rules to find the best available path to route each type of traffic. The policy model will be agnostic to the specific topology details including DPN ids, tunnel interface and logical interface names. The only reference from the policy model to the list of preferred paths is made using underlay network-ids described earlier in this document.

Each policy references an ordered set of `policy-routes`. Each `policy-route` can be a `basic-route` referencing single underlay-network or `route-group` composed of multiple underlay networks. This set will get translated in each DPN to OF *fast-failover* group. The content of the buckets in each DPN depends on the existing underlay networks configured as part of the `local_ips` in the specific DPN.

The order of the buckets in the *fast-failover* group depends on the order of the underlay networks in the `policy-routes` model. `policy-routes` with similar set of routes in different order will be translated to different groups.

Each bucket in the *fast-failover* group can either reference a single tunnel or an additional OF *select* group depending on the type of policy route as detailed in the following table:

Policy route type	Bucket actions	OF Watch type
Basic route	load reg6(tun-lport) resubmit(220)	watch_port(tun-port)
Route group	goto_group(select-grp)	watch_group(select-grp)

This OF *select* group does not have the same content as the select groups defined in [2] and the content of its' buckets is based on the defined `route-group` elements and weights.

Logical tunnel will be bounded to the policy service if and only if there is at least one `policy-route` referencing one or more of the underlay networks in the logical group.

This service will take precedence over the default weighted LB service defined in [2] for logical tunnel group interfaces.

Policy-based path selection and weighted LB service pipeline example:

```
cookie=0x6900000, duration=0.802s, table=220, n_packets=0, n_bytes=0, priority=6,
↪reg6=0x500
actions=load:0xe000500->NXM_NX_REG6[],write_metadata:0xe000500000000000/
↪0xffffffff00000000,goto_table:230
cookie=0x6900000, duration=0.802s, table=220, n_packets=0, n_bytes=0, priority=6,
↪reg6=0xe000500
actions=load:0xf000500->NXM_NX_REG6[],write_metadata:0xf000500000000000/
↪0xffffffff00000000,group:800002
cookie=0x8000007, duration=0.546s, table=220, n_packets=0, n_bytes=0, priority=7,
↪reg6=0x600 actions=output:3
cookie=0x8000007, duration=0.546s, table=220, n_packets=0, n_bytes=0, priority=7,
↪reg6=0x700 actions=output:4
cookie=0x8000007, duration=0.546s, table=220, n_packets=0, n_bytes=0, priority=7,
↪reg6=0x800 actions=output:5
cookie=0x9000007, duration=0.546s, table=230, n_packets=0, n_bytes=0,priority=7,ip,
metadata=0x222e0/0xfffffffe,nw_dst=10.0.123.2,tp_dst=8080 actions=write_
↪metadata:0x200/0xfffffffe,goto_table:231
cookie=0x9000008, duration=0.546s, table=230, n_packets=0, n_bytes=0,priority=0,
↪resubmit(,220)
```

```
cookie=0x7000007, duration=0.546s, table=231, n_packets=0, n_bytes=0,priority=7,
↪metadata=0x500000000200/0xfffff00ffffffe,
actions=group:800000
cookie=0x9000008, duration=0.546s, table=231, n_packets=0, n_bytes=0,priority=0,
↪resubmit(,220)
group_id=800000,type=ff,
bucket=weight:0,watch_group=800001,actions=group=800001,
bucket=weight:0,watch_port=5,actions=load:0x800->NXM_NX_REG6[],resubmit(,220)
group_id=800001,type=select,
bucket=weight:50,watch_port=3,actions=load:0x600->NXM_NX_REG6[],resubmit(,220),
bucket=weight:50,watch_port=4,actions=load:0x700->NXM_NX_REG6[],resubmit(,220),
group_id=800002,type=select,
bucket=weight:50,watch_port=3,actions=load:0x600->NXM_NX_REG6[],resubmit(,220),
bucket=weight:25,watch_port=4,actions=load:0x700->NXM_NX_REG6[],resubmit(,220),
bucket=weight:25,watch_port=5,actions=load:0x800->NXM_NX_REG6[],resubmit(,220)
```

Each bucket in the *fast-failover* group will set the `watch_port` or `watch_group` property to monitor the liveness of the OF port in case of `basic-route` and underlay group in case of `route-group`. This will allow the OVS to route egress traffic only to the first live bucket in each *fast-failover* group.

The policy model rules will be based on IETF ACL data model [4]. The following enhancements are proposed for this model to support policy-based path selection:

	Name	Attributes	Description	OF implementation
ACE matches	ingress-interface	name	Policy match based on the ingress port and optionally the VLAN id	Match lport-tag metadata bits
		vlan-id		
	service	service-type	Policy match based on the service-name of L2VPN/L3VPN e.g. ELAN name/VPN instance name	Match service/vrf-id metadata bits depending on service-type
		service-name		
ACE actions	set policy-classifier	policy-classifier	Set policy classifier that can be later used for policy routing etc. Only the egress classifier will be used in this feature.	Set policy classifier in the metadata service bits
		direction		

To enable matching on previous services in the pipeline e.g. L2GW-VPN, the egress service binding for tunnel interfaces will be changed to preserve the metadata of preceding services rather than override it as done in the current implementation.

Each `policy-classifier` will be associated with `policy-route`. The same route can be shared by multiple classifiers.

The policy service will also maintain counters on number of policy rules assigned to underlay network per dpn in the operational DS.

Pipeline changes

- The following new tables will be added to support the policy-based path selection service:

Table Name	Matches	Actions
Policy classifier table (230)	ACE matches	ACE policy actions: set policy-classifier
Policy routing table (231)	match policy-classifier	set FF group-id

- Each Access List Entry (ACE) composed of standard and/or policy matches and policy actions will be translated to a flow in the policy classifier table.

  Each policy-classifier name will be allocated with id from a new pool - POLICY_SERVICE_POOL. Once a policy classifier has been determined for a given ACE match, the classifier-id will be set in the `service` bits of the metadata.

- Classified traffic will be sent from the policy classifier table to the policy routing table where the classifier-id will be matched to select the preferred tunnel using OF *fast-failover* group. Multiple classifiers can point to a single group.
- The default flow in the policy tables will resubmit traffic with no predefined policy/set of routes back to the egress dispatcher table in order to continue processing in the next bounded egress service.
- For all the examples below it is assumed that a logical tunnel group was configured for both ingress and egress DPNs. The logical tunnel group is composed of { `tun1`, `tun2`, `tun3` } and bound to a policy service.

Traffic between VMs on the same DPN

No pipeline changes required

L3 traffic between VMs on different DPNs

VM originating the traffic (Ingress DPN):

- Remote next hop group in the FIB table references the logical tunnel group.
- Policy service on the logical group selects the egress interface by classifying the traffic e.g. based on destination ip and port.

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id,dst-ip=vm2-ip set dst-mac=vm2-mac tun-id=vm2-label reg6=logical-tun-lport-tag` =>
Egress table (220) `match: reg6=logical-tun-lport-tag` =>
Policy classifier table (230) `match: vpn-id=router-id,dst-ip=vm2-ip,dst-tcp-port=8080 set egress-classifier=clf1` =>
Egress policy indirection table (231) `match: reg6=logical-tun-lport-tag,egress-classifier=clf1` =>
Logical tunnel tun1 FF group `set reg6=tun1-lport-tag` =>
Egress table (220) `match: reg6=tun1-lport-tag` output to `tun1`

VM receiving the traffic (Ingress DPN):

- No pipeline changes required

Classifier table (0) =>
Internal tunnel Table (36) `match:tun-id=vm2-label` =>
Local Next-Hop group: `set dst-mac=vm2-mac,reg6=vm2-lport-tag` =>
Egress table (220) `match: reg6=vm2-lport-tag` output to VM 2

SNAT traffic from non-NAPT switch

VM originating the traffic is non-NAPT switch:

- NAPT group references the logical tunnel group.
- Policy service on the logical group selects the egress interface by classifying the traffic based on the L3VPN service id.

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id` =>
Pre SNAT table (26) `match: vpn-id=router-id` =>
NAPT Group `set tun-id=router-id reg6=logical-tun-lport-tag` =>
Egress table (220) `match: reg6=logical-tun-lport-tag` =>
Policy classifier table (230) `match: vpn-id=router-id set egress-classifier=clf2` =>
Policy routing table (231) `match: reg6=logical-tun-lport-tag,egress-classifier=clf2` =>
Logical tunnel tun2 FF group `set reg6=tun2-lport-tag` =>
Egress table (220) `match: reg6=tun2-lport-tag` output to `tun2`

Traffic from NAPT switch punted to controller:

- No explicit pipeline changes required

Classifier table (0) =>
Internal tunnel Table (36) `match:tun-id=router-id` =>
Outbound NAPT table (46) `set vpn-id=router-id, punt-to-controller`

L2 unicast traffic between VMs in different DPNs

VM originating the traffic (Ingress DPN):

- ELAN DMAC table references the logical tunnel group
- Policy service on the logical group selects the egress interface by classifying the traffic based on the ingress port.

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) =>
Dispatcher table (17) `l2vpn service: set elan-tag=vxlan-net-tag` =>
ELAN base table (48) =>
ELAN SMAC table (50) `match: elan-tag=vxlan-net-tag,src-mac=vm1-mac` =>

ELAN DMAC table (51) `match: elan-tag=vxlan-net-tag,dst-mac=vm2-mac set tun-id=vm2-lport-tag reg6=logical-tun-lport-tag` =>
Egress table (220) `match: reg6=logical-tun-lport-tag` =>
Policy classifier table (230) `match: lport-tag=vm1-lport-tag set egress-classifier=clf3` =>
Policy routing table (231) `match: reg6=logical-tun-lport-tag,egress-classifier=clf3` =>
Logical tunnel tun1 FF group `set reg6=tun1-lport-tag` =>
Egress table (220) `match: reg6=tun1-lport-tag` output to `tun1`

VM receiving the traffic (Ingress DPN):

- No explicit pipeline changes required

Classifier table (0) =>
Internal tunnel Table (36) `match:tun-id=vm2-lport-tag set reg6=vm2-lport-tag` =>
Egress table (220) `match: reg6=vm2-lport-tag` output to VM 2

L2 multicast traffic between VMs in different DPNs with undefined policy

VM originating the traffic (Ingress DPN):

- ELAN broadcast group references the logical tunnel group.
- Policy service on the logical group has no classification for this type of traffic. Fallback to the default logical tunnel service - weighted LB [2].

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) =>
Dispatcher table (17) `l2vpn service: set elan-tag=vxlan-net-tag` =>
ELAN base table (48) =>
ELAN SMAC table (50) `match: elan-tag=vxlan-net-tag,src-mac=vm1-mac` =>
ELAN DMAC table (51) =>
ELAN DMAC table (52) `match: elan-tag=vxlan-net-tag` =>
ELAN BC group `goto_group=elan-local-group, set tun-id=vxlan-net-tag reg6=logical-tun-lport-tag` =>
Egress table (220) `match: reg6=logical-tun-lport-tag set reg6=default-egress-service&logical-tun-lport-tag` =>
Policy classifier table (230) =>
Egress table (220) `match: reg6=default-egress-service&logical-tun-lport-tag` =>
Logical tunnel LB select group `set reg6=tun2-lport-tag` =>
Egress table (220) `match: reg6=tun2-lport-tag` output to `tun2`

VM receiving the traffic (Ingress DPN):

- No explicit pipeline changes required

Classifier table (0) =>
Internal tunnel Table (36) `match:tun-id=vxlan-net-tag` =>
ELAN local BC group `set tun-id=vm2-lport-tag` =>
ELAN filter equal table (55) `match: tun-id=vm2-lport-tag set reg6=vm2-lport-tag` =>
Egress table (220) `match: reg6=vm2-lport-tag` output to VM 2

Yang changes

The following yang modules will be added to support policy-based routing:

Policy Service Yang

`policy-service.yang` define policy profiles and add augmentations on top of `ietf-access-control-list:access-lists` to apply policy classifications on access control entries.

```
module policy-service {
   yang-version 1;
   namespace "urn:opendaylight:netvirt:policy";
   prefix "policy";

   import ietf-interfaces { prefix if; }

   import ietf-access-control-list { prefix ietf-acl; }

   import aclservice { prefix acl; }

   import yang-ext { prefix ext; }

   import opendaylight-l2-types { prefix ethertype; revision-date "2013-08-27"; }

   description
       "Policy Service module";

   revision "2017-02-07" {
       description
               "Initial revision";
   }

   identity policy-acl {
       base ietf-acl:acl-base;
   }

   augment "/ietf-acl:access-lists/ietf-acl:acl/"
   + "ietf-acl:access-list-entries/ietf-acl:ace/ietf-acl:matches" {
       ext:augment-identifier "ingress-interface";
       leaf name {
           type if:interface-ref;
       }
```

```
        leaf vlan-id {
            type ethertype:vlan-id;
        }
    }

    augment "/ietf-acl:access-lists/ietf-acl:acl/"
    + "ietf-acl:access-list-entries/ietf-acl:ace/ietf-acl:matches" {
        ext:augment-identifier "service";
        leaf service-type {
            type identityref {
                base service-type-base;
            }
        }

        leaf service-name {
            type string;
        }
    }

    augment "/ietf-acl:access-lists/ietf-acl:acl/"
    + "ietf-acl:access-list-entries/ietf-acl:ace/ietf-acl:actions" {
        ext:augment-identifier "set-policy-classifier";
        leaf policy-classifier {
            type leafref {
                path "/policy-profiles/policy-profile/policy-classifier";
            }
        }

        leaf direction {
            type identityref {
                base acl:direction-base;
            }
        }
    }

    container underlay-networks {
        list underlay-network {
            key "network-name";
            leaf network-name {
                type string;
            }

            leaf network-access-type {
                type identityref {
                    base access-network-base;
                }
            }

            leaf bandwidth {
                type uint64;
                description "Maximum bandwidth. Units in byte per second";
            }

            list dpn-to-interface {
                config false;
                key "dp-id";
                leaf dp-id {
```

```
                type uint64;
            }

            list tunnel-interface {
                key "interface-name";
                leaf interface-name {
                    type string;
                }
            }
        }

        list policy-profile {
            config false;
            key "policy-classifier";
            leaf policy-classifier {
                type string;
            }
        }
    }
}

container underlay-network-groups {
    list underlay-network-group {
        key "group-name";
        leaf group-name {
            type string;
        }

        list underlay-network {
            key "network-name";
            leaf network-name {
                type leafref {
                    path "/underlay-networks/underlay-network/network-name";
                }
            }

            leaf weight {
                type uint16;
                default 1;
            }
        }

        leaf bandwidth {
            type uint64;
            description "Maximum bandwidth of the group. Units in byte per second";
        }
    }
}

container policy-profiles {
    list policy-profile {
        key "policy-classifier";
        leaf policy-classifier {
            type string;
        }

        list policy-route {
            key "route-name";
```

```
                leaf route-name {
                    type string;
                }

                choice route {
                    case basic-route {
                        leaf network-name {
                            type leafref {
                                path "/underlay-networks/underlay-network/network-name
↪";
                            }
                        }
                    }

                    case route-group {
                        leaf group-name {
                            type leafref {
                                path "/underlay-network-groups/underlay-network-group/
↪group-name";
                            }
                        }
                    }
                }
            }

            list policy-acl-rule {
                config false;
                key "acl-name";
                leaf acl-name {
                    type leafref {
                        path "/ietf-acl:access-lists/ietf-acl:acl/ietf-acl:acl-name";
                    }
                }

                list ace-rule {
                    key "rule-name";
                    leaf rule-name {
                        type leafref {
                            path "/ietf-acl:access-lists/ietf-acl:acl/ietf-acl:access-
↪list-entries/ietf-acl:ace/ietf-acl:rule-name";
                        }
                    }
                }
            }
        }
    }

    container policy-route-counters {
        config false;

        list underlay-network-counters {
            key "network-name";
            leaf network-name {
                type leafref {
                    path "/underlay-networks/underlay-network/network-name";
                }
            }
```

```
        list dpn-counters {
            key "dp-id";
            leaf dp-id {
                type uint64;
            }

            leaf counter {
                type uint32;
            }
        }

        list path-counters {
            key "source-dp-id destination-dp-id";
            leaf source-dp-id {
                type uint64;
            }

            leaf destination-dp-id {
                type uint64;
            }

            leaf counter {
                type uint32;
            }
        }
    }
}

identity service-type-base {
    description "Base identity for service type";
}

identity l3vpn-service-type {
    base service-type-base;
}

identity l2vpn-service-type {
    base service-type-base;
}

identity access-network-base {
    description "Base identity for access network type";
}

identity mpls-access-network {
    base access-network-base;
}

identity docsis-access-network {
    base access-network-base;
}

identity pon-access-network {
    base access-network-base;
}

identity dsl-access-network {
    base access-network-base;
```

```
    }

    identity umts-access-network {
        base access-network-base;
    }

    identity lte-access-network {
        base access-network-base;
    }
}
```

Policy service tree view

```
module: policy-service
   +--rw underlay-networks
   |  +--rw underlay-network* [network-name]
   |     +--rw network-name           string
   |     +--rw network-access-type?   identityref
   |     +--rw bandwidth?             uint64
   |     +--ro dpn-to-interface* [dp-id]
   |     |  +--ro dp-id               uint64
   |     |  +--ro tunnel-interface*
   |     |     +--ro interface-name?   string
   |     +--ro policy-profile* [policy-classifier]
   |        +--ro policy-classifier    string
   +--rw underlay-network-groups
   |  +--rw underlay-network-group* [group-name]
   |     +--rw group-name          string
   |     +--rw underlay-network* [network-name]
   |     |  +--rw network-name    -> /underlay-networks/underlay-network/network-name
   |     |  +--rw weight?         uint16
   |     +--rw bandwidth?          uint64
   +--rw policy-profiles
   |  +--rw policy-profile* [policy-classifier]
   |     +--rw policy-classifier    string
   |     +--rw policy-route* [route-name]
   |     |  +--rw route-name      string
   |     |  +--rw (route)?
   |     |     +--:(basic-route)
   |     |     |  +--rw network-name?   -> /underlay-networks/underlay-network/
↪network-name
   |     |     +--:(route-group)
   |     |        +--rw group-name?     -> /underlay-network-groups/underlay-network-
↪group/group-name
   |     +--ro policy-acl-rule* [acl-name]
   |        +--ro acl-name    -> /ietf-acl:access-lists/acl/acl-name
   |        +--ro ace-rule* [rule-name]
   |           +--ro rule-name    -> /ietf-acl:access-lists/acl/access-list-entries/
↪ace/rule-name
   +--ro policy-route-counters
      +--ro underlay-network-counters* [network-name]
         +--ro network-name     -> /underlay-networks/underlay-network/network-name
         +--ro dpn-counters* [dp-id]
         |  +--ro dp-id      uint64
         |  +--ro counter?   uint32
         +--ro path-counters* [source-dp-id destination-dp-id]
```

```
           +--ro source-dp-id         uint64
           +--ro destination-dp-id    uint64
           +--ro counter?             uint32
 augment /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:access-list-entries/ietf-
↪acl:ace/ietf-acl:matches:
   +--rw name?      if:interface-ref
   +--rw vlan-id?   ethertype:vlan-id
 augment /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:access-list-entries/ietf-
↪acl:ace/ietf-acl:matches:
   +--rw service-type?   identityref
   +--rw service-name?     string
 augment /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:access-list-entries/ietf-
↪acl:ace/ietf-acl:actions:
   +--rw policy-classifier?   -> /policy-profiles/policy-profile/policy-classifier
   +--rw direction?           identityref
```

Configuration impact

This feature introduces a new `other_config` parameter `local_ips` to support multiple ip:network associations as detailed above. Compatibility with the current `local_ip` parameter will be maintained but if both are present, `local_ips` would take presedence over `local_ip`.

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

None

Targeted Release

Carbon

Alternatives

None

Usage

Features to Install

odl-netvirt-openstack

REST API

Sample JSON data

Create policy rule

URL: restconf/config/ietf-access-control-list:access-lists

The following REST will create rule to classify all http traffic to ports 8080-8181 from specific vpn-id

```
{
  "access-lists": {
        "acl": [
          {
            "acl-type": "policy-service:policy-acl",
            "acl-name": "http-policy",
            "access-list-entries": {
              "ace": [
                {
                  "rule-name": "http-ports",
                  "matches": {
                    "protocol": 6,
                    "destination-port-range": {
                      "lower-port": 8080,
                      "upper-port": 8181
                    },
                    "policy-service:service-type": "l3vpn",
                    "policy-service:service-name": "71f7eb47-59bc-4760-8150-
↪e5e408d2ba10"
                  },
                  "actions": {
                    "policy-service:policy-classifier" : "classifier1",
                    "policy-service:direction" : "egress"
                  }
                }
              ]
            }
          }
        ]
      }
   }
 }
```

Create underlay networks

URL: restconf/config/policy-service:underlay-networks

The following REST will create multiple underlay networks with different access types

```
{
  "underlay-networks": {
    "underlay-network": [
      {
        "network-name": "MPLS",
        "network-access-type": "policy-service:mpls-access-network"
      },
      {
        "network-name": "DLS1",
        "network-access-type": "policy-service:dsl-access-network"
      },
      {
        "network-name": "DSL2",
        "network-access-type": "policy-service:dsl-access-network"
      }
    ]
  }
}
```

Create underlay group

URL: restconf/config/policy-service:underlay-network-groups

The following REST will create group for the DSL underlay networks

```
{
  "underlay-network-groups": {
    "underlay-network-group": [
      {
        "group-name": "DSL",
        "underlay-network": [
          {
            "network-name": "DSL1",
            "weight": 75
          },
          {
            "network-name": "DSL2",
            "weight": 25
          }
        ]
      }
    ]
  }
}
```

Create policy profile

URL: restconf/config/policy-service:policy-profiles

The following REST will create profile for classifier1 with multiple policy-routes

```
{
  "policy-profiles": {
    "policy-profile": [
      {
```

```
        "policy-classifier": "classifier1",
        "policy-route": [
          {
            "route-name": "primary",
            "network-name": "MPLS"
          },
          {
            "route-name": "backup",
            "group-name": "DSL"
          }
        ]
      }
    ]
  }
}
```

CLI

None

Implementation

Assignee(s)

Primary assignee: Tali Ben-Meir <tali@hpe.com>

Other contributors: Yair Zinger <yair.zinger@hpe.com>

Work Items

Trello card: https://trello.com/c/Uk3yrjUG/25-multiple-vxlan-endpoints-for-compute

- Transport-zone creation for multiple tunnels based on underlay network definitions
- Extract ACL flow programming to common location so it can be used by the policy service
- Create policy OF groups based on underlay network/group definitions
- Create policy classifier table based on ACL rules
- Create policy routing table
- Bind policy service to logical tunnels
- Maintain policy-route-counters per dpn/dpn-path

Dependencies

None

Testing

Unit Tests

Integration Tests

The test plan defined for CSIT below could be reused for integration tests.

CSIT

Adding multiple ports to the CSIT setups is challenging due to rackspace limitations. As a result, the test plan defined for this feature uses white-box methodology and not verifying actual traffic was sent over the tunnels.

Policy routing with single tunnel per access network type

- Set `local_ips` to contain tep ips for networks `underlay1` and `underlay2`
- Each underlay network will be defined with different `access-network-type`
- Create the following policy profiles
  - Profile1: `policy-classifier=clf1, policy-routes=underlay1, underlay2`
  - Profile2: `policy-classifier=clf2, policy-routes=underlay2, underlay1`
- Create the following policy rules
  - Policy rule 1: `dst_ip=vm2_ip,dst_port=8080 set_policy_classifier=clf1`
  - Policy rule 2: `src_ip=vm1_ip set_policy_classifier=clf2`
  - Policy rule 3: `service-type=l2vpn service-name=elan-name set_policy_classifier=clf1`
  - Policy rule 4: `service-type=l3vpn service-name=router-name set_policy_classifier=clf2`
  - Policy rule 5: `ingress-port=vm3_port set_policy_classifier=clf1`
  - Policy rule 6: `ingress-port=vm4_port vlan=vlan-id set_policy_classifier=clf2`
- Verify policy service flows/groups for all policy rules
- Verify flows/groups removal after the profiles were deleted

Policy routing with multiple tunnels per access network type

- Set `local_ips` to contain tep ips for networks `underlay1`..‘‘underlay4‘‘
- `underlay1`, `underlay2` and `underlay3`, `underlay4` are from the same `access-network-type`
- Create the following policy profiles where each route can be either group or basic route
  - Profile1: `policy-classifier=clf1, policy-routes={underlay1, underlay2}, {underlay3,underlay4}`
  - Profile2: `policy-classifier=clf2, policy-routes={underlay3,underlay4}, {underlay1, underlay2}`

  - Profile3: `policy-classifier=clf3, policy-routes=underlay1, {underlay3, underlay4}`
  - Profile4: `policy-classifier=clf4, policy-routes={underlay1, underlay2}, underlay3`
  - Profile5: `policy-classifier=clf5, policy-routes={underlay1, underlay2}`
  - Profile6: `policy-classifier=clf6, policy-routes=underlay4`
- Create the following policy rules
  - Policy rule 1: `dst_ip=vm2_ip,dst_port=8080 set_policy_classifier=clf1`
  - Policy rule 2: `src_ip=vm1_ip set_policy_classifier=clf2`
  - Policy rule 3: `service-type=l2vpn service-name=elan-name set_policy_classifier=clf3`
  - Policy rule 4: `service-type=l3vpn service-name=router-name set_policy_classifier=clf4`
  - Policy rule 5: `ingress-port=vm3_port set_policy_classifier=clf5`
  - Policy rule 6: `ingress-port=vm4_port vlan=vlan-id set_policy_classifier=clf6`
- Verify policy service flows/groups for all policy rules
- Verify flows/groups removal after the profiles were deleted

Documentation Impact

Netvirt documentation needs to be updated with description and examples of policy service configuration

References

[1] OpenDaylight Documentation Guide

[2] Load balancing and high availability of multiple VxLAN tunnels

[3] Service Binding On Tunnels

[4] Network Access Control List (ACL) YANG Data Model

Table of Contents

Support for QoS Alert

https://git.opendaylight.org/gerrit/#/q/topic:qos-alert

This feature adds support to monitor the per port packet drop counts when QoS rate limit rule is applied.

Problem description

If QoS bandwidth policy is applied on a neutron port, all packets exceeding the rate limit are dropped by the switch. This spec proposes a new service to monitor the packet drop ratio and log the alert message if packet drop ratio is greater than the configured threshold value.

Use Cases

Periodically monitor the port statistics of neutron ports having bandwidth limit rule and log an alert message in a log file if packet drop ratio cross the threshold value. Log file can be analyzed offline later to check the health/diagnostics of the network.

Proposed change

Proposed new service will use the RPC `/operations/opendaylight-direct-statistics:get-node-connector-sta` provided by openflowplugin to retrieve port statistics directly from switch by polling at regular interval. Polling interval is configurable with default value of 2 minutes.

Port packet drop ratio is calculated using delta of two port statistics counters `rx_dropped` and `rx_received` between the sample interval.

```
packet drop ratio = 100 * (rx_dropped / (rx_received + rx_dropped ))
```

An message is logged if packet drop ratio is greater than the configured threshold value.

Existing logging framework `log4j` shall be used to log the alert messages in the log file. A new appender `qosalertmsg` shall be added in `org.ops4j.pax.logging.cfg` to define the logging properties.

Log file format

```
2017-01-17 01:17:49,550 Packet drop threshold hit for qos policy qospolicy1 with qos-
↪id qos-2dbf02f6-dcd1-4c13-90ee-6f727e21fe8d for port port-3afde68d-1103-4b8a-a38d-
↪9cae631f7d67 on network network-563f9610-dd91-4524-ae23-8ec3c32f328e rx_received␣
↪4831 rx_dropped 4969
2017-01-17 01:17:49,550 Packet drop threshold hit for qos policy qospolicy2 with qos-
↪id qos-cb7e5f67-2552-4d49-b534-0ce90ebc8d97 for port port-09d3a437-f4a4-43eb-8655-
↪85df8bbe4793 on network network-389532a1-2b48-4ba9-9bcd-c1705d9e28f9 rx_received␣
↪3021 rx_dropped 4768
2017-01-17 01:19:49,339 Packet drop threshold hit for qos policy qospolicy1 with qos-
↪id qos-2dbf02f6-dcd1-4c13-90ee-6f727e21fe8d for port port-3afde68d-1103-4b8a-a38d-
↪9cae631f7d67 on network network-563f9610-dd91-4524-ae23-8ec3c32f328e rx_received␣
↪3837 rx_dropped 3961
2017-01-17 01:19:49,339 Packet drop threshold hit for qos policy qospolicy2 with qos-
↪id qos-cb7e5f67-2552-4d49-b534-0ce90ebc8d97 for port port-09d3a437-f4a4-43eb-8655-
↪85df8bbe4793 on network network-389532a1-2b48-4ba9-9bcd-c1705d9e28f9 rx_received␣
↪2424 rx_dropped 2766
```

Pipeline changes

None.

Yang changes

A new yang file shall be created for qos-alert configuration as specified below:

Listing 1.12: qos-alert-config.yang

```
module qosalert-config {

  yang-version 1;
  namespace "urn:opendaylight:params:xml:ns:yang:netvirt:qosalert:config";
  prefix "qosalert";

  revision "2017-01-03" {
    description "Initial revision of qosalert model";
  }
```

```
  description "This YANG module defines QoS alert configuration.";

  container qosalert-config {

    config true;

    leaf qos-alert-enabled {
      description "QoS alert enable-disable config knob";
      type boolean;
      default false;
    }

    leaf qos-drop-packet-threshold {
      description "QoS Packet drop threshold config. Specified as % of rx packets";
      type uint8 {
        range "1..100";
      }
      default 5;
    }

    leaf qos-alert-poll-interval {
      description "Polling interval in minutes";
      type uint16 {
        range "1..3600";
      }
      default 2;
    }

  }
}
```

Configuration impact

Following new parameters shall be made available as configuration. Initial or default configuration is specified in `netvirt-qosservice-config.xml`

Sl No.	configuration	Description
1.	`qos-alert-enabled`	configuration parameter to enable/disable the alerts
2.	`qos-drop-packet-threshold`	Drop percentage threshold configuration.
3.	`qos-alert-poll-interval`	Polling interval in minutes

Logging properties like log file name, location, size and maximum number of backup files are configured in file `org.ops4j.pax.logging.cfg`

Clustering considerations

In cluster setup, only one instance of qosalert service shall poll for port statistics. Entity owner service (EOS) shall be used to determine the owner of service.

Other Infra considerations

N.A.

Security considerations

None.

Scale and Performance Impact

QoS Alert Service minimizes scale and performance impact by following:

- Proposed service uses the direct-statistics RPC instead of OpenflowPlugin statistics-manager. This is lightweight because only node-connector statistics are queried instead of all statistics.
- Polling frequency is quite slow. Default polling interval is **two minutes** and minimum allowed value is 1 minute.

Targeted Release

Carbon.

Alternatives

N.A.

Usage

Features to Install

This feature can be used by installing `odl-netvirt-openstack`. This feature doesn't add any new karaf feature.

REST API

Put Qos Alert Config

Following API puts Qos Alert Config.

Method: POST

URI: /config/qosalert-config:qosalert-config

Parameters:

Parameter	Type	Value range	Comments
`qos-alert-enabled`	Boolean	true/false	Optional (default false)
`qos-drop-packet-threshold`	Uint16	1..100	Optional (default 5)
`qos-alert-poll-interval`	Uint16	1..65535	Optional time interval in minute(s) (default 2)

Example: .. code-block:: json

{

```
            "input": {
                "qos-alert-enabled": true,
                "qos-drop-packet-threshold": 35,
                "qos-alert-poll-interval": 5
        }
    }
```

CLI

Following new karaf CLIs are added

```
qos:enable-qos-alert <true|false>

qos:drop-packet-threshold <threshold value in %>

qos:alert-poll-interval <polling interval in minutes>
```

Implementation

Assignee(s)

Primary assignee:

- Arun Sharma (arun.e.sharma@ericsson.com)

Other contributors:

- Ravi Sundareswaran (ravi.sundareswaran@ericsson.com)
- Mukta Rani (mukta.rani@tcs.com)

Work Items

Trello Link <https://trello.com/c/780v28Yw/148-netvirt-qos-alert>

1. Adding new yang file and listener.
2. Adding new `log4j appender` in odlparent `org.ops4j.pax.logging.cfg` file.
3. Retrieval of port statistics data using the openflowplugin RPC.
4. Logging alert message into the log file.
5. UT and CSIT

Dependencies

This doesn't add any new dependencies.

Testing

Capture details of testing that will need to be added.

Unit Tests

Standard UTs will be added.

Integration Tests

N.A.

CSIT

Following new CSIT tests shall be added

1. Verify that alerts are generated if drop packets percentage is more than the configured threshold value.
2. Verify that alerts are not generated if drop packets percentage is less than threshold value.
3. Verify that alerts are not generated when `qos-alert-enabled` if false irrespective of drop packet percentage.

Documentation Impact

This will require changes to User Guide.

User Guide will need to add information on how qosalert service can be used.

References

[1] Neutron QoS

[2] Spec for NetVirt QoS

[3] Openflowplugin port statistics

Table of Contents

Neutron Quality of Service API Enhancements for NetVirt

QoS patches: https://git.opendaylight.org/gerrit/#/q/topic:qos

The Carbon release will enhance the initial implementation of Neutron QoS API[1] support for NetVirt which was released in Boron. The Boron released added support for Neutron QoS policies and the Egress bandwidth rate limiting rule. The Carbon release will update the QoS feature set of NetVirt by providing support for the DSCP Marking rule and QoS Rule capability reporting.

Problem description

It is important to be able to configure QoS attributes of workloads on virtual networks. The Neutron QoS API provides a method for defining QoS policies and associated rules which can be applied to Neutron Ports and Networks. These rules include:

- Egress Bandwidth Rate Limiting
- DSCP Marking

(Note that for the Neutron API, the direction of traffic flow (ingress, egress) is from the perspective of the OpenStack instance.)

[1] Neutron QoS http://docs.openstack.org/developer/neutron/devref/quality_of_service.html

As a Neutron provider for ODL, NetVirt will provide the ability to report back to Neutron its QoS rule capabilties and provide the ability to configure and manage the supported QoS rules on supported backends (e.g. OVS, ...). The key changes in the Carbon release will be the addition of support for the DSCP Marking rule.

Use Cases

Neutron QoS API support, including:

- Egress rate limiting - Drop traffic that exceeeds the specified rate parameters for a Neutron Port or Network.
- DSCP Marking - Set the DSCP field for IP packets arriving from Neutron Ports or Networks.
- Reporting of QoS capabilities - Report to Neutron which QoS Rules are supported.

Proposed change

To handle DSCP marking, listener support will be added to the *neutronvpn* service to respond to changes in DSCP Marking Rules in QoS Policies in the Neutron Northbound QoS models[2][3] .

To implement DSCP marking support, a new ingress (from vswitch perspective) QoS Service is defined in Genius. When DSCP Marking rule changes are detected, a rule in a new OpenFlow table for QoS DSCP marking rules will be updated.

The QoS service will be bound to an interface when a DSCP Marking rule is added and removed when the DSCP Marking rule is deleted. The QoS service follows the DHCP service and precedes the IPV6 service in the sequence of Genius ingress services.

Some use cases for DSCP marking require that the DSCP mark set on the inner packet be replicated to the DSCP marking in the outer packet. Therefore, for packets egressing out of OVS through vxlan/gre tunnels the option to copy the DSCP bits from the inner IP header to the outer IP header is needed. Marking of the inner header is done via OpenFlow rules configured on the corresponding Neutron port as described above. For cases where the outer tunnel header should have a copy of the inner header DSCP marking, the `tos` option on the tunnel interface in OVSDB must be configured to the value `inherit`. The setting of the `tos` option is done with a configurable parameter defined in the ITM module. By default the `tos` option is set to *0* as specified in the OVSDB specification[4] .

On the creation of new tunnels, the `tos` field will be set to either the user provided value or to the default value, which may be controlled via configuration. This will result in the tunnel-options field in the IFM (Interface Manager) to be set which will in turn cause the `options` field for the tunnel interface on the OVSDB node to be configured.

To implement QoS rule capability reporting back towards Neutron, code will be added to the *neutronvpn* service to populate the operational qos-rule-types list in the Neutron Northbound Qos model[3] with a list of the supported QoS rules - which will be the bandwidth limit rule and DSCP marking rule for the Carbon release.

Pipeline changes

A new QoS DSCP table is added to support the new QoS Service:

Table	Match	Action
QoS DSCP [90]	Ethtype == IPv4 or IPv6 AND LPort tag	Mark packet with DSCP value

[2] Neutron Northbound QoS Model Extensions https://github.com/opendaylight/neutron/blob/master/model/src/main/yang/neutron-qos-ext.yang
[3] Neutron Northbound QoS Model https://github.com/opendaylight/neutron/blob/master/model/src/main/yang/neutron-qos.yang
[4] OVSDB Schema http://openvswitch.org/ovs-vswitchd.conf.db.5.pdf

Yang changes

A new leaf `option-tunnel-tos` is added to `tunnel-end-points` in *itm-state.yang* and to `vteps` in *itm.yang*.

Listing 1.13: itm-state.yang

```
list tunnel-end-points {
    ordered-by user;
    key "portname VLAN-ID ip-address tunnel-type";

    leaf portname {
        type string;
    }
    leaf VLAN-ID {
        type uint16;
    }
    leaf ip-address {
        type inet:ip-address;
    }
    leaf subnet-mask {
        type inet:ip-prefix;
    }
    leaf gw-ip-address {
        type inet:ip-address;
    }
    list tz-membership {
        key "zone-name";
        leaf zone-name {
            type string;
        }
    }
    leaf interface-name {
        type string;
    }
    leaf tunnel-type {
        type identityref {
            base odlif:tunnel-type-base;
        }
    }
    leaf option-of-tunnel {
        description "Use flow based tunnels for remote-ip";
        type boolean;
        default false;
    }
    leaf option-tunnel-tos {
        description "Value of ToS bits to be set on the encapsulating
            packet.  The value of 'inherit' will copy the DSCP value
            from inner IPv4 or IPv6 packets.  When ToS is given as
            and numberic value, the least significant two bits will
            be ignored. ";
        type string;
    }
}
```

Listing 1.14: itm.yang

```
list vteps {
    key "dpn-id portname";
    leaf dpn-id {
        type uint64;
    }
    leaf portname {
        type string;
    }
    leaf ip-address {
        type inet:ip-address;
    }
    leaf option-of-tunnel {
        description "Use flow based tunnels for remote-ip";
        type boolean;
        default false;
    }
    leaf option-tunnel-tos {
        description "Value of ToS bits to be set on the encapsulating
            packet.  The value of 'inherit' will copy the DSCP value
            from inner IPv4 or IPv6 packets.  When ToS is given as
            and numberic value, the least significant two bits will
            be ignored. ";
        type string;
    }
}
```

A configurable parameter `default-tunnel-tos` is added to *itm-config.yang* which defines the default ToS value to be applied to tunnel ports.

Listing 1.15: itm-config.yang

```
container itm-config {
    config true;

    leaf default-tunnel-tos {
        description "Default value of ToS bits to be set on the encapsulating
            packet.  The value of 'inherit' will copy the DSCP value
            from inner IPv4 or IPv6 packets.  When ToS is given as
            and numberic value, the least significant two bits will
            be ignored. ";
        type string;
        default 0;
    }
}
```

Configuration impact

A configurable parameter `default-tunnel-tos` is added to *genius-itm-config.xml* which specifies the default ToS to use on a tunnel if it is not specified by the user when a tunnel is created. This value may be set to `inherit` for some DSCP Marking use cases.

Listing 1.16: genius-itm-config.xml

```
<itm-config xmlns="urn:opendaylight:genius:itm:config">
    <default-tunnel-tos>0</default-tunnel-tos>
</itm-config>
```

Clustering considerations

None.

Other Infra considerations

None.

Security considerations

None.

Scale and Performance Impact

Additional OpenFlow packets will be generated to configure DSCP marking rules in response to QoS Policy changes coming from Neutron.

Targeted Release

Carbon

Alternatives

Use of OpenFlow meters was desired, but the OpenvSwitch datapath implementation does not support meters (although the OpenvSwitch OpenFlow protocol implementation does support meters).

Usage

The user will use the QoS support by enabling and configuring the QoS extension driver for networking-odl. This will allow QoS Policies and Rules to be configured for Neuetron Ports and Networks using Neutron.

Perform the following configuration steps:

- In *neutron.conf* enable the QoS service by appending `qos` to the `service_plugins` configuration:

Listing 1.17: /etc/neutron/neutron.conf

```
service_plugins = odl-router, qos
```

- Add the QoS notification driver to the *neutron.conf* file as follows:

Listing 1.18: /etc/neutron/neutron.conf

```
[qos]
notification_drivers = odl-qos
```

- Enable the QoS extension driver for the core ML2 plugin. In file *ml2.conf.ini* append `qos` to `extension_drivers`

Listing 1.19: /etc/neutron/plugins/ml2/ml2.conf.ini

```
[ml2]
extensions_drivers = port_security,qos
```

Features to Install

Install the ODL Karaf feature for NetVirt (no change):

- odl-netvirt-openstack

REST API

None.

CLI

Refer to the Neutron CLI Reference[5] for the Neutron CLI command syntax for managing QoS policies and rules for Neutron networks and ports.

Implementation

Assignee(s)

Who is implementing this feature? In case of multiple authors, designate a primary assigne and other contributors.

Primary assignee:

- Poovizhi Pugazh <poovizhi.p@ericsson.com>

Other contributors:

- Ravindra Nath Thakur <ravindra.nath.thakur@ericsson.com>
- Eric Multanen <eric.w.multanen@intel.com>
- Praveen Mala <praveen.mala@intel.com> (including CSIT)

Work Items

Task list in Carbon Trello: https://trello.com/c/bLE2n2B1/14-qos

[5] Neutron CLI Reference http://docs.openstack.org/cli-reference/neutron.html#neutron-qos-available-rule-types

Dependencies

Genius project - Code[6] to support QoS Service needs to be added.

Neutron Northbound - provides the Neutron QoS models for policies and rules (already done).

Following projects currently depend on NetVirt: Unimgr

Testing

Capture details of testing that will need to be added.

Unit Tests

Integration Tests

CSIT

Documentation Impact

Documentation to describe use of Neutron QoS support with NetVirt will be added.

OpenFlow pipeline documentation updated to show QoS service table.

References

http://specs.openstack.org/openstack/neutron-specs/specs/newton/ml2-qos-with-dscp.html

ODL gerrit adding QoS models to Neutron Northbound: https://git.opendaylight.org/gerrit/#/c/37165/

Table of Contents

[6] Genius code supporting QoS service https://git.opendaylight.org/gerrit/#/c/49084/

Setup Source-MAC-Address for routed packets destined to virtual endpoints

https://git.opendaylight.org/gerrit/#/q/topic:SMAC_virt_endpoints

All L3 Routed packets destined to virtual endpoints in the datacenter managed by ODL do not carry a proper source-mac address in such frames put out to virtual endpoints.

This spec makes sure a proper source-mac is updated in the packet at the point where the packet is delivered to the VM, regardless of the tenant network type. On the actual datapath, there will be no change in the source mac-addresses and packets continue to use the same mechanism that is used today.

Addressing the datapath requires unique MAC allocation per OVS Datapath, so that it can be used as the source MAC for all distributively routed packets of an ODL enabled cloud. It would be handled in some future spec.

Problem description

Today all L3 Routed packets destined to virtual endpoints in the datacenter either

- Incorrectly carry the source mac-address of the originator (regardless of which network the originator is in)
- Incorrectly carry sometimes the reserved source mac address of 00:00:00:00:00:00

This spec is intended to setup a source-mac-address in the frame of L3 Routed packets just before such frames are directed into the virtual endpoints themselves. This enables use-cases where certain virtual endpoints which are VNFs in the datacenter that are source-mac conscious (or mandate that src-mac in frames be valid) can become functional on their instantiation in an OpenDaylight enabled cloud.

Use Cases

- Intra-Datacenter L3 forwarded packets within a hypervisor.
- Intra-Datacenter L3 forwarded packets over Internal VXLAN Tunnels between two hypervisors in the datacenter.
- Inter-Datacenter L3 forwarded packets :
  - Destined to VMs associated floating IP over External VLAN Provider Networks.
  - Destined to VMs associated floating IP over External MPLSOverGRE Tunnels.
  - SNAT traffic from VMs over External MPLSOverGRE Tunnels.
  - SNAT traffic from VMS over External VLAN Provider Networks.

Proposed change

All the L3 Forwarded traffic today reaches the VM via a LocalNextHopGroup managed by the VPN Engine (including FIBManager).

Currently the LocalNextHopGroup sets-up the destination MAC Address of the VM and forwards the traffic to EGRESS_LPORT_DISPATCHER_TABLE (Table 220). In that LocalNextHopGroup we will additionally setup source-mac-address for the frame. There are two cases to decide what source-mac-address should go into the frame:

- If the VM is on a subnet (on a network) for which a subnet gatewayip port exists, then the source-mac address of that subnet gateway port will be setup as the frame's source-mac inside the LocalNextHop group.This is typical of the case when a subnet is added to a router, as the router interface port created by neutron will be representing the subnet's gateway-ip address.
- If the VM is on a subnet (on a network), for which there is no subnet gatewayip port but that network is part of a BGPVPN , then the source-mac address would be that of the connected mac-address of the VM itself. The connected mac-address is nothing but the mac-address on the ovs-datapath for the VMs tapxxx/vhuxxx port on that hypervisor itself.

The implementation also applies to Extra-Routes (on a router) and Discovered Routes as they both use the LocalNextHopGroup in their last mile to send packets into their Nexthop VM.

We need to note that when a network is already part of a BGPVPN, adding a subnet on such a network to a router is disallowed currently by NeutronVPN. And so the need to swap the mac-addresses inside the LocalNextHopGroup to reflect the subnet gatewayip port here does not arise.

For all the use-cases listed in the USE-CASES section above, proper source mac address will be filled-up in the frame before it enters the virtual endpoint.

Pipeline changes

There are no pipeline changes.

The only change is in the NextHopGroup created by VPN Engine (i.e., VRFEntryListener). In the NextHopGroup we will additionally fill up the ethernet source mac address field with proper mac-address as outlined in the 'Proposed change' section.

Currently the LocalNextHopGroup is used in the following tables of VPN Pipeline:

- L3_LFIB_TABLE (Table 20) - Lands all routed packets from MPLSOverGRE tunnel into the virtual endpoint.
- INTERNAL_TUNNEL_TABLE (Table 36) - Lands all routed packets on Internal VXLAN Tunnel within the DC into the virtual end point.

- L3_FIB_TABLE (Table 21) - Lands all routed packets within a specific hypervisor into the virtual endpoint.

```
cookie=0x8000002, duration=50.676s, table=20, n_packets=0, n_bytes=0, priority=10,
↪mpls,mpls_label=70006 actions=write_actions(pop_mpls:0x0800,group:150000)
cookie=0x8000003, duration=50.676s, table=21, n_packets=0, n_bytes=0, priority=42,ip,
↪metadata=0x222f2/0xfffffffe,nw_dst=10.1.1.3 actions=write_actions(group:150000)
cookie=0x9011176, duration=50.676s, table=36, n_packets=0, n_bytes=0, priority=5,tun_
↪id=0x11176 actions=write_actions(group:150000)

NEXTHOP GROUP:
group_id=150000,type=all,bucket=actions=set_field:fa:16:3e:01:1a:40->eth_src,set_
↪field:fa:16:3e:8b:c5:51->eth_dst,load:0x300->NXM_NX_REG6[],resubmit(,220)
```

Yang changes

None.

Configuration impact

None.

Clustering considerations

None.

Other Infra considerations

None.

Security considerations

None.

Scale and Performance Impact

None

Targeted Release

Carbon/Boron

Alternatives

None.

Usage

N/A.

Features to Install

odl-netvirt-openstack

REST API

N/A.

CLI

N/A.

Implementation

Assignee(s)

Primary assignee:

- Achuth Maniyedath (achuth.m@altencalsoftlabs.com)

Other contributors:

- Karthik Prasad (karthik.p@altencalsoftlabs.com)
- Vivekanandan Narasimhan (n.vivekanandan@ericsson.com)

Work Items

https://trello.com/c/IfAmnFFr/110-add-source-macs-in-frames-for-l3-routed-packets-before-such-frames-get-to-the-virtual-endpoint

- Determine the smac address to be used for L3 packets forwarded to VMs.
- Update the LocalNextHopGroup table with proper ethernet source-mac parameter.

Dependencies

No new dependencies.

Testing

Verify the Source-MAC-Address setting on frames forwarded to Virtual endpoints in following cases.

Intra-Datacenter traffic to VMs (Intra/Inter subnet).

- VM to VM traffic within a hypervisor.
- VM to VM traffic across hypervisor over Internal VXLAN tunnel.

Inter-Datacenter traffic to/from VMs.

- External access to VMs using Floating IPs on MPLSOverGRE tunnels.
- External access to VMs using Floating IPs over VLAN provider networks.
- External access from VMs using SNAT over VLAN provider networks.
- External access from VMs using SNAT on MPLSOverGRE tunnels.

Unit Tests

N/A.

Integration Tests

N/A.

CSIT

- Validate that router-interface src-mac is available on received frames within the VM when that VM is on a router-arm.
- Validate that connected-mac as src-mac available on received frames within the VM when that VM is on a network-driven L3 BGPVPN.

Documentation Impact

N/A

References

N/A

Table of Contents

Support for TCP MD5 Signature Option configuration of Quagga BGP

https://git.opendaylight.org/gerrit/#/q/topic:qbgp-tcp-md5-signature-option

This functionality adds support to odl-netvirt-impl feature to configure the TCP MD5 Signature Option *[RFC2385]* password in Quagga BGPs *[QBGP]*.

Problem description

Quagga *[QBGP]* supports TCP MD5 Signature Option *[RFC2385]* in BGP traffic but current odl-netvirt-impl feature implementation lacks support to configure the required passwords.

Use Cases

UC1: Protect (Quagga *[QBGP]*) BGP and DC gateway BGP interface using TCP MD5 Signature Option *[RFC2385]*.

Proposed change

The following components need to be enhanced:

- BGP Manager

Pipeline changes

No pipeline changes.

API changes

Changes will be needed in `ebgp.yang`, and `qbgp.thrift`.

YANG changes

A new optional leaf with the TCP MD5 Signature Option *[RFC2385]* password is added (by means of a choice) to list `neighbors`.

Listing 1.20: ebgp.yang additions

```
typedef tcp-md5-signature-password-type {
  type string {
    length 1..80;
  } // subtype string
  description
    "The shared secret used by TCP MD5 Signature Option.  The length is
     limited to 80 chars because A) it is identified by the RFC as current
     practice and B) it is the maximum length accepted by Quagga
     implementation.";
  reference "RFC 2385";
} // typedef tcp-md5-signature-password-type


grouping tcp-security-option-grouping {
  description "TCP security options.";
  choice tcp-security-option {
    description "The tcp security option in use, if any.";

    case tcp-md5-signature-option {
      description "The connection uses TCP MD5 Signature Option.";
      reference "RFC 2385";
      leaf tcp-md5-signature-password {
        type tcp-md5-signature-password-type;
        description "The shared secret used to sign the packets.";
      } // leaf tcp-md5-signature-password
    } // case tcp-md5-signature-option

  } // choice tcp-security-option
} // grouping tcp-security-option-grouping
```

Listing 1.21: ebgp.yang modifications

```
    list neighbors {
      key "address";
      leaf address {
        type inet:ipv4-address;
        mandatory "true";
      }
      leaf remote-as {
        type uint32;
```

```
        mandatory "true";
      }
 +    use tcp-security-option-grouping;
```

Thrift changes

A new function `setPeerSecret` is added to the service `BgpConfigurator`.

Listing 1.22: qbgp.thrift modifications

```
--- a/vpnservice/bgpmanager/bgpmanager-impl/src/main/java/org/opendaylight/netvirt/
↪bgpmanager/thrift/idl/qbgp.thrift
+++ b/vpnservice/bgpmanager/bgpmanager-impl/src/main/java/org/opendaylight/netvirt/
↪bgpmanager/thrift/idl/qbgp.thrift
@@ -31,6 +31,8 @@ const i32 GET_RTS_NEXT = 1
  * ERR_NOT_ITER when GET_RTS_NEXT is called without
  *    initializing with GET_RTS_INIT
  * ERR_PARAM when there is an issue with params
+ * ERR_NOT_SUPPORTED when the server does not support
+ *    the operation.
  */

 const i32 BGP_ERR_FAILED = 1
@@ -38,6 +40,7 @@ const i32 BGP_ERR_ACTIVE = 10
 const i32 BGP_ERR_INACTIVE = 11
 const i32 BGP_ERR_NOT_ITER = 15
 const i32 BGP_ERR_PARAM = 100
+const i32 BGP_ERR_NOT_SUPPORTED = 200

 // these are the supported afi-safi combinations
 enum af_afi {
@@ -122,6 +125,33 @@ service BgpConfigurator {
                         6:i32 stalepathTime, 7:bool announceFlush),
     i32 stopBgp(1:i64 asNumber),
     i32 createPeer(1:string ipAddress, 2:i64 asNumber),
+
+    /* 'setPeerSecret' sets the shared secret needed to protect the peer
+     * connection using TCP MD5 Signature Option (see rfc 2385).
+     *
+     * Params:
+     *
+     *   'ipAddress' is the peer (neighbour) address.  Mandatory.
+     *
+     *   'rfc2385_sharedSecret' is the secret.  Mandatory.  Length must be
+     *   greater than zero.
+     *
+     * Return codes:
+     *
+     *   0 on success.
+     *
+     *   BGP_ERR_FAILED if 'ipAddress' is missing or unknown.
+     *
+     *   BGP_ERR_PARAM if 'rfc2385_sharedSecret' is missing or invalid (e.g.
+     *   it is too short or too long).
+     *
+     *   BGP_ERR_INACTIVE when there is no session.
```

```
+     *
+     *   BGP_ERR_NOT_SUPPORTED when TCP MD5 Signature Option is not supported
+     *   (e.g. the underlying TCP stack does not support it)
+     *
+     */
+    i32 setPeerSecret(1:string ipAddress, 2:string rfc2385_sharedSecret),
     i32 deletePeer(1:string ipAddress)
     i32 addVrf(1:layer_type l_type, 2:string rd, 3:list<string> irts, 4:list<string>␣
↪erts),
     i32 delVrf(1:string rd),
```

An old server (i.e. using a previous version of `qbgp.thrift`) will return a `TApplicationException` with type `UNKNOWN_METHOD`. See *[TBaseProcessor]*.

Configuration impact

No configuration parameters deprecated.

New optional leaf `tcp-md5-signature-password` does not impact existing deployments.

The recommended AAA configuration (See *Security considerations*) may impact existing deployments.

Clustering considerations

NA

Other Infra considerations

Signature mismatch

On signature mismatch TCP MD5 Signature Option *[RFC2385]* (page 2) specifies the following behaviour:

Listing 1.23: RFC 2385 page 2

```
Upon receiving a signed segment, the receiver must validate it by
calculating its own digest from the same data (using its own key) and
comparing the two digest.  A failing comparison must result in the
segment being dropped and must not produce any response back to the
sender.  Logging the failure is probably advisable.
```

A BGP will be unable to connect with a neighbor with a wrong password because the TCP SYN,ACK will be dropped. The neighbor state will bounce between “Active” and “Connect” while it retries.

Security considerations

`tcp-md5-signature-password` is stored in clear in the datastore. This is a limitation of the proposed change.

Because `tcp-md5-signature-password` is stored in clear the REST access to `neighbors` list should be restricted. See the following AAA configuration examples:

Listing 1.24: etc/shiro.ini example

```
#
# DISCOURAGED since Carbon
#
/config/ebgp:bgp/neighbors/** = authBasic, roles[admin]
```

Listing 1.25: AAA MDSALDynamicAuthorizationFilter example

```
{ "aaa:policies":
   {  "aaa:policies": [
      {  "aaa:resource": "/restconf/config/ebgp:bgp/neighbors/**",
         "aaa:permissions": [
         {  "aaa:role": "admin",
            "aaa:actions": [ "get","post","put","patch","delete" ]
         } ]
      } ]
   }
}
```

If `BgpConfigurator` thrift service is not secured then `tcp-md5-signature-password` goes clear on the wire.

Quagga *[QBGP]* (up to version 1.0) keeps the password in memory in clear. The password can be retrieved through Quagga's configuration interface.

Scale and Performance Impact

Negligible scale or performance impacts.

- datastore: A bounded (<=80) string per configured neighbor.
- Traffic (thrift `BgpConfigurator` service): A bounded (<=80) string field per neighbor addition operation.

Targeted Release

Carbon

Alternatives

Three alternatives have been considered in order to avoid storing the plain password in datastore: RPC, post-update, and transparent encryption. They are briefly described below.

The best alternative is transparent encryption, but in Carbon time-frame is not feasible.

The post-update alternative does not actually solve the limitation.

The RPC alternative is feasible in Carbon time-frame but, given that currently `BgpConfigurator` thrift service is not secured, to add an RPC does not pull its weight.

RPC encryption

A new RPC `add-neighbor(address, as-number[, tcp-md5-signature-password])` is in charge of create `neighbors` elements. The password is salted and encrypted with `aaa-encryption-service`. Both

the salt and the encrypted password are stored in the `neighbors` element.

Post-update encryption

The `neighbors` element contains both a `plain-password` leaf and a `encrypted-password-with-salt` leaf. The listener `BgpConfigurationManager.NeighborsReactor` is in charge of encrypt and remove the `plain-password` leaf when it is present (and the encrypted one is not).

This alternative does not really solve the limitation because during a brief period the password is stored in plain.

Transparent encryption

A plain value is provided in REST write operations but it is *automagically* encrypted before it reaches MD-SAL. Read operations never decrypts the encrypted values.

This alternative impacts at least `aaa`, `yangtools`, and `netconf` projects. It can not possibly be done in Carbon.

Usage

Features to Install

odl-netvirt-openstack

REST API

The RESTful API for neighbors creation (`/restconf/config/ebgp:bgp/neighbors/{address}`) will be enhanced to accept an additional `tcp-md5-signature-password` attribute:

```
{ "neighbors": {
   "address": "192.168.50.2",
   "remote-as": "2791",
   "tcp-md5-signature-password": "password"
}}
```

CLI

A new option `--tcp-md5-password` will be added to commands `odl:configure-bgp` and `odl:bgp-nbr`.

```
opendaylight-user@root> odl:configure-bgp -op add-neighbor --ip 192.168.50.2 --as-num␣
↪2791 --tcp-md5-password password
opendaylight-user@root> odl:bgp-nbr --ip-address 192.168.50.2 --as-number 2791 --tcp-
↪md5-password password add
```

Implementation

Assignee(s)

Primary assignee: Jose-Santos Pulido, JoseSantos, jose.santos.pulido.garcia@ericsson.com

Other contributors: TBD

Work Items

- https://trello.com/c/87MAFjRf

1. Spec
2. `ebgp.yang`
3. `BgpConfigurator` thrift service (both idl and client)
4. `BgpConfigurationManager.NeighborsReactor`
5. `ConfigureBgpCli`

Dependencies

Internal

No internal dependencies are added or removed.

External

To enable TCP MD5 Signature Option *[RFC2385]* in a BGP the following conditions need to be met:

- `BgpConfigurator` thrift service provider (e.g. `Zebra Remote Procedure Call` *[ZRPC]*) must support the new function `setPeerSecret`.
- BGP's TCP stack must support TCP MD5 Signature Option (e.g. in linux the kernel option CONFIG_TCP_MD5SIG must be set).

Testing

Unit Tests

Currently `bgpmanager` has no unit tests related to configuration.

Integration Tests

Currently `bgpmanager` has no integration tests.

CSIT

Currently there is no CSIT test exercising `bgpmanager`.

Documentation Impact

Currently there is no documentation related to `bgpmanager`.

References

Table of Contents

Support of VXLAN based L2 connectivity across Datacenters

https://git.opendaylight.org/gerrit/#/q/topic:EVPN_RT2

Enable realization of L2 connectivity over VXLAN tunnels using L2 BGPVPNs, internally taking advantage of EVPN as the BGP Control Plane mechanism.

Problem description

OpenDaylight NetVirt service today supports L3VPN connectivity over VXLAN tunnels. L2DCI communication is not possible so far.

This spec attempts to enhance the BGPVPN service in NetVirt to embrace inter-DC L2 connectivity over external VXLAN tunnels.

In scope

The scope primarily includes providing ability to support intra-subnet connectivity across DataCenters over VXLAN tunnels using BGP EVPN with type L2.

When we mention that we are using EVPN BGP Control plane, this spec proposes using the RouteType 2 as the primary means to provision the control plane to enable inter-DC connectivity over external VXLAN tunnels.

With this inplace we will be able to support the following.

- Intra-subnet connectivity across dataCenters over VXLAN tunnels.

The following are already supported as part of the other spec(RT5) and will continue to function.

- Intra-subnet connectivity within a DataCenter over VXLAN tunnels.
- Inter-subnet connectivity within a DataCenter over VXLAN tunnels.
- Inter-subnet connectivity across dataCenters over VXLAN tunnels.

Out of scope

Use Cases

The following high level use-cases will be realized by the implementation of this Spec.

Datacenter access from another Datacenter over WAN via respective DC-Gateways (L2 DCI)

This use-case involves providing intra-subnet connectivity between two DataCenters. Tenant VMs in one datacenter will be able to communicate with tenant VMs on the other datacenter provided they are part of the same BGP EVPN and they are on same subnets.

The dataplane between the tenant VMs themselves and between the tenant VMs towards the DC-Gateway will be over VXLAN Tunnels.

The dataplane between the DC-Gateway to its other WAN-based BGP Peers is transparent to this spec. It is usually MPLS-based EPVPN.

The BGP Control plane between the ODL Controller and the DC-Gateway will be via EVPN RouteType 2 as defined in EVPN_RT2.

The control plane between the DC-Gateway and it other BGP Peers in the WAN is transparent to this spec, but can be EVPN IP-MPLS.

In this use-case:

1. We will have only a single DCGW for WAN connectivity
2. MAC IP prefix exchange between ODL controller and DC-GW (iBGP) using EVPN RT2
3. WAN control plane may use EVPN IP-MPLS for route exchange.
4. On the DC-Gateway, the VRF instance will be configured with two sets of import/export targets. One set of import/export route targets belong to EVPN inside DataCenter (realized using EVPN RT2) and the second set of import/export route target belongs to WAN control plane.
5. EVPN single homing to be used in all RT2 exchanges inside the DataCenter i.e., ESI=0 for all prefixes sent from DataCenter to the DC-Gateway.

Proposed change

The following components of an Openstack-ODL-based solution need to be enhanced to provide intra-subnet and inter-subnet connectivity across DCs using EVPN MAC IP Advertisement (Route Type 2) mechanism (refer EVPN_RT2):

- Openstack Neutron BGPVPN Driver
- OpenDaylight Controller (NetVirt)
- BGP Quagga Stack to support EVPN with RouteType 2 NLRI
- DC-Gateway BGP Neighbour that supports EVPN with RouteType 2 NLRI

The changes required in Openstack Neutron BGPVPN Driver and BGP Quagga Stack are captured in the Solution considerations section down below.

Pipeline changes

INTRA DC

Intra Subnet, Local DPN: VMs on the same subnet, same VPN, same DPN

There are no explicit pipeline changes for this use-case.

Intra Subnet, Remote DPN: VMs on two different DPNs, both VMs on the same subnet and same VPN

There are no explicit pipeline changes for this use-case.

Inter Subnet, Local DPN: VMs on different subnet, same VPN, same DPN

There are no explicit pipeline changes for this use-case.

Inter Subnet, Local DPN: VMs on different subnet, same VPN, same DPN

There are no explicit pipeline changes for this use-case.

INTER DC

Intra subnet Traffic from DC-Gateway to Local DPN

Classifier table (0) =>
Dispatcher table (17) `match: tunnel-type=vxlan` =>
L2VNI_EXTERNAL_TUNNEL_DEMUX_TABLE (24) => `match tunnel-id=l2vni, set elan-tag`
ELAN DMAC table (51) `match: elan-tag=vxlan-net-tag,dst-mac=vm2-mac set reg6=vm-lport-tag` =>
Egress table (220) `match: reg6=vm-lport-tag output to vm port`

Intra subnet Traffic from Local DPN to DC-Gateway

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) =>
Dispatcher table (17) `l2vpn service: set elan-tag=vxlan-net-tag` =>
ELAN base table (48) =>
ELAN SMAC table (50) `match: elan-tag=vxlan-net-tag,src-mac=vm1-mac` =>
ELAN DMAC table (51) `match: elan-tag=vxlan-net-tag,dst-mac=external-vm-mac set tun-id=vxlan-net-tag group=next-hop-group`
Next Hop Group `bucket0 :set reg6=tunnel-lport-tag bucket1 :set reg6=tunnel2-lport-tag`
Egress table (220) `match: reg6=tunnel2-lport-tag` output to `tunnel2`

Inter subnet Traffic from Local DPN to DC-Gateway (Symmetric IRB)

Classifier Table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set tun-id=l3vni output to nexthopgroup` =>
NextHopGroup: `set-eth-dst router-gw-vm, reg6=tunnel-lport-tag` =>
Lport Egress Table (220) `Output to tunnel port`

Inter subnet Traffic from DC-Gateway to Local DPN (Symmetric IRB)

Classifier table (0) =>
Dispatcher table (17) `match: tunnel-type=vxlan` =>
L3VNI_EXTERNAL_TUNNEL_DEMUX_TABLE (23) => `match tunnel-id=l3vni, set l3vpn-id` =>
L3 Gateway MAC Table (19) => `match dst-mac=vpn-subnet-gateway-mac-address` =>
FIB table (21) `match: l3vpn-tag=l3vpn-id,dst-ip=vm2-ip set reg6=vm-lport-tag goto=local-nexthop-group` =>
local nexthop group `set dst-mac=vm2-mac table=220` =>
Egress table (220) `match: reg6=vm-lport-tag output to vm port`

Inter subnet Traffic from Local DPN to DC-Gateway (ASymmetric IRB)

Classifier Table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set tun-id=l2vni output to nexthopgroup` =>
NextHopGroup: `set-eth-dst dst-vm-mac, reg6=tunnel-lport-tag` =>

Lport Egress Table (220) `Output to tunnel port`

Intra subnet Traffic from DC-Gateway to Local DPN (ASymmetric IRB)

Classifier table (0) =>
Dispatcher table (17) `match: tunnel-type=vxlan` =>
L2VNI_EXTERNAL_TUNNEL_DEMUX_TABLE (24) => `match tunnel-id=l2vni, set elan-tag`
ELAN DMAC table (51) `match: elan-tag=vxlan-net-tag,dst-mac=vm2-mac set reg6=vm-lport-tag` =>
Egress table (220) `match: reg6=vm-lport-tag output to vm port`

ARP Pipeline changes

Local DPN: VMs on the same subnet, same DPN

a. Introducing a new Table aka ELAN_ARP_SERVICE_TABLE (Table 81). This table will be the first table in elan pipeline.

Classifier table (0) =>
Dispatcher table (17) `elan service: set elan-id=vxlan-net-tag` =>
Arp Service table (81) => `match: arp-op=req, dst-ip=vm-ip, ela-id=vxlan-net-tag inline arp reply`

Intra Subnet, Local DPN: VMs on the same subnet, on remote DC

Classifier table (0) =>
Dispatcher table (17) `elan service: set elan-id=vxlan-net-tag` =>
Arp Service table (81) => `match: arp-op=req, dst-ip=vm-ip, ela-id=vxlan-net-tag inline arp reply`

Yang changes

Changes will be needed in `l3vpn.yang` , `odl-l3vpn.yang` , `odl-fib.yang` and `neutronvpn.yang` to start supporting EVPN functionality.

ODL-L3VPN YANG changes

A new container evpn-rd-to-networks is added This holds the rd to networks mapping This will be useful to extract in which elan the received RT2 route can be injected into.

Listing 1.26: odl-l3vpn.yang

```
container evpn-rd-to-networks {
    config false;
    description "Holds the networks to which given evpn is attached to";
    list evpn-rd-to-network {
       key rd;
       leaf rd {
```

```
          type string;
        }
        list evpn-networks {
         key network-id;
         leaf network-id {
           type string;
         }
        }
    }
}
```

ODL-FIB YANG changes

A new field macVrfEntries is added to the container `fibEntries` This holds the RT2 routes received for the given rd

Listing 1.27: odl-fib.yang

```
grouping vrfEntryBase {
    list vrfEntry{
        key  "destPrefix";
        leaf destPrefix {
            type string;
            mandatory true;
        }
        leaf origin {
            type string;
            mandatory true;
        }
        leaf encap-type {
           type enumeration {
              enum mplsgre {
                 value "0";
                 description "MPLSOverGRE";
              }
              enum vxlan {
                 value "1";
                 description "VNI";
              }
           }
           default "mplsgre";
        }
        leaf l3vni {
           type uint32;
        }
        list route-paths {
            key "nexthop-address";
            leaf nexthop-address {
                type string;
            }
            leaf label {
                type uint32;
            }
            leaf gateway_mac_address {
                type string;
            }
```

```
        }
    }
}

grouping vrfEntries{
    list vrfEntry{
        key  "destPrefix";
        uses vrfEntryBase;
    }
}

grouping macVrfEntries{
    list MacVrfEntry {
        key  "mac_address";
        uses vrfEntryBase;
        leaf l2vni {
           type uint32;
        }
    }
}

container fibEntries {
     config true;
     list vrfTables {
        key "routeDistinguisher";
        leaf routeDistinguisher {type string;}
        uses vrfEntries;
        uses macVrfEntries;//new field
     }
     container ipv4Table{
        uses ipv4Entries;
     }
}
```

NEUTRONVPN YANG changes

A new rpc `createEVPN` is added Existing rpc associateNetworks is reused to attach a network to EVPN assuming uuid of L3VPN and EVPN does not collide with each other.

Listing 1.28: neutronvpn.yang

```
rpc createEVPN {
    description "Create one or more EVPN(s)";
    input {
        list evpn {
            uses evpn-instance;
        }
    }
    output {
        leaf-list response {
            type    string;
            description "Status response for createVPN RPC";
        }
    }
}
```

```
rpc deleteEVPN{
    description "delete EVPNs for specified Id list";
    input {
        leaf-list id {
            type    yang:uuid;
            description "evpn-id";
        }
    }
    output {
        leaf-list response {
            type    string;
            description "Status response for deleteEVPN RPC";
        }
    }
}

grouping evpn-instance {

    leaf id {
        mandatory "true";
        type    yang:uuid;
        description "evpn-id";
    }

    leaf name {
      type    string;
      description "EVPN name";
    }

    leaf tenant-id {
        type    yang:uuid;
        description "The UUID of the tenant that will own the subnet.";
    }

    leaf-list route-distinguisher {
        type string;
        description
        "configures a route distinguisher (RD) for the EVPN instance.
         Format is ASN:nn or IP-address:nn.";
    }

    leaf-list import-RT {
        type string;
        description
        "configures a list of import route target.
         Format is ASN:nn or IP-address:nn.";
    }

    leaf-list export-RT{
        type string;
        description
        "configures a list of export route targets.
         Format is ASN:nn or IP-address:nn.";
    }

    leaf l2vni {
       type uint32;
    }
```

```
}
```

ELAN YANG changes

Existing container elan-instances is augmented with evpn information.

A new list `external-teps` is added to elan container. This captures the broadcast domain of the given network/elan. When the first RT2 route is received from the dc gw, it's tep ip is added to the elan to which this RT2 route belongs to.

Listing 1.29: elan.yang

```
augment "/elan:elan-instances/elan:elan-instance" {
    ext:augment-identifier "evpn";
    leaf evpn-name {
        type string;
    }
    leaf l3vpn-name {
        type string;
    }
}

container elan-instances {
    list elan-instance {
        key "elan-instance-name";
        leaf elan-instance-name {
            type string;
        }
        //omitted other existing fields
        list external-teps {
            key tep-ip;
            leaf tep-ip {
                type inet:ip-address;
            }
        }
    }
}

container elan-interfaces {
    list elan-interface  {
        key "name";
        leaf name {
            type leafref {
                path "/if:interfaces/if:interface/if:name";
            }
        }
        leaf elan-instance-name {
            mandatory true;
            type string;
        }
        list static-mac-entries {
            key "mac";
            leaf mac {
                type yang:phys-address;
            }
            leaf prefix {//new field
```

```
                mandatory false;
                type inet:ip-address;
            }
        }
    }
}

grouping forwarding-entries {
    list mac-entry {
      key "mac-address";
      leaf mac-address {
          type yang:phys-address;
      }
      leaf interface {
         type leafref {
             path "/if:interfaces/if:interface/if:name";
         }
      }
      leaf controllerLearnedForwardingEntryTimestamp {
        type uint64;
      }
      leaf isStaticAddress {
        type boolean;
      }
      leaf prefix {//new field
        mandatory false;
        type inet:ip-address;
      }
    }
}
```

Solution considerations

Proposed change in Openstack Neutron BGPVPN Driver

The Openstack Neutron BGPVPN's ODL driver in Newton release is changed (mitaka release), so that it is able to relay the configured L2 BGPVPNs, to the OpenDaylight Controller.

The Newton changes for the BGPVPN Driver has merged and is here: https://review.openstack.org/#/c/370547/

Proposed change in BGP Quagga Stack

The BGP Quagga Stack is a component that interfaces with ODL Controller to enable ODL Controller itself to become a BGP Peer. This BGP Quagga Stack need to be enhanced so that it is able to embrace EVPN with Route Type 5 on the following two interfaces:

- Thrift Interface where ODL pushes routes to BGP Quagga Stack
- Route exchanges from BGP Quagga Stack to other BGP Neighbors (including DC-GW).

Proposed change in OpenDaylight-specific features

The following components within OpenDaylight Controller needs to be enhanced:

- NeutronvpnManager
- VPN Engine (VPN Manager)
- ELAN Manager
- FIB Manager
- BGP Manager

Reboot Scenarios

This feature support all the following Reboot Scenarios for EVPN:

- Entire Cluster Reboot
- Leader PL reboot
- Candidate PL reboot
- OVS Datapath reboots
- Multiple PL reboots
- Multiple Cluster reboots
- Multiple reboots of the same OVS Datapath.
- Openstack Controller reboots

Configuration impact

The following parameters have been initially made available as configurable for EVPN. These configurations can be made via the RESTful interface:

1.Multi-homing-mode – For multi-homing use cases where redundant DCGWs are used ODL can be configured with 'none', 'all-active' or 'single-active' multi-homing mode. Default will be 'none'.

2.IRB-mode – Depending upon the support on DCGW, ODL can be configured with either 'Symmetric' or 'Asymmetric' IRB mode. Default is 'Symmetric'.

There is another important parameter though it won't be configurable:

MAC Address Prefix for EVPN – This MAC Address prefix represents the MAC Address prefix that will be hard-coded and that MACAddress will be used as the gateway mac address if it is not supplied from Openstack. This will usually be the case when networks are associated to an L3VPN with no gateway port yet configured in Openstack for such networks.

Clustering considerations

The feature should operate in ODL Clustered environment reliably.

Other Infra considerations

N.A.

Security considerations

N.A.

Scale and Performance Impact

Not covered by this Design Document.

Targeted Release

Carbon.

Alternatives

Alternatives considered and why they were not selected.

Usage

Features to Install

This feature can be used by installing odl-netvirt-openstack. This feature doesn't add any new karaf feature.

REST API

A new rpc is added to create and delete evpn:

```
{'input': {
    'evpn': [
        {'name': 'EVPN1',
         'export-RT': ['50:2'],
         'route-distinguisher': ['50:2'],
         'import-RT': ['50:2'],
         'id': '4ae8cd92-48ca-49b5-94e1-b2921a260007',
         'l2vni': '200',
         'tenant-id': 'a565b3ed854247f795c0840b0481c699'
}]}}
```

There is no change in the REST API for associating networks to the EVPN.

On the Openstack-side configuration, the vni_ranges configured in Openstack Neutron ml2_conf.ini should not overlap with the L3VNI provided in the ODL RESTful API. In an inter-DC case, where both the DCs are managed by two different Openstack Controller Instances, the workflow will be to do the following:

1. Configure the DC-GW2 facing OSC2 (Openstack) and DC-GW1 facing OSC1 with the same BGP configuration parameters.
2. On first Openstack Controller (OSC1) create an L3VPN1 with RD1 and L3VNI1
3. On first Openstack Controller (OSC1) create an EVPN1 with RD2 and L2VNI1
4. Create a network Net1 and Associate that Network Net1 to L3VPN1

5. Create a network Net1 and Associate that Network Net1 to EVPN1
6. On second Openstack Controller (OSC2) create an L3VPN2 with RD1 with L3VNI1
7. On second Openstack Controller (OSC2) create an EVPN2 with RD2 with L2VNI1
8. Create a network Net2 on OSC2 with same cidr as the first one with a different allocation pool and associate that Network Net2 to L3VPN2.
9. Associate that Network Net2 to EVPN2.
10. Spin-off VM1 on Net1 in OSC1.
11. Spin-off VM2 on Net2 in OSC2.
12. Now VM1 and VM2 should be able to communicate.

Implementation

Assignee(s)

Primary assignee: Vyshakh Krishnan C H <vyshakh.krishnan.c.h@ericsson.com>

Yugandhar Reddy Kaku <yugandhar.reddy.kaku@ericsson.com>

Riyazahmed D Talikoti <riyazahmed.d.talikoti@ericsson.com>

Other contributors: K.V Suneelu Verma <k.v.suneelu.verma@ericsson.com>

Work Items

Trello card details https://trello.com/c/PysPZscm/150-evpn-evpn-rt2.

Dependencies

Requires a DC-GW that is supporting EVPN RT2 on BGP Control plane.

Testing

Capture details of testing that will need to be added.

Unit Tests

Appropriate UTs will be added for the new code coming in once framework is in place.

Integration Tests

There won't be any Integration tests provided for this feature.

CSIT

CSIT will be enhanced to cover this feature by providing new CSIT tests.

Documentation Impact

This will require changes to User Guide and Developer Guide.

References

[1] EVPN_RT5

[2] Network Virtualization using EVPN

[3] Integrated Routing and Bridging in EVPN

[4] VXLAN DCI using EVPN

[5] BGP MPLS-Based Ethernet VPN

[6] Trello card details

Table of Contents

- *References*

Support of VXLAN based connectivity across Datacenters

https://git.opendaylight.org/gerrit/#/q/topic:EVPN_RT5

Enable realization of L3 connectivity over VXLAN tunnels using L3 BGPVPNs, internally taking advantage of EVPN as the BGP Control Plane mechanism.

Problem description

OpenDaylight NetVirt service today supports VLAN-based, VXLAN-based connectivity and MPLSOverGRE-based overlays.

In this VXLAN-based underlay is supported only for traffic within the DataCenter. For all the traffic that need to go via the DC-Gateway the only supported underlay is MPLSOverGRE.

Though there is a way to provision an external VXLAN tunnel via the ITM service in Genius, the BGPVPN service in NetVirt does not have the ability to take advantage of such a tunnel to provide inter-DC connectivity.

This spec attempts to enhance the BGPVPN service (runs on top of the current L3 Forwarding service) in NetVirt to embrace inter-DC L3 connectivity over external VXLAN tunnels.

In scope

The scope primarily includes providing ability to support Inter-subnet connectivity across DataCenters over VXLAN tunnels by modeling a new type of L3VPN which will realize this connectivity using EVPN BGP Control plane semantics.

When we mention that we are using EVPN BGP Control plane, this spec proposes using the RouteType 5 explained in EVPN_RT5 as the primary means to provision the control plane en enable inter-DC connectivity over external VXLAN tunnels.

This new type of L3VPN will also inclusively support:

- Intra-subnet connectivity within a DataCenter over VXLAN tunnels.
- Inter-subnet connectivity within a DataCenter over VXLAN tunnels.

Out of scope

- Does not cover providing VXLAN connectivity between hypervisors (with OVS Datapath) and Top-Of-Rack switches that might be positioned within such DataCenters.
- Does not cover providing intra-subnet connectivity across DCs.

Both the points above will be covered by another spec that will be Phase 2 of realizing intra-subnet inter-DC connectivity.

Use Cases

The following high level use-cases will be realized by the implementation of this Spec.

DataCenter access from a WAN-client via DC-Gateway (Single Homing)

This use case involves communication within the DataCenter by tenant VMs and also communication between the tenant VMs to a remote WAN-based client via DC-Gateway. The dataplane between the tenant VMs themselves and between the tenant VMs towards the DC-Gateway will be over VXLAN Tunnels.

The dataplane between the DC-Gateway to its other WAN-based BGP Peers is transparent to this spec. It is usually MPLS-based IPVPN.

The BGP Control plane between the ODL Controller and the DC-Gateway will be via EVPN RouteType 5 as defined in EVPN_RT5.

The control plane between the DC-Gateway and it other BGP Peers in the WAN is transparent to this spec, but can be IP-MPLS.

In this use-case:

1. We will have only a single DCGW for WAN connectivity
2. IP prefix exchange between ODL controller and DC-GW (iBGP) using EVPN RT5
3. WAN control plane will use L3VPN IP-MPLS route exchange.
4. On the DC-Gateway, the VRF instance will be configured with two sets of import/export targets. One set of import/export route targets belong to L3VPN inside DataCenter (realized using EVPN RT5) and the second set of import/export route target belongs to WAN control plane.
5. EVPN single homing to be used in all RT5 exchanges inside the DataCenter i.e., ESI=0 for all prefixes sent from DataCenter to the DC-Gateway.
6. Inter AS option B is used at DCGW, route regeneration at DCGW

Datacenter access from another Datacenter over WAN via respective DC-Gateways (L3 DCI)

This use-case involves providing inter-subnet connectivity between two DataCenters. Tenant VMs in one datacenter will be able to communicate with tenant VMs on the other datacenter provided they are part of the same L3VPN and they are on different subnets.

Both the Datacenters can be managed by different ODL Controllers, but the L3VPN configured on both ODL Controllers will use identical RDs and RTs.

Proposed change

The following components of an Openstack-ODL-based solution need to be enhanced to provide intra-subnet and inter-subnet connectivity across DCs using EVPN IP Prefix Advertisement (Route Type 5) mechanism (refer EVPN_RT5):

- Openstack Neutron BGPVPN Driver
- OpenDaylight Controller (NetVirt)
- BGP Quagga Stack to support EVPN with RouteType 5 NLRI
- DC-Gateway BGP Neighbour that supports EVPN with RouteType 5 NLRI

The changes required in Openstack Neutron BGPVPN Driver and BGP Quagga Stack are captured in the Solution considerations section down below.

Pipeline changes

For both the use-cases above, we have put together the required pipeline changes here. For ease of understanding, we have made subsections that talk about Intra-DC traffic and Inter-DC traffic.

INTRA DC

Intra Subnet, Local DPN: VMs on the same subnet, same VPN, same DPN

There are no explicit pipeline changes for this use-case. However the tables that a packet will traverse through is shown below for understanding purposes.

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `tablemiss: goto_table=17` =>
Lport Dispatcher Table (17) `elan service: set elan-id=elan-tag` =>
ELAN Source MAC Table (50) `match: elan-id=elan-tag, src-mac=source-vm-mac` =>
ELAN Destination MAC Table (51) `match: elan-id=elan-tag, dst-mac=dst-vm-mac set output to port-of-dst-vm`

Intra Subnet, Remote DPN: VMs on two different DPNs, both VMs on the same subnet and same VPN

There are no explicit pipeline changes for this use-case. However the tables that a packet will traverse through is shown below for understanding purposes.

VM sourcing the traffic (Ingress DPN)

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `l3vpn service: tablemiss: goto_table=17` =>
Lport Dispatcher Table (17) `elan service: set elan-id=elan-tag` =>
ELAN Source MAC Table (50) `match: elan-id=elan-tag, src-mac=source-vm-mac` =>
ELAN Destination MAC Table (51) `match: elan-id=elan-tag, dst-mac=dst-vm-mac set tun-id=dst-vm-lport-tag, output to vxlan-tun-port`

VM receiving the traffic (Egress DPN)

Classifier Table (0) =>
Internal Tunnel Table (36) `match: tun-id=lport-tag set reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

Inter Subnet, Local DPN: VMs on different subnet, same VPN, same DPN

There are no explicit pipeline changes for this use-case. However the tables that a packet will traverse through is shown below for understanding purposes.

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set output to nexthopgroup-dst-vm` =>
NextHopGroup-dst-vm: `set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

```
cookie=0x8000000, table=0, priority=4,in_port=1 actions=write_metadata:0x10000000000/
↪0xffffff0000000001,goto_table:17
cookie=0x8000001, table=17, priority=5,metadata=0x5000010000000000/0xffffff0000000000␣
↪actions=write_metadata:0x60000100000222e0/0xfffffffffffffffe,goto_table:19
cookie=0x8000009, table=19, priority=20,metadata=0x222e0/0xffffffff,dl_
↪dst=de:ad:be:ef:00:01 actions=goto_table:21
cookie=0x8000003, table=21, priority=42,ip,metadata=0x222e0/0xffffffff,nw_dst=10.0.0.
↪2 actions=apply_actions(group:150001)
```

Inter Subnet, Remote DPN: VMs on two different DPNs, both VMs on different subnet, but same VPN

For this use-case there is a change in the remote flow rule to L3 Forward the traffic to the remote VM. The flow-rule will use the LPortTag as the vxlan-tunnel-id, in addition to setting the destination mac address of the remote destination vm.

VM sourcing the traffic (Ingress DPN)

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set eth-dst-mac=dst-vm-mac, tun-id=dst-vm-lport-tag, output to vxlan-tun-port`

```
cookie=0x8000000, table=0, priority=4,in_port=1 actions=write_metadata:0x10000000000/
↪0xffffff0000000001,goto_table:17
cookie=0x8000001, table=17, priority=5,metadata=0x5000010000000000/0xffffff0000000000␣
↪actions=write_metadata:0x60000100000222e0/0xfffffffffffffffe,goto_table:19
cookie=0x8000009, table=19, priority=20,metadata=0x222e0/0xffffffff,dl_
↪dst=de:ad:be:ef:00:01 actions=goto_table:21
cookie=0x8000003, table=21, priority=42,ip,metadata=0x222e0/0xffffffff,nw_dst=10.0.0.
↪2 actions=apply_actions(group:150001)
cookie=0x8000003, table=21, priority=42,ip,metadata=0x222e0/0xffffffff,nw_dst=10.0.0.
↪3 actions=apply_actions(set_field:fa:16:3e:f8:59:af->eth_dst,set_field:0x2->tun_id,
↪output:2)
```

As you can notice 0x2 set in the above flow-rule as tunnel-id is the LPortTag assigned to VM holding IP Address 10.0.0.3.

VM receiving the traffic (Egress DPN)

Classifier Table (0) =>
Internal Tunnel Table (36) `match: tun-id=lport-tag set reg6=lport-tag-dst-vm` =>
Lport Egress Table (220) `Output to dst vm port`

```
cookie=0x8000001, table=0, priority=5,in_port=2 actions=write_metadata:0x40000000001/
↪0xffffff0000000001,goto_table:36
cookie=0x9000001, table=36, priority=5,tun_id=0x2 actions=load:0x400->NXM_NX_REG6[],
↪resubmit(,220)
```

As you notice, 0x2 tunnel-id match in the above flow-rule in INTERNAL_TUNNEL_TABLE (Table 36), is the LPort-Tag assigned to VM holding IP Address 10.0.0.3.

INTER DC

Intra Subnet

Not supported in this Phase

Inter Subnet

For this use-case we are doing a couple of pipeline changes:

a. Introducing a new Table aka L3VNI_EXTERNAL_TUNNEL_DEMUX_TABLE (Table 23). **L3VNI_EXTERNAL_TUNNEL_DEMUX_TABLE (Table 23)** - This table is a new table in the L3VPN pipeline and will be responsible only to process VXLAN packets coming from External VXLAN tunnels.

The packets coming from External VXLAN Tunnels (note: not Internal VXLAN Tunnels), would be directly punted to this new table from the CLASSIFIER TABLE (Table 0) itself. Today when multiple services bind to a tunnel port on GENIUS, the service with highest priority binds directly to Table 0 entry for the tunnel port. So such a service should make sure to provide a fallback to Dispatcher Table so that subsequent service interested in that tunnel traffic would be given the chance.

The new table L3VNI_EXTERNAL_TUNNEL_DEMUX_TABLE will have flows to match on VXLAN VNIs that are L3VNIs. On a match, their action is to fill the metadata with the VPNID, so that further tables in the L3VPN pipeline would be able to continue and operate with the VPNID metadata seamlessly. After filling the metadata, the packets are resubmitted from this new table to the L3_GW_MAC_TABLE (Table 19). The TableMiss in L3VNI_EXTERNAL_TUNNEL_DEMUX_TABLE will resubmit the packet to LPORT_DISPATCHER_TABLE to enable next service if any to process the packet ingressing from the external VXLAN tunnel.

b. For all packets going from VMs within the DC, towards the external gateway device via the External VXLAN Tunnel, we are setting the VXLAN Tunnel ID to the L3VNI value of VPNInstance to which the VM belongs to.

Traffic from DC-Gateway to Local DPN (SYMMETRIC IRB)

Classifier Table (0) =>
L3VNI External Tunnel Demux Table (23) `match: tun-id=l3vni set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set output to nexthopgroup-dst-vm` =>
NextHopGroup-dst-vm: `set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

```
cookie=0x8000001, table=0, priority=5,in_port=9 actions=write_metadata:0x70000000001/
↪0x1fffff0000000001,goto_table:23
cookie=0x8000001, table=19, priority=20,metadata=0x222e0/0xffffffff,dl_
↪dst=de:ad:be:ef:00:06 actions=goto_table:21
cookie=0x8000001, table=23, priority=5,tun_id=0x16 actions= write_metadata:0x222e0/
↪0xfffffffe,resubmit(19)
cookie=0x8000001, table=23, priority=0,resubmit(17)
cookie=0x8000003, table=21, priority=42,ip,metadata=0x222e0/0xfffffffe,nw_dst=10.0.0.
↪2 actions=apply_actions(group:150001)
cookie=0x8000003, table=21, priority=42,ip,metadata=0x222e0/0xfffffffe,nw_dst=10.0.0.
↪3 actions=apply_actions(set_field:fa:16:3e:f8:59:af->eth_dst,set_field:0x2->tun_id,
↪output:2)
```

In the above flow rules, Table 23 is the new L3VNI_EXTERNAL_TUNNEL_DEMUX_TABLE. The in_port=9 reprsents an external VXLAN Tunnel port.

Traffic from Local DPN to DC-Gateway (SYMMETRIC IRB)

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=ext-ip-address set eth-dst-mac=dst-mac-address, tun-id=l3vni, output to ext-vxlan-tun-port`

```
cookie=0x7000001, table=0, priority=5,in_port=8, actions=write_metadata:0x60000000001/
↪0x1fffff0000000001,goto_table:17
cookie=0x7000001, table=17, priority=5,metadata=0x60000000001/0x1fffff0000000001␣
↪actions=goto_table:19
cookie=0x7000001, table=19, priority=20,metadata=0x222e0/0xffffffff,dl_
↪dst=de:ad:be:ef:00:06 actions=goto_table:21
cookie=0x7000001, table=23, priority=5,tun_id=0x16 actions= write_metadata:0x222e0/
↪0xfffffffe,resubmit(19)
cookie=0x7000001, table=23, priority=0,resubmit(17)
cookie=0x7000003, table=21, priority=42,ip,metadata=0x222e0/0xfffffffe,nw_dst=10.0.0.
↪2 actions=apply_actions(group:150001)
cookie=0x7000003, table=21, priority=42,ip,metadata=0x222e0/0xfffffffe,nw_dst=10.0.0.
↪3 actions=apply_actions(set_field:fa:16:3e:f8:59:af->eth_dst,set_field:0x2->tun_id,
↪output:2)
```

SNAT pipeline (Access to External Network Access over VXLAN)

SNAT Traffic from Local DPN to External IP (assuming this DPN is NAPT Switch)

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id` =>
PSNAT Table (26) `match: vpn-id=l3vpn-id` =>
Outbound NAPT Table (46) `match: nw-src=vm-ip,port=int-port set src-ip=router-gateway-ip,src-mac=external-router-gateway-mac-address, vpn-id=external-vpn-id,port=ext-port` =>
NAPT PFIB Table (47) `match: vpn-id=external-vpn-id` =>
L3 FIB Table (21) `match: vpn-id=external-vpn-id nw-dst=external-entity-ip set eth-dst=external-entity-mac tun-id=external-l3vni, output to ext-vxlan-tun-port`

SNAT Reverse Traffic from External IP to Local DPN (assuming this DPN is NAPT Switch)

Classifier Table (0) =>
L3VNI External Tunnel Demux Table (23) `match: tun-id=external-l3vni set vpn-id=external-vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=external-vpn-id, dst-mac=external-router-gateway-mac-address` =>
Inbound NAPT Table (44) `match: vpn-id=external-vpn-id nw-dst=router-gateway-ip port=ext-port set vpn-id=l3vpn-id, dst-ip=vm-ip`
NAPT PFIB Table (47) `match: vpn-id=l3vpn-id` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set output to nexthopgroup-dst-vm` =>
NextHopGroup-dst-vm: `set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

DNAT pipeline (Access from External Network over VXLAN)

DNAT Traffic from External IP to Local DPN

Classifier Table (0) =>
L3VNI External Tunnel Demux Table (23) `match: tun-id=external-l3vni set vpn-id=external-vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=external-vpn-id, eth-dst=floating-ip-dst-vm-mac-address` =>
PDNAT Table (25) `match: nw-dst=floating-ip,eth-dst=floating-ip-dst-vm-mac-address set ip-dst=dst-vm-ip, vpn-id=l3vpn-id` =>
DNAT Table (27) `match: vpn-id=l3vpn-id,nw-dst=dst-vm-ip` =>

L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip set output to nexthopgroup-dst-vm` =>
NextHopGroup-dst-vm: `set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

DNAT Reverse Traffic from Local DPN to External IP

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id` =>
PSNAT Table (26) `match: vpn-id=l3vpn-id nw-src=src-vm-ip set ip-src=floating-ip-src-vm, vpn-id=external-vpn-id` =>
SNAT Table (28) `match: vpn-id=external-vpn-id nw-src=floating-ip-src-vm set eth-src=floating-ip-src-vm-mac-address` =>
L3 FIB Table (21) `match: vpn-id=external-vpn-id nw-dst=external-floating-ip set eth-dst=external-mac-address tun-id=external-l3vni, output to ext-vxlan-tun-port`

DNAT to DNAT Traffic (Intra DC)

1. FIP VM to FIP VM on Different Hypervisor

DPN1:

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id` =>
PSNAT Table (26) `match: vpn-id=l3vpn-id nw-src=src-vm-ip set ip-src=floating-ip-src-vm, vpn-id=external-vpn-id` =>
SNAT Table (28) `match: vpn-id=external-vpn-id nw-src=floating-ip-src-vm set eth-src=floating-ip-src-vm-mac-address` =>
L3 FIB Table (21) `match: vpn-id=external-vpn-id nw-dst=destination-floating-ip set eth-dst=floating-ip-dst-vm-mac-address tun-id=external-l3vni, output to vxlan-tun-port`

DPN2:

Classifier Table (0) =>
Internal Tunnel Table (36) `match: tun-id= external-l3vni` =>
PDNAT Table (25) `match: nw-dst=floating-ip eth-dst=floating-ip-dst-vm-mac-address set ip-dst=dst-vm-ip, vpn-id=l3vpn-id` =>
DNAT Table (27) `match: vpn-id=l3vpn-id,nw-dst=dst-vm-ip` =>

L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip set output to nexthopgroup-dst-vm` =>
NextHopGroup-dst-vm: `set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

In the above flow rules `INTERNAL_TUNNEL_TABLE` (table=36) will take the packet to the `PDNAT_TABLE` (table 25) for an exact match with floating-ip and floating-ip-dst-vm-mac-address in `PDNAT_TABLE`.

In case of a successful floating-ip and floating-ip-dst-vm-mac-address match, `PDNAT_TABLE` will set IP destination as VM IP and VPN ID as internal l3 VPN ID then it will pointing to `DNAT_TABLE` (table=27)

In case of no match, the packet will be redirected to the SNAT pipeline towards the `INBOUND_NAPT_TABLE` (table=44). This is the use-case where `DPN2` also acts as the NAPT DPN.

In summary, on an given NAPT switch, if both DNAT and SNAT are configured, the incoming traffic will first be sent to the `PDNAT_TABLE` and if there is no FIP and FIP Mac match found, then it will be forwarded to `INBOUND_NAPT_TABLE` for SNAT translation. As part of the response, the `external-l3vni` will be used as `tun_id` to reach floating IP VM on `DPN1`.

2. FIP VM to FIP VM on same Hypervisor

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id` =>
PSNAT Table (26) `match: vpn-id=l3vpn-id nw-src=src-vm-ip set ip-src=floating-ip-src-vm, vpn-id=external-vpn-id` =>
SNAT Table (28) `match: vpn-id=external-vpn-id nw-src=floating-ip-src-vm set eth-src=floating-ip-src-vm-mac-address` =>
L3 FIB Table (21) `match: vpn-id=external-vpn-id nw-dst=destination-floating-ip set eth-dst= floating-ip-dst-vm-mac-address` =>
PDNAT Table (25) `match: nw-dst=floating-ip eth-dst=floating-ip-dst-vm-mac-address set ip-dst=dst-vm-ip, vpn-id=l3vpn-id` =>
DNAT Table (27) `match: vpn-id=l3vpn-id,nw-dst=dst-vm-ip` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip set output to nexthopgroup-dst-vm` =>
NextHopGroup-dst-vm: `set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

SNAT to DNAT Traffic (Intra DC)

SNAT Hypervisor:

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id` =>
PSNAT Table (26) `match: vpn-id=l3vpn-id` =>
Outbound NAPT Table (46) `match: nw-src=vm-ip,port=int-port set src-ip=router-gateway-ip,src-mac=external-router-gateway-mac-address,`

vpn-id=external-vpn-id,port=ext-port
=>
NAPT PFIB Table (47) match: vpn-id=external-vpn-id =>
L3 FIB Table (21) match: vpn-id=external-vpn-id nw-dst=destination-floating-ip set eth-dst=floating-ip-dst-vm-mac-address tun-id=external-l3vni, output to vxlan-tun-port

DNAT Hypervisor:

Classifier Table (0) =>
Internal Tunnel Table (36) match: tun-id= external-l3vni =>
PDNAT Table (25) ‘‘match: nw-dst=floating-ip eth-dst= floating-ip-dst-vm-mac-address set ip-dst=dst-vm-ip, vpn-id=l3vpn-id‘‘=>
DNAT Table (27) match: vpn-id=l3vpn-id,nw-dst=dst-vm-ip =>
L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip set output to nexthopgroup-dst-vm =>
NextHopGroup-dst-vm: set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag =>
Lport Egress Table (220) Output to dst vm port

Non-NAPT to NAPT Forward Traffic (Intra DC)

Non-NAPT Hypervisor:

Classifier Table (0) =>
Lport Dispatcher Table (17) l3vpn service: set vpn-id=l3vpn-id =>
L3 Gateway MAC Table (19) match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address =>
L3 FIB Table (21) match: vpn-id=l3vpn-id =>
PSNAT Table (26) match: vpn-id=l3vpn-id set tun-id=router-lport-tag,group =>
group: output to NAPT vxlan-tun-port

NAPT Hypervisor:

Classifier Table (0) =>
Internal Tunnel Table (36) match: tun-id=router-lport-tag =>
Outbound NAPT Table (46) match: nw-src=vm-ip,port=int-port set src-ip=router-gateway-ip,src-mac=external-router-gateway-mac-address, vpn-id=external-vpn-id,port=ext-port
=>
NAPT PFIB Table (47) match: vpn-id=external-vpn-id =>
L3 FIB Table (21) match: vpn-id=external-vpn-id nw-dst=external-entity-ip set eth-dst=external-entity-mac tun-id=external-l3vni, output to ext-vxlan-tun-port

For forwarding the traffic from Non-NAPT to NAPT DPN the tun-id will be setting with “router-lport-tag” which will be carved out per router.

NAPT to Non-NAPT Reverse Traffic (Intra DC)

NAPT Hypervisor:

Classifier Table (0) =>
L3VNI External Tunnel Demux Table (23) `match: tun-id=external-l3vni set vpn-id=external-vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=external-vpn-id, dst-mac=external-router-gateway-mac-address` =>
Inbound NAPT Table (44) `match: vpn-id=external-vpn-id nw-dst=router-gateway-ip port=ext-port set vpn-id=l3vpn-id, dst-ip=vm-ip` =>
NAPT PFIB Table (47) `match: vpn-id=l3vpn-id` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set eth-dst-mac=dst-vm-mac, tun-id=dst-vm-lport-tag, output to vxlan-tun-port`

Non-NAPT Hypervisor:

Classifier Table (0) =>
Internal Tunnel Table (36) `match: tun-id=dst-vm-lport-tag` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip set output to nexthopgroup-dst-vm` =>
NextHopGroup-dst-vm: `set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

More details of the NAT pipeline changes are in the NAT Service section of this spec.

Yang changes

Changes will be needed in `l3vpn.yang` , `odl-l3vpn.yang` , `odl-fib.yang` and `neutronvpn.yang` to start supporting EVPN functionality.

L3VPN YANG changes

A new leaf l3vni and a new leaf type will be added to container `vpn-instances`

Listing 1.30: l3vpn.yang

```
leaf type {
          description
          "The type of the VPN Instance.
          ipvpn indicates it is an L3VPN.
          evpn indicates it is EVPN";

          type enumeration {
                enum ipvpn {
                value "0";
                description "L3VPN";
                }
                enum evpn {
```

```
                value "1";
                description "EVPN";
                }
          }
          default "ipvpn";
}

leaf l3vni {
           description
           "The L3 VNI to use for this L3VPN Instance.
           If this attribute is non-zero, it indicates
           this L3VPN will do L3Forwarding over VXLAN.
           If this value is non-zero, and the type field is 'l2',
           it is an error.
           If this value is zero, and the type field is 'l3', it is
           the legacy L3VPN that will do L3Forwarding
           with MPLSoverGRE.
           If this value is zero, and the type field is 'l2', it
           is an EVPN that will provide L2 Connectivity with
           Openstack supplied VNI".

           type uint24;
           mandatory false;
}

The **type** value comes from Openstack BGPVPN ODL Driver based on what type of␣
↪BGPVPN is
orchestrated by the tenant. That same **type** value must be retrieved and stored␣
↪into
VPNInstance model above maintained by NeutronvpnManager.
```

ODL-L3VPN YANG changes

A new leaf l3vni and a new leaf type will be added to container `vpn-instance-op-data`

Listing 1.31: odl-l3vpn.yang

```
leaf type {
          description
          "The type of the VPN Instance.
          ipvpn indicates it is an L3VPN.
          evpn indicates it is EVPN";

          type enumeration {
                enum ipvpn {
                value "0";
                description "L3VPN";
                }
                enum evpn {
                value "1";
                description "EVPN";
                }
          }
          default "ipvpn";
}
```

```
leaf l3vni {
           description
           "The L3 VNI to use for this L3VPN Instance.
           If this attribute is non-zero, it indicates
           this L3VPN will do L3Forwarding over VXLAN.
           If this value is non-zero, and the type field is 'l2',
           it is an error.
           If this value is zero, and the type field is 'l3', it is
           the legacy L3VPN that will do L3Forwarding
           with MPLSoverGRE.
           If this value is zero, and the type field is 'l2', it
           is an EVPN that will provide L2 Connectivity with
           Openstack supplied VNI".

           type uint24;
           mandatory false;
}

For every interface in the cloud that is part of an L3VPN which has an L3VNI setup,␣
↪we should
extract that L3VNI from the config VPNInstance and use that to both program the flows␣
↪as well
as advertise to BGP Neighbour using RouteType 5 BGP Route exchange.
Fundamentally, what we are accomplishing is L3 Connectivity over VXLAN tunnels by␣
↪using the
EVPN RT5 mechanism.
```

ODL-FIB YANG changes

Few new leafs like mac_address , gateway_mac_address , l2vni, l3vni and a leaf encap-type will be added to container `fibEntries`

Listing 1.32: odl-fib.yang

```
leaf encap-type {
   description
   "This flag indicates how to interpret the existing label field.
   A value of mpls indicates that the label will continue to
   be considered as an MPLS Label.
   A value of vxlan indicates that vni should be used to
   advertise to bgp.
   type enumeration {
       enum mplsgre {
           value "0";
           description "MPLSOverGRE";
       }
       enum vxlan {
           value "1";
           description "VNI";
       }
   }
   default "mplsgre";
}

leaf mac_address {
    type string;
```

```
    mandatory false;
}

leaf l3vni {
    type uint24;
    mandatory false;
}

leaf l2vni {
    type uint24;
    mandatory false;
}

leaf gateway_mac_address {
    type string;
    mandatory false;
}
Augment:parent_rd {
    type string;
    mandatory false;
}
```

The encaptype indicates whether an MPLSOverGre or VXLAN encapsulation should be used for this route. If the encapType is MPLSOverGre then the usual label field will carry the MPLS Label to be used in datapath for traffic to/from this VRFEntry IP prefix.

If the encaptype is VXLAN, the VRFEntry implicitly refers that this route is reachable via a VXLAN tunnel. The L3VNI will carry the VRF VNI and there will also be an L2VNI which represents the VNI of the network to which the VRFEntry belongs to.

Based on whether Symmetric IRB (or) Asymmetric IRB is configured to be used by the CSC (see section on Configuration Impact below). If Symmetric IRB is configured, then the L3VNI should be used to program the flows rules. If Asymmetric IRB is configured, then L2VNI should be used in the flow rules.

The mac_address field must be filled for every route in an EVPN. This mac_address field will be used for support intra-DC communication for both inter-subnet and intra-subnet routing.

The gateway_mac_address must always be filled for every route in an EVPN.[AKMA7] [NV8] This gateway_mac_address will be used for all packet exchanges between DC-GW and the DPN in the DC to support L3 based forwarding with Symmetric IRB.

NEUTRONVPN YANG changes

One new leaf l3vni will be added to container grouping `vpn-instance`

Listing 1.33: odl-fib.yang

```
leaf l3vni {
    type uint32;
    mandatory false;
}
```

Solution considerations

Proposed change in Openstack Neutron BGPVPN Driver

The Openstack Neutron BGPVPN's ODL driver in Newton release needs to be changed, so that it is able to relay the configured L2 BGPVPNs, to the OpenDaylight Controller. As of Mitaka release, only L3 BGPVPNs configured in Openstack are being relayed to the OpenDaylight Controller. So in addition to addressing the ODL BGPVPN Driver changes in Newton, we will provide a Mitaka based patch that will integrate into Openstack.

The Newton changes for the BGPVPN Driver has merged and is here: https://review.openstack.org/#/c/370547/

Proposed change in BGP Quagga Stack

The BGP Quagga Stack is a component that interfaces with ODL Controller to enable ODL Controller itself to become a BGP Peer. This BGP Quagga Stack need to be enhanced so that it is able to embrace EVPN with Route Type 5 on the following two interfaces:

- Thrift Interface where ODL pushes routes to BGP Quagga Stack
- Route exchanges from BGP Quagga Stack to other BGP Neighbors (including DC-GW).

Proposed change in OpenDaylight-specific features

The following components within OpenDaylight Controller needs to be enhanced:

- NeutronvpnManager
- VPN Engine (VPN Manager and VPN Interface Manager)
- FIB Manager
- BGP Manager
- VPN SubnetRoute Handler
- NAT Service

Import Export RT support for EVPN

Currently Import/Export logic for L3VPN uses a LabelRouteInfo structure to build information about imported prefixes using MPLS Label as the key. However, this structure cannot be used for EVPN as the L3VNI will be applicable for an entire EVPN Instance instead of the MPLS Label. In lieu of LabelRouteInfo, we will maintain an IPPrefixInfo keyed structure that can be used for facilitating Import/Export of VRFEntries across both EVPNs and L3VPNs.

Listing 1.34: odl-fib.yang

```
list ipprefix-info {

    key "prefix, parent-rd"
    leaf prefix {
        type string;
    }

    leaf parent-rd {
        type string;
    }

    leaf label {
        type uint32;
    }

    leaf dpn-id {
        type uint64;
    }

    leaf-list next-hop-ip-list {
        type string;
    }

    leaf-list vpn-instance-list {
        type string;
    }

    leaf parent-vpnid {
        type uint32;
    }

    leaf vpn-interface-name {
        type string;
    }

    leaf elan-tag {
        type uint32;
    }

    leaf is-subnet-route {
        type boolean;
    }

    leaf encap-type {
        description
        "This flag indicates how to interpret the existing label field.
        A value of mpls indicates that the l3label should be considered as an MPLS
        Label.
        A value of vxlan indicates that l3label should be considered as an VNI.
        type enumeration {
            enum mplsgre {
                value "0";
                description "MPLSOverGRE";
            }
            enum vxlan {
                value "1";
```

```
                description "VNI";
            }
            default "mplsgre";
        }
    }

    leaf l3vni {
        type uint24;
        mandatory false;
    }

    leaf l2vni {
        type uint24;
        mandatory false;
    }

    leaf gateway_mac_address {
        type string;
        mandatory false;
    }
}
```

SubnetRoute support on EVPN

The subnetRoute feature will continue to be supported on EVPN and we will use RT5 to publish subnetRoute entries with either the router-interface-mac-address if available (or) if not available use the pre-defined hardcoded MAC Address described in section Configuration Impact. For both ExtraRoutes and MIPs (invisible IPs) discovered via subnetroute, we will continue to use RT5 to publish those prefixes.[AKMA9] [NV10] On the dataplane, VXLAN packets from the DC-GW will carry the MAC Address of the gateway-ip for the subnet in the inner DMAC.

NAT Service support for EVPN

However, since external network NAT should continue to be supported on VXLAN, making NAT service work on L3VPNs that use VXLAN as the tunnel type becomes imperative.

Existing SNAT/DNAT design assumed internetVpn to be using mplsogre as the connectivity from external network towards DCGW. This needs to be changed such that it can handle even EVPN case with VXLAN connectivity as well.

As of the implementation required for this specification, the workflow will be to create InternetVPN with and associate a single external network to that is of VXLAN Provider Type. The Internet VPN itself will be an L3VPN that will be created via the ODL RESTful API and during creation an L3VNI parameter will be supplied to enable this L3VPN to operate on a VXLAN dataplane. The L3VNI provided to the Internet VPN can be different from the VXLAN segmentation ID associated to the external network.

However, it will be a more viable use-case in the community if we mandate in our workflow that both the L3VNI configured for Internet VPN and the VXLAN segmentation id of the associated external network to the Internet VPN be the same. NAT service can use vpninstance-op-data model to classify the DCGW connectivity for internetVpn.

For the Pipeline changes for NAT Service, please refer to 'Pipeline changes' section.

SNAT to start using Router Gateway MAC, in translated entry in table 46 (Outbound SNAT table) and in table 19 (L3_GW_MAC_Table). Presently Router gateway mac is already stored in odl-nat model in External Routers.

DNAT to start using Floating MAC, in table 28 (SNAT table) and in table 19 (L3_GW_MAC Table). Change in pipeline mainly reverse traffic for SNAT and DNAT so that when packet arrives from DCGW, it goes to 0->38->17->19 and based on Vni and MAC matching, take it back to SNAT or DNAT pipelines.

Also final Fib Entry pointing to DCGW in forward direction also needs modification where we should start using VXLAN's vni, FloatingIPMAC (incase of DNAT) and ExternalGwMacAddress(incase of SNAT) and finally encapsulation type as VXLAN.

For SNAT advertise to BGP happens during external network association to Vpn and during High availability scenarios where you need to re-advertise the NAPT switch. For DNAT we need to advertise when floating IP is associated to the VM. For both SNAT and DNAT this IS mandates that we do only RT5 based advertisement. That RT5 advertisement must carry the external gateway mac address assigned for the respective Router for SNAT case while for DNAT case the RT5 will carry the floating-ip-mac address.

ARP request/response and MIP handling Support for EVPN

Will not support ARP across DCs, as we donot support intra-subnet inter-DC scenarios.

- For intra-subnet intra-DC scenarios, the ARPs will be serviced by existing ELAN pipeline.
- For inter-subnet intra-DC scenarios, the ARPs will be processed by ARP Responder implementation that is already pursued in Carbon.
- For inter-subnet inter-DC scenarios, ARP requests won't be generated by DC-GW. Instead the DC-GW will use 'gateway mac' extended attribute MAC Address information and put that directly into DSTMAC field of Inner MAC Header by the DC-GW for all packets sent to VMs within the DC.
- As quoted, intra-subnet inter-DC scenario is not a supported use-case as per this Implementation Spec.

Tunnel state handling Support

We have to handle both the internal and external tunnel events for L3VPN (with L3VNI) the same way it is handled for current L3VPN.

InterVPNLink support for EVPN

Not supported as this is not a requirement for this Spec.

Supporting VLAN Aware VMs (Trunk and SubPorts)

Not supported as this is not a requirement for this Spec.

VM Mobility with RT5

We will continue to support cold migration of VMs across hypervisors across L3VPNs as supported already in current ODL Carbon Release.

Reboot Scenarios

This feature support all the following Reboot Scenarios for EVPN:

- Entire Cluster Reboot
- Leader PL reboot
- Candidate PL reboot

- OVS Datapath reboots
- Multiple PL reboots
- Multiple Cluster reboots
- Multiple reboots of the same OVS Datapath.
- Openstack Controller reboots

Configuration impact

The following parameters have been initially made available as configurable for EVPN. These configurations can be made via the RESTful interface:

1.Multi-homing-mode – For multi-homing use cases where redundant DCGWs are used ODL can be configured with 'none', 'all-active' or 'single-active' multi-homing mode. Default will be 'none'.

2.IRB-mode – Depending upon the support on DCGW, ODL can be configured with either 'Symmetric' or 'Asymmetric' IRB mode. Default is 'Symmetric'.

There is another important parameter though it won't be configurable:

MAC Address Prefix for EVPN – This MAC Address prefix represents the MAC Address prefix that will be hard-coded and that MACAddress will be used as the gateway mac address if it is not supplied from Openstack. This will usually be the case when networks are associated to an L3VPN with no gateway port yet configured in Openstack for such networks.

Clustering considerations

The feature should operate in ODL Clustered environment reliably.

Other Infra considerations

N.A.

Security considerations

N.A.

Scale and Performance Impact

Not covered by this Design Document.

Targeted Release

Carbon.

Alternatives

Alternatives considered and why they were not selected.

Usage

Features to Install

This feature can be used by installing odl-netvirt-openstack. This feature doesn't add any new karaf feature.

REST API

The creational RESTful API for the L3VPN will be enhanced to accept the L3VNI as an additional attribute as in the below request format:

```
{'input': {
    'l3vpn': [
        {'name': 'L3VPN2',
         'export-RT': ['50:2'],
         'route-distinguisher': ['50:2'],
         'import-RT': ['50:2'],
         'id': '4ae8cd92-48ca-49b5-94e1-b2921a260007',
         'l3vni': '200',
         'tenant-id': 'a565b3ed854247f795c0840b0481c699'
}]}}
```

There is no change in the REST API for associating networks, associating routers (or) deleting the L3VPN.

On the Openstack-side configuration, the vni_ranges configured in Openstack Neutron ml2_conf.ini should not overlap with the L3VNI provided in the ODL RESTful API. In an inter-DC case, where both the DCs are managed by two different Openstack Controller Instances, the workflow will be to do the following:

1. Configure the DC-GW2 facing OSC2 and DC-GW1 facing OSC1 with the same BGP configuration parameters.
2. On first Openstack Controller (OSC1) create an L3VPN1 with RD1 and L3VNI1
3. Create a network Net1 and Associate that Network Net1 to L3VPN1
4. On second Openstack Controller (OSC2) create an L3VPN2 with RD1 with L3VNI2
5. Create a network Net2 on OSC2 and associate that Network Net2 to L3VPN2.
6. Spin-off VM1 on Net1 in OSC1.
7. Spin-off VM2 on Net2 in OSC2.
8. Now VM1 and VM2 should be able to communicate.

Implementation

Assignee(s)

Primary assignee: Kiran N Upadhyaya (kiran.n.upadhyaya@ericsson.com)

Sumanth MS (sumanth.ms@ericsson.com)

Basavaraju Chickmath (basavaraju.chickmath@ericsson.com)

Other contributors: Vivekanandan Narasimhan (n.vivekanandan@ericsson.com)

Work Items

The Trello cards have already been raised for this feature under the EVPN_RT5.

Here is the link for the Trello Card: https://trello.com/c/Tfpr3ezF/33-evpn-evpn-rt5

New tasks into this will be added to cover Java UT and CSIT.

Dependencies

Requires a DC-GW that is supporting EVPN RT5 on BGP Control plane.

Testing

Capture details of testing that will need to be added.

Unit Tests

Appropriate UTs will be added for the new code coming in once framework is in place.

Integration Tests

There won't be any Integration tests provided for this feature.

CSIT

CSIT will be enhanced to cover this feature by providing new CSIT tests.

Documentation Impact

This will require changes to User Guide and Developer Guide.

User Guide will need to add information on how OpenDaylight can be used to deploy L3 BGPVPNs and enable communication across datacenters between virtual endpoints in such L3 BGPVPN.

Developer Guide will capture the ODL L3VPN API changes to enable management of an L3VPN that can use VXLAN overlay to enable communication across datacenters.

References

[1] EVPN_RT5

[2] Network Virtualization using EVPN

[3] Integrated Routing and Bridging in EVPN

[4] VXLAN DCI using EVPN

[5] BGP MPLS-Based Ethernet VPN

- http://docs.opendaylight.org/en/latest/documentation.html

- https://wiki.opendaylight.org/view/Genius:Carbon_Release_Plan

Temporary Source MAC Learning

https://git.opendaylight.org/gerrit/#/q/topic:temp-smac-learning

Temporary source MAC learning introduces two new tables to the ELAN service, for OVS-based source MAC learning using a learn action, to reduce a large scale of packets punted to the controller for an unlearned source MAC.

Problem description

Currently any packet originating from an unknown source MAC address is punted to the controller from the ELAN service (L2 SMAC table 50).

This behavior continues for each packet from this source MAC until ODL properly processes this packet and adds an explicit source MAC rule to this table.

During the time that is required to punt a packet, process it by the ODL and create an appropriate flow, it is not necessary to punt any other packet from this source MAC, as it causes an unnecessary load.

Use Cases

Any L2 traffic from unknown source MACs passing through the ELAN service.

Proposed change

A preliminary logic will be added prior to the SMAC learning table, that will use OpenFlow learn action to add a temporary rule for each source MAC after the first packet is punted.

Pipeline changes

Two new tables will be introduced to the ELAN service:

Table 48 for resubmitting to tables 49 and 50 (trick required to use the learned flows, similar to the ACL implementation).
Table 49 for setting a register value to mark that this SMAC was already punted to the ODL for learning. The flows in this table will be generated automatically by OVS.
Table 50 will be modified, with a new flow, which has a lower priority than the existing known SMAC flows but a higher priority than the default flow. This flow passes packets marked with the register directly to the DMAC table 51 without punting to the controller, as it is already being processed. In addition, the default flow that punts packets to the controller, will also have a new learn action, temporarily adding a flow matching this source MAC to table 49.

Example of flows after change:

```
cookie=0x8040000, duration=1575.755s, table=17, n_packets=7865, n_
↪bytes=1451576, priority=6,metadata=0x6000020000000000/0xffffff0000000000␣
↪actions=write_metadata:0x7000021389000000/0xfffffffffffffffe,goto_table:48
cookie=0x8500000, duration=1129.530s, table=48, n_packets=4149, n_
↪bytes=729778, priority=0 actions=resubmit(,49),resubmit(,50)
```

```
cookie=0x8600000, duration=6.875s, table=49, n_packets=0, n_bytes=0, hard_
↪timeout=60, priority=0,dl_src=fa:16:3e:2f:73:61 actions=load:0x1->NXM_NX_
↪REG4[0..7]
cookie=0x8051389, duration=7.078s, table=50, n_packets=0, n_bytes=0,␣
↪priority=20,metadata=0x21389000000/0xffffffffff000000,dl_
↪src=fa:16:3e:2f:73:61 actions=goto_table:51
cookie=0x8050000, duration=440.925s, table=50, n_packets=49, n_bytes=8030,␣
↪priority=10,reg4=0x1 actions=goto_table:51
cookie=0x8050000, duration=124.209s, table=50, n_packets=68, n_bytes=15193,␣
↪priority=0 actions=CONTROLLER:65535,learn(table=49,hard_timeout=60,
↪priority=0,cookie=0x8600000,NXM_OF_ETH_SRC[],load:0x1->NXM_NX_REG4[0..7]),
↪goto_table:51
```

Yang changes

None.

Configuration impact

None.

Clustering considerations

None.

Other Infra considerations

None.

Security considerations

None.

Scale and Performance Impact

This change should substantially reduce the packet in load from SMAC learning, resulting in a reduced load of the ODL in high performance traffic scenarios.

Targeted Release

Due to scale and performance criticality, and the low risk of this feature, suggest to target this functionality for Boron.

Alternatives

None.

Usage

N/A.

Features to Install

odl-netvirt-openstack

REST API

N/A.

CLI

N/A.

Implementation

Assignee(s)

Who is implementing this feature? In case of multiple authors, designate a primary assigne and other contributors.

Primary assignee: Olga Schukin (olga.schukin@hpe.com)

Other contributors: Alon Kochba (alonko@hpe.com)

Work Items

N/A.

Dependencies

No new dependencies. Learn action is already in use in netvirt pipeline and has been available in OVS since early versions. However this is a non-standard OpenFlow feature.

Testing

Existing source MAC learning functionality should be verified.

Unit Tests

N/A.

Integration Tests

N/A.

CSIT

N/A.

Documentation Impact

Pipeline documentation should be updated accordingly to reflect the changes to the ELAN service.

Table of Contents

Enhancement to VLAN Provider Network Support

https://git.opendaylight.org/gerrit/#/q/topic:vlan-provider-network

This feature aims to enhance the support for VLAN provider networks that are not of type external.As part of this enhancement, ELAN pipeline processing for the network will be done on the switch only if there is at least one VM port in the network on the switch. The behavior of VLAN provider networks of type external and flat networks will remain unchanged as of now. The optimization for external network is out of scope of this spec and will be handled as part of future releases.

Problem description

Current ODL implementation supports all configured VLAN segments corresponding to VLAN provider networks on a particular patch port on all Open vSwitch which are part of the network. This could have adverse performance impacts because every provider patch port will receive and processes broadcast traffic for all configured VLAN segments even in cases when the switch doesn't have a VM port in the network. Furthermore, for unknown SMACs it leads to unnecessary punts from ELAN pipeline to controller for source MAC learning from all the switches.

Use Cases

L2 forwarding between OVS switches using provider type VLAN over L2 segment of the underlay fabric

Proposed change

Instead of creating the VLAN member interface on the patch port at the time of network creation, VLAN member interface creation will be deferred until a VM port comes up in the switch in the VLAN provider network. Switch pipeline will not process broadcast traffic on this switch in a VLAN provider network until VM port is added to the network. This will be applicable to VLAN provider network without external router attribute set.

Elan service binding will also be done at the time of VLAN member interface creation. Since many neutron ports on same switch can belong to a single VLAN provider network, the flow rule should be created only once when first VM comes up and should be deleted when there are no more neutron ports in the switch for the VLAN provider network.

Pipeline changes

None.

Yang changes

`elan:elan-instances` container will be enhanced with information whether an external router is attached to VLAN provider network.

Listing 1.35: elan.yang

```
container elan-instances {
     description
        "elan instances configuration parameters. Elan instances support both the␣
↪VLAN and VNI based elans.";

     list elan-instance {
```

```
        max-elements "unbounded";
        min-elements "0";
        key "elan-instance-name";
        description
            "Specifies the name of the elan instance. It is a string of 1 to 31
             case-sensitive characters.";
        leaf elan-instance-name {
            type string;
            description "The name of the elan-instance.";
        }
        ...

        leaf external {
            description "indicates whether the network has external router attached
↪to it";
            type boolean;
            default "false";
        }
    }
}
```

Configuration impact

None

Clustering considerations

None

Other Infra considerations

N.A.

Security considerations

None.

Scale and Performance Impact

Performance will improve because of the following:

1. Switch will drop packets if it doesn't have a VM port in the VLAN on which packet is received.
2. Unnecessary punts to the controller from ELAN pipeline for source mac learning will be prevented.

Targeted Release

Carbon.

Alternatives

N.A.

Usage

Features to Install

This feature can be used by installing odl-netvirt-openstack. This feature doesn't add any new karaf feature.

REST API

CLI

Implementation

Assignee(s)

Primary assignee:

- Ravindra Nath Thakur (ravindra.nath.thakur@ericsson.com)
- Naveen Kumar Verma (naveen.kumar.verma@ericsson.com)

Other contributors:

- Ravi Sundareswaran (ravi.sundareswaran@ericsson.com)

Work Items

N.A.

Dependencies

This doesn't add any new dependencies.

Testing

Capture details of testing that will need to be added.

Unit Tests

Integration Tests

CSIT

Documentation Impact

This feature will not require any change in User Guide.

References

[1] https://trello.com/c/A6Km6J3D/110-flat-and-vlan-network-type

Table of Contents

VNI based L2 switching, L3 forwarding and NATing

https://git.opendaylight.org/gerrit/#/q/topic:vni-based-l2-l3-nat

Important: All gerrit links raised for this feature will have topic name as **vni-based-l2-l3-nat**

This feature attempts to realize the use of VxLAN VNI (Virtual Network Identifier) for VxLAN tenant traffic flowing on the cloud data-network. This is applicable to L2 switching, L3 forwarding and NATing for all VxLAN based provider networks. In doing so, it eliminates the presence of `LPort tags`, `ELAN tags` and `MPLS labels` on the wire and instead, replaces them with VNIs supplied by the tenant's OpenStack.

This will be selectively done for the use-cases covered by this spec and hence, its implementation won't completely remove the usage of the above entities. The usage of `LPort tags` and `ELAN tags` within an OVS datapath (not on the wire) of the hypervisor will be retained, as eliminating it completely is a large redesign and can be pursued incrementally later.

This spec is the first step in the direction of enforcing datapath semantics that uses tenant supplied VNI values on VxLAN Type networks created by tenants in OpenStack Neutron.

Note: The existing L3 BGPVPN control-path and data-path semantics will continue to use L3 labels on the wire as well as inside the OVS datapaths of the hypervisor to realize both intra-dc and inter-dc connectivity.

Problem description

OpenDaylight NetVirt service today supports the following types of networks:

- Flat
- VLAN
- VxLAN
- GRE

Amongst these, VxLAN-based overlay is supported only for traffic within the DataCenter. External network accesses over the DC-Gateway are supported via VLAN or GRE type external networks. For rest of the traffic over the DC-Gateway, the only supported overlay is GRE.

Today, for VxLAN enabled networks by the tenant, the labels are generated by L3 forwarding service and used. Such labels are re-used for inter-DC use-cases with BGPVPN as well. This does not honor and is not in accordance with the datapath semantics from an orchestration point of view.

This spec attempts to change the datapath semantics by enforcing the VNIs (unique for every VxLAN enabled network in the cloud) **as dictated by the tenant's OpenStack configuration for L2 switching, L3 forwarding and NATing**.

This implementation will remove the reliance on using the following (on the wire) within the DataCenter:

- Labels for L3 forwarding
- LPort tags for L2 switching

More specifically, the traffic from source VM will be routed in source OVS by the L3VPN / ELAN pipeline. After that, the packet will travel as a switched packet in the VxLAN underlay within the DC, containing the VNI in the VxLAN header instead of MPLS label / LPort tag. In the destination OVS, the packet will be collected and sent to the destination VM through the existing ELAN pipeline.

In the nodes themselves, the LPort tag will continue to be used when pushing the packet from ELAN / L3VPN pipeline towards the VM as ACLService continues to use `LPort tags`.

Simiarly `ELAN tags` will continue to be used for handling L2 broadcast packets:

- locally generated in the OVS datapath
- remotely received from another OVS datapath via internal VxLAN tunnels

LPort tag uses 8 bits and ELAN tag uses 21 bits in the metadata. The existing use of both in the metadata will remain unaffected.

In Scope

Since VNIs are provisioned only for VxLAN based underlays, this feature has in its scope the use-cases pertaining to **intra-DC connectivity over internal VxLAN tunnels only**.

On the cloud data network wire, all the VxLAN traffic for basic L2 switching within a VxLAN network and L3 forwarding across VxLAN-type networks using routers will use tenant supplied VNI values for such VXLAN networks.

Inter-DC connectivity over external VxLAN tunnels is covered by the EVPN_RT5 spec.

Out of Scope

- Complete removal of use of `LPort tags` everywhere in ODL: Use of `LPort tags` within the OVS Datapath of a hypervisor, for streaming traffic to the right virtual endpoint on that hypervisor (note: not on the wire) will be retained
- Complete removal of use of `ELAN tags` everywhere in ODL: Use of `ELAN tags` within the OVS Datapath to handle local/remote L2 broadcasts (note: not on the wire) will be retained
- Complete removal of use of `MPLS labels` everywhere in ODL: Use of `MPLS labels` for realizing an L3 BGPVPN (regardless of type of networks put into such BGPVPN that may include networks of type VxLAN) both on the wire and within the OVS Datapaths will be retained.
- Addressing or testing IPv6 use-cases
- Intra DC NAT usecase where no explicit Internet VPN is created for VxLAN based external provider networks: Detailed further in Intra DC subsection in NAT section below.

Complete removal of use of `LPort tags`, `ELAN tags` and `MPLS labels` for VxLAN-type networks has large scale design/pipeline implications and thus need to be attempted as future initiatives via respective specs.

Use Cases

This feature involves amendments/testing pertaining to the following:

L2 switching use cases

1. L2 Unicast frames exchanged within an OVS datapath
2. L2 Unicast frames exchanged over OVS datapaths that are on different hypervisors
3. L2 Broadcast frames transmitted within an OVS datapath
4. L2 Broadcast frames received from remote OVS datapaths

L3 forwarding use cases

1. Router realized using VNIs for networks attached to a new router (with network having pre-created VMs)
2. Router realized using VNIs for networks attached to a new router (with new VMs booted later on the network)
3. Router updated with one or more extra route(s) to an existing VM.
4. Router updated to remove previously added one/more extra routes.

NAT use cases

The provider network types for external networks supported today are:

- External VLAN Provider Networks (transparent Internet VPN)
- External Flat Networks (transparent Internet VPN)
- Tenant-orchestrated Internet VPN of type GRE (actually MPLSOverGRE)

Following are the SNAT/DNAT use-cases applicable to the network types listed above:

1. SNAT functionality.
2. DNAT functionality.
3. DNAT to DNAT functionality (Intra DC)
    - FIP VM to FIP VM on same hypervisor
    - FIP VM to FIP VM on different hypervisors
4. SNAT to DNAT functionality (Intra DC)
    - Non-FIP VM to FIP VM on the same NAPT hypervisor
    - Non-FIP VM to FIP VM on the same hypervisor, but NAPT on different hypervisor
    - Non-FIP VM to FIP VM on different hypervisors (with NAPT on FIP VM hypervisor)
    - Non-FIP VM to FIP VM on different hypervisors (with NAPT on Non-FIP VM hypervisor)

Proposed change

The following components within OpenDaylight Controller needs to be enhanced:

- NeutronVPN Manager
- ELAN Manager
- VPN Engine (VPN Manager, VPN Interface Manager and VPN Subnet Route Handler)
- FIB Manager
- NAT Service

Pipeline changes

L2 Switching

Unicast

Within hypervisor

There are no explicit pipeline changes for this use-case.

Across hypervisors

- *Ingress OVS*

  Instead of setting the destination LPort tag, destination network VNI will be set in the `tun_id` field in `L2_DMAC_FILTER_TABLE` (table 51) while egressing the packet on the tunnel port.

  The modifications in flows and groups on the ingress OVS are illustrated below:

```
cookie=0x8000000, duration=65.484s, table=0, n_packets=23, n_bytes=2016,␣
↪priority=4,in_port=6actions=write_metadata:0x30000000000/0xffffff0000000001,
↪goto_table:17
cookie=0x6900000, duration=63.106s, table=17, n_packets=23, n_bytes=2016,␣
↪priority=1,metadata=0x30000000000/0xffffff0000000000 actions=write_
↪metadata:0x2000030000000000/0xfffffffffffffffe,goto_table:40
cookie=0x6900000, duration=64.135s, table=40, n_packets=4, n_bytes=392,␣
↪priority=61010,ip,dl_src=fa:16:3e:86:59:fd,nw_src=12.1.0.4 actions=ct(table=41,
↪zone=5002)
cookie=0x6900000, duration=5112.542s, table=41, n_packets=21, n_bytes=2058,␣
↪priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,17)
cookie=0x8040000, duration=62.125s, table=17, n_packets=15, n_bytes=854,␣
↪priority=6,metadata=0x6000030000000000/0xffffff0000000000 actions=write_
↪metadata:0x700003138a000000/0xfffffffffffffffe,goto_table:48
cookie=0x8500000, duration=5113.124s, table=48, n_packets=24, n_bytes=3044,␣
↪priority=0 actions=resubmit(,49),resubmit(,50)
cookie=0x805138a, duration=62.163s, table=50, n_packets=15, n_bytes=854,␣
↪priority=20,metadata=0x3138a000000/0xffffffffff000000,dl_src=fa:16:3e:86:59:fd␣
↪actions=goto_table:51
cookie=0x803138a, duration=62.163s, table=51, n_packets=6, n_bytes=476,␣
↪priority=20,metadata=0x138a000000/0xffff000000,dl_dst=fa:16:3e:31:fb:91␣
↪actions=set_field:**0x710**->tun_id,output:1
```

- *Egress OVS*

  On the egress OVS, for the packets coming in via the internal VxLAN tunnel (OVS - OVS), `INTERNAL_TUNNEL_TABLE` currently matches on destination LPort tag for unicast packets. Since the incoming packets will now contain the network VNI in the VxLAN header, the `INTERNAL_TUNNEL_TABLE` will match on this VNI, set the ELAN tag in the metadata and forward the packet to `L2_DMAC_FILTER_TABLE` so as to reach the destination VM via the ELAN pipeline.

  The modifications in flows and groups on the egress OVS are illustrated below:

```
cookie=0x8000001, duration=5136.996s, table=0, n_packets=12601, n_bytes=899766,␣
↪priority=5,in_port=1,actions=write_metadata:0x10000000001/0xfffff0000000001,
↪goto_table:36
cookie=0x9000004, duration=1145.594s, table=36, n_packets=15, n_bytes=476,␣
↪priority=5,**tun_id=0x710,actions=write_metadata:0x138a000001/0xfffffffff000000,
↪goto_table:51**
cookie=0x803138a, duration=62.163s, table=51, n_packets=9, n_bytes=576,␣
↪priority=20,metadata=0x138a000001/0xffff000000,dl_dst=fa:16:3e:86:59:fd␣
↪actions=load:0x300->NXM_NX_REG6[],resubmit(,220)
cookie=0x6900000, duration=63.122s, table=220, n_packets=9, n_bytes=1160,␣
↪priority=6,reg6=0x300actions=load:0x70000300->NXM_NX_REG6[],write_
↪metadata:0x7000030000000000/0xfffffffffffffffe,goto_table:251
cookie=0x6900000, duration=65.479s, table=251, n_packets=8, n_bytes=392,␣
↪priority=61010,ip,dl_dst=fa:16:3e:86:59:fd,nw_dst=12.1.0.4 actions=ct(table=252,
↪zone=5002)
cookie=0x6900000, duration=5112.299s, table=252, n_packets=19, n_bytes=1862,␣
↪priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,220)
```

```
cookie=0x8000007, duration=63.123s, table=220, n_packets=8, n_bytes=1160,␣
↪priority=7,reg6=0x70000300actions=output:6
```

Broadcast

Across hypervisors

The ARP broadcast by the VM will be a (local + remote) broadcast.

For the local broadcast on the VM's OVS itself, the packet will continue to get flooded to all the VM ports by setting the destination LPort tag in the local broadcast group. Hence, there are no explicit pipeline changes for when a packet is transmitted within the source OVS via a local broadcast.

The changes in pipeline for the remote broadcast are illustrated below:

- *Ingress OVS*

  Instead of setting the ELAN tag, network VNI will be set in the `tun_id` field as part of bucket actions in remote broadcast group while egressing the packet on the tunnel port.

  The modifications in flows and groups on the ingress OVS are illustrated below:

```
cookie=0x8000000, duration=65.484s, table=0, n_packets=23, n_bytes=2016,␣
↪priority=4,in_port=6actions=write_metadata:0x30000000000/0xffffff0000000001,
↪goto_table:17
cookie=0x6900000, duration=63.106s, table=17, n_packets=23, n_bytes=2016,␣
↪priority=1,metadata=0x30000000000/0xffffff0000000000 actions=write_
↪metadata:0x2000030000000000/0xfffffffffffffffe,goto_table:40
cookie=0x6900000, duration=64.135s, table=40, n_packets=4, n_bytes=392,␣
↪priority=61010,ip,dl_src=fa:16:3e:86:59:fd,nw_src=12.1.0.4 actions=ct(table=41,
↪zone=5002)
cookie=0x6900000, duration=5112.542s, table=41, n_packets=21, n_bytes=2058,␣
↪priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,17)
cookie=0x8040000, duration=62.125s, table=17, n_packets=15, n_bytes=854,␣
↪priority=6,metadata=0x6000030000000000/0xffffff0000000000 actions=write_
↪metadata:0x700003138a000000/0xfffffffffffffffe,goto_table:48
cookie=0x8500000, duration=5113.124s, table=48, n_packets=24, n_bytes=3044,␣
↪priority=0 actions=resubmit(,49),resubmit(,50)
cookie=0x805138a, duration=62.163s, table=50, n_packets=15, n_bytes=854,␣
↪priority=20,metadata=0x3138a000000/0xffffffffff000000,dl_src=fa:16:3e:86:59:fd␣
↪actions=goto_table:51
cookie=0x8030000, duration=5112.911s, table=51, n_packets=18, n_bytes=2568,␣
↪priority=0 actions=goto_table:52
cookie=0x870138a, duration=62.163s, table=52, n_packets=9, n_bytes=378,␣
↪priority=5,metadata=0x138a000000/0xffff000001 actions=write_
↪actions(group:210004)

group_id=210004,type=all,bucket=actions=group:210003,bucket=actions=set_
↪field:**0x710**->tun_id,output:1
```

- *Egress OVS*

  On the egress OVS, for the packets coming in via the internal VxLAN tunnel (OVS - OVS), `INTERNAL_TUNNEL_TABLE` currently matches on ELAN tag for broadcast packets. Since the incoming packets will now contain the network VNI in the VxLAN header, the `INTERNAL_TUNNEL_TABLE` will match on this VNI, set the ELAN tag in the metadata and forward the packet to `L2_DMAC_FILTER_TABLE` to be broadcasted via the local broadcast groups traversing the ELAN pipeline.

The TUNNEL_INGRESS_BIT being set in the CLASSIFIER_TABLE (table 0) ensures that the packet is always sent to the local broadcast group only and hence, remains within the OVS. This is necessary to avoid switching loop back to the source OVS.

The modifications in flows and groups on the egress OVS are illustrated below:

```
cookie=0x8000001, duration=5136.996s, table=0, n_packets=12601, n_bytes=899766,␣
↪priority=5,in_port=1,actions=write_metadata:0x10000000001/0xfffff0000000001,
↪goto_table:36
cookie=0x9000004, duration=1145.594s, table=36, n_packets=15, n_bytes=476,␣
↪priority=5,**tun_id=0x710,actions=write_metadata:0x138a000001/0xffffffff000000,
↪goto_table:51**
cookie=0x8030000, duration=5137.609s, table=51, n_packets=9, n_bytes=1293,␣
↪priority=0 actions=goto_table:52
cookie=0x870138a, duration=1145.592s, table=52, n_packets=0, n_bytes=0,␣
↪priority=5,metadata=0x138a000001/0xffff000001 actions=apply_
↪actions(group:210003)

group_id=210003,type=all,bucket=actions=set_field:0x4->tun_id,resubmit(,55)

cookie=0x8800004, duration=1145.594s, table=55, n_packets=9, n_bytes=378,␣
↪priority=9,tun_id=0x4,actions=load:0x400->NXM_NX_REG6[],resubmit(,220)
cookie=0x6900000, duration=63.122s, table=220, n_packets=9, n_bytes=1160,␣
↪priority=6,reg6=0x300actions=load:0x70000300->NXM_NX_REG6[],write_
↪metadata:0x7000030000000000/0xfffffffffffffffe,goto_table:251
cookie=0x6900000, duration=65.479s, table=251, n_packets=8, n_bytes=392,␣
↪priority=61010,ip,dl_dst=fa:16:3e:86:59:fd,nw_dst=12.1.0.4 actions=ct(table=252,
↪zone=5002)
cookie=0x6900000, duration=5112.299s, table=252, n_packets=19, n_bytes=1862,␣
↪priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,220)
cookie=0x8000007, duration=63.123s, table=220, n_packets=8, n_bytes=1160,␣
↪priority=7,reg6=0x70000300actions=output:6
```

The ARP response will be a unicast packet, and as indicated above, for unicast packets, there are no explicit pipeline changes.

L3 Forwarding

Between VMs on a single OVS

There are no explicit pipeline changes for this use-case. The destination LPort tag will continue to be set in the nexthop group since when The EGRESS_DISPATCHER_TABLE sends the packet to EGRESS_ACL_TABLE, it is used by the ACL service.

Between VMs on two different OVS

L3 forwarding between VMs on two different hypervisors is asymmetric forwarding since the traffic is routed in the source OVS datapath while it is switched over the wire and then all the way to the destination VM on the destination OVS datapath.

VM sourcing the traffic (Ingress OVS)

L3_FIB_TABLE will set the destination network VNI in the tun_id field instead of the MPLS label.

```
CLASSIFIER_TABLE => DISPATCHER_TABLE => INGRESS_ACL_TABLE =>
DISPATCHER_TABLE => L3_GW_MAC_TABLE =>
L3_FIB_TABLE (set destination MAC, **set tunnel-ID as destination network VNI**)
=> Output to tunnel port
```

The modifications in flows and groups on the ingress OVS are illustrated below:

```
cookie=0x8000000, duration=128.140s, table=0, n_packets=25, n_bytes=2716, priority=4,
↪in_port=5 actions=write_metadata:0x50000000000/0xffffff0000000001,goto_table:17
cookie=0x8000000, duration=4876.599s, table=17, n_packets=0, n_bytes=0, priority=0,
↪metadata=0x5000000000000000/0xf000000000000000 actions=write_
↪metadata:0x6000000000000000/0xf000000000000000,goto_table:80
cookie=0x1030000, duration=4876.563s, table=80, n_packets=0, n_bytes=0, priority=0␣
↪actions=resubmit(,17)
cookie=0x6900000, duration=123.870s, table=17, n_packets=25, n_bytes=2716, priority=1,
↪metadata=0x50000000000/0xffffff0000000000 actions=write_metadata:0x2000050000000000/
↪0xfffffffffffffffe,goto_table:40
cookie=0x6900000, duration=126.056s, table=40, n_packets=15, n_bytes=1470,␣
↪priority=61010,ip,dl_src=fa:16:3e:63:ea:0c,nw_src=10.1.0.4 actions=ct(table=41,
↪zone=5001)
cookie=0x6900000, duration=4877.057s, table=41, n_packets=17, n_bytes=1666,␣
↪priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,17)
cookie=0x6800001, duration=123.485s, table=17, n_packets=28, n_bytes=3584, priority=2,
↪metadata=0x2000050000000000/0xffffff0000000000 actions=write_
↪metadata:0x5000050000000000/0xfffffffffffffffe,goto_table:60
cookie=0x6800000, duration=3566.900s, table=60, n_packets=24, n_bytes=2184,␣
↪priority=0 actions=resubmit(,17)
cookie=0x8000001, duration=123.456s, table=17, n_packets=17, n_bytes=1554, priority=5,
↪metadata=0x5000050000000000/0xffffff0000000000 actions=write_
↪metadata:0x60000500000222e0/0xfffffffffffffffe,goto_table:19
cookie=0x8000009, duration=124.815s, table=19, n_packets=15, n_bytes=1470,␣
↪priority=20,metadata=0x222e0/0xffffffff,dl_dst=fa:16:3e:51:da:ee actions=goto_
↪table:21
cookie=0x8000003, duration=125.568s, table=21, n_packets=9, n_bytes=882, priority=42,
↪ip,metadata=0x222e0/0xfffffffe,nw_dst=12.1.0.3 actions=**set_field:0x710->tun_id**,
↪set_field:fa:16:3e:31:fb:91->eth_dst,output:1
```

VM receiving the traffic (Egress OVS)

On the egress OVS, for the packets coming in via the VxLAN tunnel, `INTERNAL_TUNNEL_TABLE` currently matches on MPLS label and sends it to the nexthop group to be taken to the destination VM via `EGRESS_ACL_TABLE`. Since the incoming packets will now contain network VNI in the VxLAN header, the `INTERNAL_TUNNEL_TABLE` will match on the VNI, set the ELAN tag in the metadata and forward the packet to `L2_DMAC_FILTER_TABLE`, from where it will be taken to the destination VM via the ELAN pipeline.

```
CLASSIFIER_TABLE => INTERNAL_TUNNEL_TABLE (Match on network VNI, set ELAN tag in the␣
↪metadata)
=> L2_DMAC_FILTER_TABLE (Match on destination MAC) => EGRESS_DISPATCHER_TABLE
=> EGRESS_ACL_TABLE => Output to destination VM port
```

The modifications in flows and groups on the egress OVS are illustrated below:

```
cookie=0x8000001, duration=4918.647s, table=0, n_packets=12292, n_bytes=877616,␣
↪priority=5,in_port=1actions=write_metadata:0x10000000001/0xfffff0000000001,goto_
↪table:36
cookie=0x9000004, duration=927.245s, table=36, n_packets=8234, n_bytes=52679,␣
↪priority=5,**tun_id=0x710,actions=write_metadata:0x138a000001/0xffffffffff000000,
↪goto_table:51**
```

```
cookie=0x803138a, duration=62.163s, table=51, n_packets=9, n_bytes=576, priority=20,
↪metadata=0x138a000001/0xffff000000,dl_dst=fa:16:3e:86:59:fd actions=load:0x300->NXM_
↪NX_REG6[],resubmit(,220)
cookie=0x6900000, duration=63.122s, table=220, n_packets=9, n_bytes=1160, priority=6,
↪reg6=0x300actions=load:0x70000300->NXM_NX_REG6[],write_metadata:0x7000030000000000/
↪0xfffffffffffffffe,goto_table:251
cookie=0x6900000, duration=65.479s, table=251, n_packets=8, n_bytes=392,␣
↪priority=61010,ip,dl_dst=fa:16:3e:86:59:fd,nw_dst=12.1.0.4 actions=ct(table=252,
↪zone=5002)
cookie=0x6900000, duration=5112.299s, table=252, n_packets=19, n_bytes=1862,␣
↪priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,220)
cookie=0x8000007, duration=63.123s, table=220, n_packets=8, n_bytes=1160, priority=7,
↪reg6=0x70000300actions=output:6
```

NAT Service

For NAT, we need VNIs to be used in two scenarios:

- When packet is forwarded from non-NAPT to NAPT hypervisor (VNI per router)
- Between hypervisors (intra DC) over Internet VPN (VNI per Internet VPN)

Hence, a pool titled `opendaylight-vni-ranges`, non-overlapping with the OpenStack Neutron vni_ranges configuration, needs to be configured by the OpenDaylight Controller Administrator.

This `opendaylight-vni-ranges` pool will be used to carve out a unique VNI per router to be then used in the datapath for traffic forwarding from non-NAPT to NAPT switch for this router.

Similarly, for MPLSOverGRE based external networks, the `opendaylight-vni-ranges` pool will be used to carve out a unique VNI per Internet VPN (GRE-provider-type) to be then used in the datapath for traffic forwarding for `SNAT-to-DNAT` and `DNAT-to-DNAT` cases within the DataCenter. Only one external network can be associated to Internet VPN today and this spec doesn't attempt to address that limitation.

A NeutronVPN configuration API will be exposed to the administrator to configure the lower and higher limit for this pool. If the administrator doesn't configure this explicitly, then the pool will be created with default values of lower limit set to 70000 and upper limit set to 100000, during the first NAT session configuration.

FIB Manager changes: For external network of type GRE, it is required to use `Internet VPN VNI` for intra-DC communication, but we still require `MPLS labels` to reach SNAT/DNAT VMs from external entities via MPLSOverGRE. Hence, we will make use of the `l3vni` attribute added to fibEntries container as part of EVPN_RT5 spec. NAT will populate both `label` and `l3vni` values for fibEntries created for floating-ips and external-fixed-ips with external network of type GRE. This `l3vni` value will be used while programming remote FIB flow entries (on all the switches which are part of the same VRF). But still, MPLS label will be used to advertise prefixes and in `L3_LFIB_TABLE` taking the packet to `INBOUND_NAPT_TABLE` and `PDNAT_TABLE`.

For SNAT/DNAT use-cases, we have following provider network types for External Networks:

1. VLAN - not VNI based
2. Flat - not VNI based
3. VxLAN - VNI based (covered by the EVPN_RT5 spec)
4. GRE - not VNI based (will continue to use `MPLS labels`)

Inter DC

SNAT

- From a VM on a NAPT switch to reach Internet, and reverse traffic reaching back to the VM

  There are no explicit pipeline changes.

- From a VM on a non-NAPT switch to reach Internet, and reverse traffic reaching back to the VM

  On the non-NAPT switch, `PSNAT_TABLE` (table 26) will be set with `tun_id` field as `Router Based VNI` allocated from the pool and send to group to reach NAPT switch.

  On the NAPT switch, `INTERNAL_TUNNEL_TABLE` (table 36) will match on the `tun_id` field which will be `Router Based VNI` and send the packet to `OUTBOUND_NAPT_TABLE` (table 46) for SNAT Translation and to be taken to Internet.

  - *Non-NAPT switch*

```
cookie=0x8000006, duration=2797.179s, table=26, n_packets=47, n_bytes=3196,␣
↪priority=5,ip,metadata=0x23a50/0xfffffffe actions=**set_field:0x710->tun_
↪id**,group:202501

group_id=202501,type=all,bucket=actions=output:1
```

  - *NAPT switch*

```
cookie=0x8000001, duration=4918.647s, table=0, n_packets=12292, n_
↪bytes=877616, priority=5,in_port=1,actions=write_metadata:0x10000000001/
↪0xfffff0000000001,goto_table:36
cookie=0x9000004, duration=927.245s, table=36, n_packets=8234, n_bytes=52679,␣
↪priority=10,ip,**tun_id=0x710**,actions=write_metadata:0x23a50/0xfffffffe,
↪goto_table:46
```

    As part of the response from NAPT switch, the packet will be taken to the Non-NAPT switch after SNAT reverse translation using destination VMs Network VNI.

DNAT

There is no NAT specific explicit pipeline change for DNAT traffic to DC-gateway.

Intra DC

- VLAN Provider External Networks: VNI is not applicable on the external VLAN Provider network. However, the Router VNI will be used for datapath traffic from non-NAPT switch to NAPT-switch over the internal VxLAN tunnel.
- VxLAN Provider External Networks:
  - **Explicit creation of Internet VPN**: An L3VNI, mandatorily falling within the `opendaylight-vni-ranges`, will be provided by the Cloud admin (or tenant). This VNI will be used uniformly for all packet transfer over the VxLAN wire for this Internet VPN (uniformly meaning all the traffic on Internal or External VXLAN Tunnel, except the non-NAPT to NAPT communication). This usecase is covered by EVPN_RT5 spec

  - **No explicit creation of Internet VPN**: A transparent Internet VPN having UUID same as that of the corresponding external network UUID is created implicitly and the VNI configured for this external network should be used on the VxLAN wire. This usecase is **out of scope** from the perspective of this spec, and the same is indicated in *Out of Scope* section.

- GRE Provider External Networks: `Internet VPN VNI` will be carved per Internet VPN using `opendaylight-vni-ranges` to be used on the wire.

DNAT to DNAT

- FIP VM to FIP VM on different hypervisors

  After DNAT translation on the first hypervisor `DNAT-OVS-1`, the traffic will be sent to the `L3_FIB_TABLE` (table=21) in order to reach the floating IP VM on the second hypervisor `DNAT-OVS-2`. Here, the `tun_id` action field will be set as the `INTERNET VPN VNI` value.

  - *DNAT-OVS-1*

```
cookie=0x8000003, duration=518.567s, table=21, n_packets=0, n_bytes=0,␣
↪priority=42,ip,metadata=0x222e8/0xfffffffe,nw_dst=172.160.0.200␣
↪actions=**set_field:0x11178->tun_id**,output:9
```

  - *DNAT-OVS-2*

```
cookie=0x9011177, duration=411685.075s, table=36, n_packets=2, n_bytes=196,␣
↪priority=**6**,**tun_id=0x11178**actions=resubmit(,25)
cookie=0x9011179, duration=478573.171s, table=36, n_packets=2, n_bytes=140,␣
↪priority=5,**tun_id=0x11178**,actions=goto_table:44

cookie=0x8000004, duration=408145.805s, table=25, n_packets=600, n_
↪bytes=58064, priority=10,ip,nw_dst=172.160.0.100,**eth_
↪dst=fa:16:3e:e6:e3:c6** actions=set_field:10.0.0.5->ip_dst,write_
↪metadata:0x222e0/0xfffffffe,goto_table:27
cookie=0x8000004, duration=408145.805s, table=25, n_packets=600, n_
↪bytes=58064, priority=10,ipactions=goto_table:44
```

  First, the `INTERNAL_TUNNEL_TABLE` (table=36) will take the packet to the `PDNAT_TABLE` (table 25) for an exact FIP match in `PDNAT_TABLE`.

  - In case of a successful FIP match, `PDNAT_TABLE` will further match on floating IP MAC. This is done as a security prerogative since in DNAT usecases, the packet can land to the hypervisor directly from the external world. Hence, better to have a second match criteria.
  - In case of no match, the packet will be redirected to the SNAT pipeline towards the `INBOUND_NAPT_TABLE` (table=44). This is the use-case where `DNAT-OVS-2` also acts as the NAPT switch.

  In summary, on an given NAPT switch, if both DNAT and SNAT are configured, the incoming traffic will first be sent to the `PDNAT_TABLE` and if there is no FIP match found, then it will be forwarded to `INBOUND_NAPT_TABLE` for SNAT translation.

  As part of the response, the `Internet VPN VNI` will be used as `tun_id` to reach floating IP VM on `DNAT-OVS-1`.

- FIP VM to FIP VM on same hypervisor

  The pipeline changes will be similar as are for different hypervisors, the only difference being that `INTERNAL_TUNNEL_TABLE` will never be hit in this case.

SNAT to DNAT

- Non-FIP VM to FIP VM on different hypervisors (with NAPT elected as the FIP VM hypervisor)

  The packet will be sent to the NAPT hypervisor from non-FIP VM (for SNAT translation) using `Router VNI` (similar to as described in *SNAT* section). As part of the response from the NAPT switch after SNAT reverse translation, the packet is forwarded to non-FIP VM using destination VM's Network VNI.

- Non-FIP VM to FIP VM on the same NAPT hypervisor

  There are no explicit pipeline changes for this use-case.

- Non-FIP VM to FIP VM on the same hypervisor, but a different hypervisor elected as NAPT switch

  - *NAPT hypervisor*

    The packet will be sent to the NAPT hypervisor from non-FIP VM (for SNAT translation) using `Router VNI` (similar to as described in *SNAT* section). On the NAPT switch, the `INTERNAL_TUNNEL_TABLE` will match on the `Router VNI` in the `tun_id` field and send the packet to `OUTBOUND_NAPT_TABLE` for SNAT translation (similar to as described in *SNAT* section).

```
cookie=0x8000005, duration=5073.829s, table=36, n_packets=61, n_bytes=4610,␣
↪priority=10,ip,**tun_id=0x11170**,actions=write_metadata:0x222e0/0xfffffffe,
↪goto_table:46
```

    The packet will later be sent back to the FIP VM hypervisor from L3_FIB_TABLE with `tun_id` field set as the `Internet VPN VNI`.

```
cookie=0x8000003, duration=518.567s, table=21, n_packets=0, n_bytes=0,␣
↪priority=42,ip,metadata=0x222e8/0xfffffffe,nw_dst=172.160.0.200␣
↪actions=**set_field:0x11178->tun_id**,output:9
```

  - *FIP VM hypervisor*

    On reaching the FIP VM Hypervisor, the packet will be sent for DNAT translation. The `INTERNAL_TUNNEL_TABLE` will match on the `Internet VPN VNI` in the `tun_id` field and send the packet to `PDNAT_TABLE`.

```
cookie=0x9011177, duration=411685.075s, table=36, n_packets=2, n_bytes=196,␣
↪priority=**6**,**tun_id=0x11178**,actions=resubmit(,25)
cookie=0x8000004, duration=408145.805s, table=25, n_packets=600, n_
↪bytes=58064, priority=10,ip,nw_dst=172.160.0.100,**eth_
↪dst=fa:16:3e:e6:e3:c6** actions=set_field:10.0.0.5->ip_dst,write_
↪metadata:0x222e0/0xfffffffe,goto_table:27
```

    Upon FIP VM response, DNAT reverse translation happens and traffic is sent back to the NAPT switch for SNAT translation. The `L3_FIB_TABLE` will be set with `Internet VPN VNI` in the `tun_id` field.

```
cookie=0x8000003, duration=95.300s, table=21, n_packets=2, n_bytes=140,␣
↪priority=42,ip,metadata=0x222ea/0xfffffffe,nw_dst=172.160.0.3 actions=**set_
↪field:0x11178->tun_id**,output:5
```

  - *NAPT hypervisor*

    On NAPT hypervisor, the `INTERNAL_TUNNEL_TABLE` will match on the `Internet VPN VNI` in the `tun_id` field and send the packet to '' INBOUND_NAPT_TABLE'' for SNAT reverse translation (external fixed IP to VM IP). The packet will then be sent back to the non-FIP VM using destination VM's Network VNI.

- Non-FIP VM to FIP VM on different hypervisors (with NAPT elected as the non-FIP VM hypervisor)

  After SNAT Translation, `Internet VPN VNI` will be used to reach FIP VM. On FIP VM hypervisor, the `INTERNAL_TUNNEL_TABLE` will take the packet to the `PDNAT_TABLE` to match on `Internet VPN VNI` in the `tun_id` field for DNAT translation.

  Upon response from FIP, DNAT reverse translation happens and uses `Internet VPN VNI` to reach back to the non-FIP VM.

YANG changes

- `opendaylight-vni-ranges` and `enforce-openstack-semantics` leaf elements will be added to neutronvpn-config container in `neutronvpn-config.yang`:
  - `opendaylight-vni-ranges` will be introduced to accept inputs for the VNI range pool from the configurator via the corresponding exposed REST API. In case this is not defined, the default value defined in `netvirt-neutronvpn-config.xml` will be used to create this pool.
  - `enforce-openstack-semantics` will be introduced to have the flexibility to enable or disable OpenStack semantics in the dataplane for this feature. It will be defaulted to true, meaning these semantics will be enforced by default. In case it is set to false, the dataplane will continue to be programmed with LPort tags / ELAN tags for switching and with labels for routing use-cases. Once this feature gets stabilized and the semantics are in place to use VNIs on the wire for BGPVPN based forwarding too, this config can be permanently removed if deemed fit.

Listing 1.36: neutronvpn-config.yang

```
container neutronvpn-config {
    config true;
    ...
    ...
    leaf opendaylight-vni-ranges {
        type string;
        default "70000:99999";
    }
    leaf enforce-openstack-semantics {
        type boolean;
        default true;
    }
}
```

- Provider network-type and provider segmentation-ID need to be propagated to FIB Manager to manipulate flows based on the same. Hence:
  - A new grouping `network-attributes` will be introduced in `neutronvpn.yang` to hold network type and segmentation ID. This grouping will replace the leaf-node `network-id` in `subnetmaps` MD-SAL configuration datastore:

Listing 1.37: neutronvpn.yang

```
grouping network-attributes {
    leaf network-id {
        type    yang:uuid;
        description "UUID representing the network";
    }
    leaf network-type {
        type enumeration {
            enum "FLAT";
```

```
            enum "VLAN";
            enum "VXLAN";
            enum "GRE";
        }
    }
    leaf segmentation-id {
        type uint32;
        description "Optional. Isolated segment on the physical network.
            If segment-type is vlan, this ID is a vlan identifier.
            If segment-type is vxlan, this ID is a vni.
            If segment-type is flat/gre, this ID is set to 0";
    }
}

container subnetmaps {
    ...
    ...
    uses network-attributes;
}
```

– These attributes will be propagated upon addition of a router-interface or addition of a subnet to a BGPVPN to VPN Manager module via the `subnet-added-to-vpn` notification modelled in `neutronvpn.yang`. Hence, the following node will be added:

Listing 1.38: neutronvpn.yang

```
notification subnet-added-to-vpn {
    description "new subnet added to vpn";
    ...
    ...
    uses network-attributes;
}
```

– VpnSubnetRouteHandler will act on these notifications and store these attributes in `subnet-op-data` MD-SAL operational datastore as described below. FIB Manager will get to retrieve the `subnetID` from the primary adjacency of the concerned VPN interface. This `subnetID` will be used as the key to retrieve `network-attributes` from `subnet-op-data` datastore.

Listing 1.39: odl-l3vpn.yang

```
import neutronvpn {
      prefix nvpn;
      revision-date "2015-06-02";
}

container subnet-op-data {
    ...
    ...
    uses nvpn:network-attributes;
}
```

- `subnetID` and `nat-prefix` leaf elements will be added to `prefix-to-interface` container in `odl-l3vpn.yang`:
  – For NAT use-cases where the VRF entry is not always associated with a VPN interface (eg. for NAT entries such as floating IP and router-gateway-IPs for external VLAN / flat networks), `subnetID` leaf element will be added to make it possible to retrieve the `network-attributes`.

– To distinguish a non-NAT prefix from a NAT prefix, `nat-prefix` leaf element will be added. This is a boolean attribute indicating whether the prefix is a NAT prefix (meaning a floating IP, or an external-fixed-ip of a router-gateway). The VRFEntry corresponding to the NAT prefix entries here may carry both the `MPLS label` and the `Internet VPN VNI`. For SNAT-to-DNAT within the datacenter, where the Internet VPN contains an MPLSOverGRE based external network, this VRF entry will publish the `MPLS label` to BGP while the `Internet VPN VNI` (also known as `L3VNI`) will be used to carry intra-DC traffic on the external segment within the datacenter.

Listing 1.40: odl-l3vpn.yang

```
container prefix-to-interface {
    config false;
    list vpn-ids {
        key vpn-id;
        leaf vpn-id {type uint32;}
        list prefixes {
            key ip_address;
            ...
            ...
            leaf subnet-id {
                type yang:uuid;
            }
            leaf nat-prefix {
                type boolean;
                default false;
            }
        }
    }
}
```

Configuration impact

- We have to make sure that we do not accept configuration of VxLAN type provider networks without the `segmentation-ID` available in them since we are using it to represent the VNI on the wire and in the flows/groups.

Clustering considerations

No specific additional clustering considerations to be adhered to.

Other Infra considerations

None.

Security considerations

None.

Scale and Performance Impact

None.

Targeted Release(s)

Carbon.

Known Limitations

None.

Alternatives

N.A.

Usage

Features to Install

odl-netvirt-openstack

REST API

No new changes to the existing REST APIs.

CLI

No new CLI is being added.

Implementation

Assignee(s)

Primary assignee: Abhinav Gupta <abhinav.gupta@ericsson.com> Vivekanandan Narasimhan <n.vivekanandan@ericsson.com>

Other contributors: Chetan Arakere Gowdru <chetan.arakere@altencalsoftlabs.com> Karthikeyan Krishnan <karthikeyan.k@altencalsoftlabs.com> Yugandhar Sarraju <yugandhar.s@altencalsoftlabs.com>

Work Items

Trello card: https://trello.com/c/PfARbEmU/84-enforce-vni-on-the-wire-for-l2-switching-l3-forwarding-and-nating-on-vxlan-overlay-

1. Code changes to alter the pipeline and e2e testing of the use-cases mentioned.
2. Add Documentation

Dependencies

This doesn't add any new dependencies.

Testing

Unit Tests

Appropriate UTs will be added for the new code coming in once framework is in place.

Integration Tests

There won't be any Integration tests provided for this feature.

CSIT

No new testcases to be added, existing ones should continue to succeed.

Documentation Impact

This will require changes to the Developer Guide.

Developer Guide needs to capture how this feature modifies the existing Netvirt L3 forwarding service implementation.

References

- http://docs.opendaylight.org/en/latest/documentation.html
- https://wiki.opendaylight.org/view/Genius:Carbon_Release_Plan
- EVPN_RT5

Table of Contents

Neutron Port Allocation For DHCP Service

https://git.opendaylight.org/gerrit/#/q/topic:neutron_port_dhcp

This feature will enable the Neutron DHCP proxy service within controller to reserve and use a Neutron port per subnet for communication with Neutron endpoints.

Problem description

The DHCP service currently assumes availability of the subnet gateway IP address and its mac address for its DHCP proxy service, which may or may not be available to the controller. This can lead to service unavailability.

Problem - 1: L2 Deployment with 3PP gateway

There can be deployment scenario in which L2 network is created with no distributed Router/VPN functionality. This deployment can have a separate gateway for the network such as a 3PP LB VM, which acts as a TCP termination point and this LB VM is configured with a default gateway IP. It means all inter-subnet traffic is terminated on this VM which takes the responsibility of forwarding the traffic.

But the current DHCP proxy service in controller hijacks gateway IP address for serving DHCP discover/request messages. If the LB is up, this can continue to work, DHCP broadcasts will get hijacked by the ODL, and responses sent as PKT_OUTs with SIP = GW IP.

However, if the LB is down, and the VM ARPs for the same IP as part of a DHCP renew workflow, the ARP resolution can fail, due to which renew request will not be generated. This can cause the DHCP lease to lapse.

Problem - 2: Designated DHCP for SR-IOV VMs via HWVTEP

In this Deployment scenario, L2 network is created with no distributed Router/VPN functionality, and HWVTEP for SR-IOV VMs. DHCP flood requests from SR-IOV VMs (DHCP discover, request during bootup), are flooded by the HWVTEP on the ELAN, and punted to the controller by designated vswitch. DHCP offers are sent as unicast responses from Controller, which are forwarded by the HWVTEP to the VM. DHCP renews can be unicast requests, which the HWVTEP may forward to an external Gateway VM (3PP LB VM) as unicast packets. Designated vswitch will never receive these pkts, and thus not be able to punt them to the controller, so renews will fail.

High-Level Components:

The following components of the Openstack - ODL solution need to be enhanced to provide port allocation for DHCP service.

- Openstack ODL Mechanism Driver
- OpenDaylight Controller (NetVirt VpnService/DHCP Service/Elan Service)

We will review enhancements that will be made to each of the above components in following sections.

Proposed change

The following components within OpenDaylight Controller needs to be enhanced:

- Neutron VPN module
- DHCP module
- ELAN and L3VPN modules

OpenDaylight controller needs to preserve a Neutron port for every subnet so that DHCP proxy service can be enabled in Openstack deployment. The Neutron port's device owner property is set to `network:dhcp` and uses this port for all outgoing DHCP messages. Since this port gets a distinct IP address and MAC address from the subnet, both problem-1 and problem-2 will be solved.

ODL Driver Changes:

ODL driver will need a config setting when ODL DHCP service is in use, as against when Neutron DHCP agent is deployed (Community ODL default setting). This needs to be enabled for ODL deployment

ODL driver will insert an async call in subnet create/update workflow in POST_COMMIT for subnets with DHCP set to 'enabled', with a port create request, with device owner set to `network:dhcp`, and device ID set to controller hostname/IP (from ml2_conf.ini file)

ODL driver will insert an async call in subnet delete, and DHCP 'disable' workflow to ensure the allocated port is deleted

ODL driver needs to ensure at any time no more than a single port is allocated per subnet for these requirements

Pipeline changes

For example, If a VM interface is having 30.0.0.1/de:ad:be:ef:00:05 as its Gateway (or) Router Interface IP/MAC address and its subnet DHCP neutron port is created with IP/MAC address 30.0.0.4/de:ad:be:ef:00:04. The ELAN pipeline is changed like below.

```
LPort Dispatcher Table (17)=>ELAN ARP Check Table(43) => ARP Responder Group (5000) =>
↪ ARP Responder Table (81) => Egress dispatcher Table(220)

cookie=0x8040000, duration=627.038s, table=17, n_packets=0, n_bytes=0, priority=6,␣
↪metadata=0xc019a00000000000/0xffffff0000000000 actions=write_
↪metadata:0xc019a01771000000/0xfffffffffffffffe,goto_table:43
cookie=0x1080000, duration=979.712s, table=43, n_packets=0, n_bytes=0, priority=100,
↪arp,arp_op=1 actions=group:5000
cookie=0x1080000, duration=979.713s, table=43, n_packets=0, n_bytes=0, priority=100,
↪arp,arp_op=2 actions=CONTROLLER:65535,resubmit(,48)
cookie=0x8030000, duration=979.717s, table=43, n_packets=0, n_bytes=0, priority=0␣
↪actions=goto_table:48
cookie=0x262219a4, duration=312.151s, table=81, n_packets=0, n_bytes=0, priority=100,
↪arp,metadata=0xc019a000000/0xfffffffff000000,arp_tpa=30.0.0.1,arp_op=1␣
↪actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:de:ad:be:ef:00:05->eth_
↪src,load:0x2->NXM_OF_ARP_OP[], move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_
↪ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xdeadbeef0005->NXM_NX_ARP_SHA[],load:0x1e000001->
↪NXM_OF_ARP_SPA[],load:0->NXM_OF_IN_PORT[],load:0x19a000->NXM_NX_REG6[],resubmit(,
↪220)
cookie=0x262219a4, duration=312.151s, table=81, n_packets=0, n_bytes=0, priority=100,
↪arp,metadata=0xc019a000000/0xfffffffff000000,arp_tpa=30.0.0.4,arp_op=1␣
↪actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:de:ad:be:ef:00:04->eth_
↪src,load:0x2->NXM_OF_ARP_OP[], move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_
↪ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xdeadbeef0004->NXM_NX_ARP_SHA[],load:0x1e000001->
↪NXM_OF_ARP_SPA[],load:0->NXM_OF_IN_PORT[],load:0x19a000->NXM_NX_REG6[],resubmit(,
↪220)

group_id=5000,type=all,bucket=actions=CONTROLLER:65535,bucket=actions=resubmit(,48),
↪bucket=actions=resubmit(,81)
```

ARP Changes for DHCP port

1. Client VM ARP requests for DHCP server IP need to be answered in L2 as well as L3 deployment. 2. Create ARP responder table flow entry for DHCP server IP in computes nodes on which ELAN footprint is available. 3. Currently ARP responder is part of L3VPN pipeline, however no L3 service may be available in an L2 deployment to leverage the current ARP pipeline, for DHCP IP ARP responses. To ensure ARP responses are sent in L2 deployment, ARP processing needs to be migrated to the ELAN pipeline. 4. ELAN service to provide API to other services needing ARP responder entries including L3VPN service (for router MAC, router-gw MAC and floating IPs, and EVPN remote MAC entries). 5. ELAN service will be responsible for punting a copy of each ARP packet to the controller if the source MAC address is not already learned.

Assumptions

Support for providing port allocation for DHCP service is available from Openstack Pike release.

Reboot Scenarios

This feature support all the following Reboot Scenarios for EVPN:

- Entire Cluster Reboot
- Leader PL reboot
- Candidate PL reboot

- OVS Datapath reboots
- Multiple PL reboots
- Multiple Cluster reboots
- Multiple reboots of the same OVS Datapath.
- Openstack Controller reboots

Clustering considerations

The feature should operate in ODL Clustered environment reliably.

Other Infra considerations

N.A.

Security considerations

N.A.

Scale and Performance Impact

Not covered by this Design Document.

Targeted Release

Nitrogen, Carbon

Alternatives

Alternatives considered and why they were not selected.

Usage

Features to Install

This feature doesn't add any new karaf feature.

REST API

Implementation

The programming of flow rules in Table 43 and Table 81 is handled in ELAN module and following APIs are exposed from `IElanService` so that L3VPN and DHCP modules can use it to program ARP responder table flow entries for Gateway/Router Interface, floating IPs and DHCP port.

```
void addArpResponderEntry(BigIneger dpId, String ingressInterfaceName,
    String ipAddress, String macAddress, Optional<Integer> lportTag);
void removeArpResponderEntry(BigIneger dpId, String ingressInterfaceName,
    String ipAddress, String macAddress, Optional<Integer> lportTag);
```

A new container is introduced to hold the subnet DHCP port information.

Listing 1.41: dhcpservice-api.yang

```
    container subnet-dhcp-port-data {
        config true;
        list subnet-to-dhcp-port {
            key "subnet-id";
            leaf subnet-id {
                type string;
            }
            leaf port-name {
                type string;
            }
            leaf port-fixedip {
                type string;
            }
            leaf port-macaddress {
                type string;
            }
        }
    }
```

When no DHCP port is available for the subnet we will flag an error to indicate DHCP service failure for virtual endpoints on such subnets which are dhcp-enabled in Openstack neutron.

Assignee(s)

Primary assignee: Karthik Prasad <karthik.p@altencalsoftlabs.com> Achuth Maniyedath <achuth.m@altencalsoftlabs.com> Vijayalakshmi CN <vijayalakshmi.c@altencalsoftlabs.com>

Other contributors: Dayavanti Gopal Kamath <dayavanti.gopal.kamath@ericsson.com> Vivekanandan Narasimhan <n.vivekanandan@ericsson.com> Periyasamy Palanisamy <periyasamy.palanisamy@ericsson.com>

Work Items

Dependencies

Testing

CSIT

CSIT will be enhanced to cover this feature by providing new CSIT tests.

Documentation Impact

This will require changes to User Guide and Developer Guide.

References

- OpenStack Spec - https://review.openstack.org/#/c/453160

OpFlex agent-ovs Install Guide

Required Packages

You'll need to install the following packages and their dependencies:

- libuv
- openvswitch
- libopflex
- libmodelgbp
- agent-ovs

Packages are available for Red Hat Enterprise Linux 7 and Ubuntu 14.04 LTS. Some of the examples below are specific to RHEL7 but you can run the equivalent commands for upstart instead of systemd.

Note that many of these steps may be performed automatically if you're deploying this along with a larger orchestration system.

Host Networking Configuration

You'll need to set up your VM host uplink interface. You should ensure that the MTU of the underlying network is sufficient to handle tunneled traffic. We will use an example of setting up *eth0* as your uplink interface with a vlan of 4093 used for the networking control infrastructure and tunnel data plane.

We just need to set the MTU and disable IPv4 and IPv6 autoconfiguration. The MTU needs to be large enough to allow both the VXLAN header and VLAN tags to pass through without fragmenting for best performance. We'll use 1600 bytes which should be sufficient assuming you are using a default 1500 byte MTU on your virtual machine traffic. If you already have any NetworkManager connections configured for your uplink interface find the connection name and proceed to the next step. Otherwise, create a connection with (be sure to update the variable UPLINK_IFACE as needed):

```
UPLINK_IFACE=eth0
nmcli c add type ethernet ifname $UPLINK_IFACE
```

Now, configure your interface as follows:

```
CONNECTION_NAME="ethernet-$UPLINK_IFACE"
nmcli connection mod "$CONNECTION_NAME" connection.autoconnect yes \
    ipv4.method link-local \
    ipv6.method ignore \
    802-3-ethernet.mtu 9000 \
    ipv4.routes '224.0.0.0/4 0.0.0.0 2000'
```

Then bring up the interface with:

```
nmcli connection up "$CONNECTION_NAME"
```

Next, create the infrastructure interface using the infrastructure VLAN (4093 by default). We'll need to create a vlan subinterface of your uplink interface, the configure DHCP on that interface. Run the following commands. Be sure to replace the variable values if needed. If you're not using NIC teaming, replace the variable team0 below:

```
UPLINK_IFACE=team0
INFRA_VLAN=4093
nmcli connection add type vlan ifname $UPLINK_IFACE.$INFRA_VLAN dev $UPLINK_IFACE id
↪$INFRA_VLAN
nmcli connection mod vlan-$UPLINK_IFACE.$INFRA_VLAN \
    ethernet.mtu 1600 ipv4.routes '224.0.0.0/4 0.0.0.0 1000'
sed "s/CLIENT_ID/01:$(ip link show $UPLINK_IFACE | awk '/ether/ {print $2}')/" \
    > /etc/dhcp/dhclient-$UPLINK_IFACE.$INFRA_VLAN.conf <<EOF
send dhcp-client-identifier CLIENT_ID;
request subnet-mask, domain-name, domain-name-servers, host-name;
EOF
```

Now bring up the new interface with:

```
nmcli connection up vlan-$UPLINK_IFACE.$INFRA_VLAN
```

If you were successful, you should be able to see an IP address when you run:

```
ip addr show dev $UPLINK_IFACE.$INFRA_VLAN
```

OVS Bridge Configuration

We'll need to configure an OVS bridge which will handle the traffic for any virtual machines or containers that are hosted on the VM host. First, enable the openvswitch service and start it:

```
# systemctl enable openvswitch
ln -s '/usr/lib/systemd/system/openvswitch.service' '/etc/systemd/system/multi-user.
↪target.wants/openvswitch.service'
# systemctl start openvswitch
# systemctl status openvswitch
openvswitch.service - Open vSwitch
   Loaded: loaded (/usr/lib/systemd/system/openvswitch.service; enabled)
   Active: active (exited) since Fri 2014-12-12 17:20:13 PST; 3s ago
  Process: 3053 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
 Main PID: 3053 (code=exited, status=0/SUCCESS)
Dec 12 17:20:13 ovs-server.cisco.com systemd[1]: Started Open vSwitch.
```

Next, we can create an OVS bridge (you may wish to use a different bridge name):

```
# ovs-vsctl add-br br0
# ovs-vsctl show
34aa83d7-b918-4e49-bcec-1b521acd1962
    Bridge "br0"
        Port "br0"
            Interface "br0"
                type: internal
    ovs_version: "2.3.90"
```

Next, we configure a tunnel interface on our new bridge as follows:

```
# ovs-vsctl add-port br0 br0_vxlan0 -- \
    set Interface br0_vxlan0 type=vxlan \
```

```
    options:remote_ip=flow options:key=flow options:dst_port=8472
# ovs-vsctl show
34aa83d7-b918-4e49-bcec-1b521acd1962
    Bridge "br0"
        Port "br0_vxlan0"
            Interface "br0_vxlan0"
                type: vxlan
                options: {dst_port="8472", key=flow, remote_ip=flow}
        Port "br0"
            Interface "br0"
                type: internal
    ovs_version: "2.3.90"
```

Open vSwitch is now configured and ready.

Agent Configuration

Before enabling the agent, we'll need to edit its configuration file, which is located at "/etc/opflex-agent-ovs/opflex-agent-ovs.conf".

First, we'll configure the Opflex protocol parameters. If you're using an ACI fabric, you'll need the OpFlex domain from the ACI configuration, which is the name of the VMM domain you mapped to the interface for this hypervisor. Set the "domain" field to this value. Next, set the "name" field to a hostname or other unique identifier for the VM host. Finally, set the "peers" list to contain the fixed static anycast peer address of 10.0.0.30 and port 8009. Here is an example of a completed section (bold text shows areas you'll need to modify):

```
"opflex": {
    // The globally unique policy domain for this agent.
    "domain": "[CHANGE ME]",

    // The unique name in the policy domain for this agent.
    "name": "[CHANGE ME]",

    // a list of peers to connect to, by hostname and port.  One
    // peer, or an anycast pseudo-peer, is sufficient to bootstrap
    // the connection without needing an exhaustive list of all
    // peers.
    "peers": [
        {"hostname": "10.0.0.30", "port": 8009}
    ],

    "ssl": {
        // SSL mode.  Possible values:
        // disabled: communicate without encryption
        // encrypted: encrypt but do not verify peers
        // secure: encrypt and verify peer certificates
        "mode": "encrypted",

        // The path to a directory containing trusted certificate
        // authority public certificates, or a file containing a
        // specific CA certificate.
        "ca-store": "/etc/ssl/certs/"
    }
},
```

Next, configure the appropriate policy renderer for the ACI fabric. You'll want to use a stitched-mode renderer. You'll

need to configure the bridge name and the uplink interface name. The remote anycast IP address will need to be obtained from the ACI configuration console, but unless the configuration is unusual, it will be 10.0.0.32:

```
// Renderers enforce policy obtained via OpFlex.
"renderers": {
    // Stitched-mode renderer for interoperating with a
    // hardware fabric such as ACI
    "stitched-mode": {
        "ovs-bridge-name": "br0",

        // Set encapsulation type.  Must set either vxlan or vlan.
        "encap": {
            // Encapsulate traffic with VXLAN.
            "vxlan" : {
                // The name of the tunnel interface in OVS
                "encap-iface": "br0_vxlan0",

                // The name of the interface whose IP should be used
                // as the source IP in encapsulated traffic.
                "uplink-iface": "eth0.4093",

                // The vlan tag, if any, used on the uplink interface.
                // Set to zero or omit if the uplink is untagged.
                "uplink-vlan": 4093,

                // The IP address used for the destination IP in
                // the encapsulated traffic.  This should be an
                // anycast IP address understood by the upstream
                // stitched-mode fabric.
                "remote-ip": "10.0.0.32"
            }
        },
        // Configure forwarding policy
        "forwarding": {
            // Configure the virtual distributed router
            "virtual-router": {
                // Enable virtual distributed router.  Set to true
                // to enable or false to disable.  Default true.
                "enabled": true,

                // Override MAC address for virtual router.
                // Default is "00:22:bd:f8:19:ff"
                "mac": "00:22:bd:f8:19:ff",

                // Configure IPv6-related settings for the virtual
                // router
                "ipv6" : {
                    // Send router advertisement messages in
                    // response to router solicitation requests as
                    // well as unsolicited advertisements.
                    "router-advertisement": true
                }
            },

            // Configure virtual distributed DHCP server
            "virtual-dhcp": {
                // Enable virtual distributed DHCP server.  Set to
                // true to enable or false to disable.  Default
                // true.
```

```
                "enabled": true,

                // Override MAC address for virtual dhcp server.
                // Default is "00:22:bd:f8:19:ff"
                "mac": "00:22:bd:f8:19:ff"
            }
        },

        // Location to store cached IDs for managing flow state
        "flowid-cache-dir": "DEFAULT_FLOWID_CACHE_DIR"
    }
}
```

Finally, enable the agent service:

```
# systemctl enable agent-ovs
ln -s '/usr/lib/systemd/system/agent-ovs.service' '/etc/systemd/system/multi-user.
↪target.wants/agent-ovs.service'
# systemctl start agent-ovs
# systemctl status agent-ovs
agent-ovs.service - Opflex OVS Agent
   Loaded: loaded (/usr/lib/systemd/system/agent-ovs.service; enabled)
   Active: active (running) since Mon 2014-12-15 10:03:42 PST; 5min ago
 Main PID: 6062 (agent_ovs)
   CGroup: /system.slice/agent-ovs.service
           -6062 /usr/bin/agent_ovs
```

The agent is now running and ready to enforce policy. You can add endpoints to the local VM hosts using the OpFlex Group-based policy plugin from OpenStack, or manually.

TSDR Installation Guide

This document is for the user to install the artifacts that are needed for using Time Series Data Repository (TSDR) functionality in the ODL Controller by enabling either an HSQLDB, HBase, or Cassandra Data Store.

Overview

The Time Series Data Repository (TSDR) project in OpenDaylight (ODL) creates a framework for collecting, storing, querying, and maintaining time series data in the OpenDaylight SDN controller. Please refer to the User Guide for the detailed description of the functionality of the project and how to use the corresponding features provided in TSDR.

Pre Requisites for Installing TSDR

The software requirements for TSDR HBase Data Store are as follows:

- In the case when the user chooses HBase or Cassandra data store, besides the software that ODL requires, we also require HBase and Cassandra database running in single node deployment scenario.

No additional software is required for the HSQLDB Data Stores.

Preparing for Installation

- When using HBase data store, download HBase from the following website:

http://archive.apache.org/dist/hbase/hbase-0.94.15/

- When using Cassandra data store, download Cassandra from the following website:

  http://www.eu.apache.org/dist/cassandra/2.1.10/

- No additional steps are required to install the TSDR HSQL Data Store.

Installing TSDR Data Stores

Installing HSQLDB Data Store

Once OpenDaylight distribution is up, from karaf console install the HSQLDB data store using the following command:

```
feature:install odl-tsdr-hsqldb-all
```

This will install hsqldb related dependency features (and can take sometime) as well as OpenFlow statistics collector before returning control to the console.

Installing HBase Data Store

Installing TSDR HBase Data Store contains two steps:

1. Installing HBase server, and
2. Installing TSDR HBase Data Store features from ODL Karaf console.

In this release, we only support HBase single node running together on the same machine as OpenDaylight. Therefore, follow the steps to download and install HBase server onto the same machine as where OpenDaylight is running:

1. Create a folder in Linux operating system for the HBase server. For example, create an hbase directory under `/usr/lib`:

   ```
   mkdir /usr/lib/hbase
   ```

2. Unzip the downloaded HBase server tar file.

   Run the following command to unzip the installation package:

   ```
   tar xvf <hbase-installer-name>  /usr/lib/hbase
   ```

3. Make proper changes in hbase-site.xml

   (a) Under <hbase-install-directory>/conf/, there is a hbase-site.xml. Although it is not recommended, an experienced user with HBase can modify the data directory for hbase server to store the data.

   (b) Modify the value of the property with name "hbase.rootdir" in the file to reflect the desired file directory for storing hbase data.

   The following is an example of the file:

   ```
   <configuration>
     <property>
       <name>hbase.rootdir</name>
       <value>file:///usr/lib/hbase/data</value>
     </property>
     <property>
   ```

```
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/usr/lib/hbase/zookeeper</value>
  </property>
</configuration>
```

4. start hbase server:

```
cd <hbase-installation-directory>
./start-hbase.sh
```

5. start hbase shell:

```
cd <hbase-insatllation-directory>
./hbase shell
```

6. start Karaf console
7. install hbase data store feature from Karaf console:

```
feature:install odl-tsdr-hbase
```

Installing Cassandra Data Store

Installing TSDR Cassandra Data Store contains two steps:

1. Installing Cassandra server, and
2. Installing TSDR Cassandra Data Store features from ODL Karaf console.

In this release, we only support Cassadra single node running together on the same machine as OpenDaylight. Therefore, follow these steps to download and install Cassandra server onto the same machine as where OpenDaylight is running:

1. Install Cassandra (latest stable version) by downloading the zip file and untar the tar ball to cassandra/ directory on the testing machine:

```
mkdir cassandra
wget http://www.eu.apache.org/dist/cassandra/2.1.10/apache-cassandra-2.1.10-bin.
↪tar.gz[2.1.10 is current stable version, it can vary]
mv apache-cassandra-2.1.10-bin.tar.gz cassandra/
cd cassandra
tar -xvzf apache-cassandra-2.1.10-bin.tar.gz
```

2. Start Cassandra from cassandra directory by running:

```
./apache-cassandra-2.1.10/bin/cassandra
```

3. Start cassandra shell by running:

```
./apache-cassandra-2.1.10/bin/cqlsh
```

4. Start Karaf according to the instructions above.
5. Install Cassandra data store feature from Karaf console:

```
feature:install odl-tsdr-cassandra
```

Verifying your Installation

After the TSDR data store is installed, no matter whether it is HBase data store, Cassandra data store, or HSQLDB data store, the user can verify the installation with the following steps.

1. Verify if the following two TSDR commands are available from Karaf console:

```
tsdr:list
tsdr:purgeAll
```

2. Verify if OpenFlow statistics data can be received successfully:
   (a) Run "feature:install odl-tsdr-openflow-statistics-collector" from Karaf.
   (b) Run mininet to connect to ODL controller. For example, use the following command to start a three node topology:

```
mn --topo single,3  --controller 'remote,ip=172.17.252.210,port=6653' --
↪switch ovsk,protocols=OpenFlow13
```

   (c) From Karaf console, the user should be able to retrieve the statistics data of OpenFlow statistics data from the console:

```
tsdr:list FLOWSTATS
```

Troubleshooting

Check the `../data/log/karaf.log` for any exception related to TSDR features.

Post Installation Configuration

Post Installation Configuration for HSQLDB Data Store

The feature installation takes care of automated configuration of the datasource by installing a file in <install folder>/etc named org.ops4j.datasource-metric.cfg. This contains the default location of <install folder>/tsdr where the HSQLDB datastore files are stored. If you want to change the default location of the datastore files to some other location update the last portion of the url property in the org.ops4j.datasource-metric.cfg and then restart the Karaf container.

Post Installation Configuration for HBase Data Store

Please refer to HBase Data Store User Guide.

Post Installation Configuration for Cassandra Data Store

There is no post configuration for TSDR Cassandra data store.

Upgrading From a Previous Release

The HBase data store was supported in the previous release as well as in this release. However, we do not support data store upgrade for HBase data store. The user needs to reinstall TSDR and start to collect data in TSDR HBase datastore after the installation.

HSQLDB and Cassandra are new data stores introduced in this release. Therefore, upgrading from previous release does not apply in these two data store scenarios.

Uninstalling TSDR Data Stores

To uninstall TSDR HSQLDB data store

To uninstall the TSDR functionality with the default store, you need to do the following from karaf console:

```
feature:uninstall odl-tsdr-hsqldb-all
feature:uninstall odl-tsdr-core
feature:uninstall odl-tsdr-hsqldb
feature:uninstall odl-tsdr-openflow-statistics-collector
```

It is recommended to restart the Karaf container after the uninstallation of the TSDR functionality with the default store.

To uninstall TSDR HBase Data Store

To uninstall the TSDR functionality with the HBase data store,

- Uninstall HBase data store related features from karaf console:

```
feature:uninstall odl-tsdr-hbase
feature:uninstall odl-tsdr-core
```

- stop hbase server:

```
cd <hbase-installation-directory>
./stop-hbase.sh
```

- remove the file directory that contains the HBase server installation:

```
rm -r <hbase-installation-directory>
```

It is recommended to restart the Karaf container after the uninstallation of the TSDR data store.

To uninstall TSDR Cassandra Data Store

To uninstall the TSDR functionality with the Cassandra store,

- uninstall cassandra data store related features following from karaf console:

```
feature:uninstall odl-tsdr-cassandra
feature:uninstall odl-tsdr-core
```

- stop cassandra database:

```
ps auwx | grep cassandra
sudo kill pid
```

- remove the cassandra installation files:

```
rm <cassandra-installation-directory>
```

It is recommended to restart the Karaf container after uninstallation of the TSDR data store.

ElasticSearch

Setting Up the environment

To setup and run the TSDR data store ElasticSearch feature, you need to have an ElasticSearch node (or a cluster of such nodes) running. You can use a customized ElasticSearch docker image for this purpose.

Your ElasticSearch (ES) setup must have the "Delete By Query Plugin" installed. Without this, some of the ES functionality won't work properly.

Creating a custom ElasticSearch docker image

(You can skip this section if you already have an instance of ElasticSearch running)

Run the following set of commands:

```
cat << EOF > Dockerfile
FROM elasticsearch:2
RUN /usr/share/elasticsearch/bin/plugin install --batch delete-by-query
EOF
```

To build the image, run the following command in the directory where the Dockerfile was created:

```
docker build . -t elasticsearch-dd
```

You can check whether the image was properly created by running:

```
docker images
```

This should print all your container images including the elasticsearch-dd.

Now we can create and run a container from our image by typing:

```
docker run -d -p 9200:9200 -p 9300:9300 --name elasticsearch-dd elasticsearch-dd
```

To see whether the container is running, run the following command:

```
docker ps
```

The output should include a row with elasticsearch-dd in the NAMES column. To check the std out of this container use

```
docker logs elasticsearch-dd
```

Running the ElasticSearch feature

Once the features have been installed, you can change some of its properties. For example, to setup the URL where your ElasticSearch installation runs, change the *serverUrl* parameter in tsdr/persistence-elasticsearch/src/main/resources/configuration/initial/:

```
tsdr-persistence-elasticsearch.properties
```

All the data are stored into the TSDR index under a type. The metric data are stored under the metric type and the log data are store under the log type. You can modify the files in tsdr/persistence-elasticsearch/src/main/resources/configuration/initial/:

```
tsdr-persistence-elasticsearch_metric_mapping.json
tsdr-persistence-elasticsearch_log_mapping.json
```

to change or tune the mapping for those types. The changes in those files will be promoted after the feature is reloaded or the distribution is restarted.

Testing the setup

We can now test whether the setup is correct by downloading and installing mininet, which we use to send some data to the running ElasticSearch instance.

Installing the necessary features:

```
start OpenDaylight
feature:install odl-restconf odl-l2switch-switch odl-tsdr-core odl-tsdr-openflow-
↪statistics-collector
feature:install odl-tsdr-elasticsearch
```

We can check whether the distribution is now listening on port 6653:

```
netstat -an | grep 6653
```

Run mininet

```
sudo mn --topo single,3 --controller 'remote,ip=distro_ip,port=6653' --switch ovsk,
↪protocols=OpenFlow13
```

where the distro_ip is the IP address of the machine where the OpenDaylight distribution is running. This command will create three hosts connected to one OpenFlow capable switch.

We can check if data was stored by ElasticSearch in TSDR by running the following command:

```
tsdr:list FLOWTABLESTATS
```

The output should look similar to the following:

```
[NID=openflow:1][DC=FLOWTABLESTATS][MN=ActiveFlows][RK=Node:openflow:1,
↪Table:50][TS=1473427383598][3]
[NID=openflow:1][DC=FLOWTABLESTATS][MN=PacketMatch][RK=Node:openflow:1,
↪Table:50][TS=1473427383598][12]
[NID=openflow:1][DC=FLOWTABLESTATS][MN=PacketLookup][RK=Node:openflow:1,
↪Table:50][TS=1473427383598][12]
[NID=openflow:1][DC=FLOWTABLESTATS][MN=ActiveFlows][RK=Node:openflow:1,
↪Table:80][TS=1473427383598][3]
[NID=openflow:1][DC=FLOWTABLESTATS][MN=PacketMatch][RK=Node:openflow:1,
↪Table:80][TS=1473427383598][17]
```

```
[NID=openflow:1][DC=FLOWTABLESTATS][MN=PacketMatch][RK=Node:openflow:1,
↪Table:246][TS=1473427383598][19]
...
```

Or you can query your ElasticSearch instance:

```
curl -XPOST "http://elasticseach_ip:9200/_search?pretty" -d'{ "from": 0, "size":␣
↪10000, "query": { "match_all": {} } }'
```

The elasticseach_ip is the IP address of the server where the ElasticSearch is running.

Web Activity Collector

The Web Activity Collector records the meaningful REST requests made through the OpenDaylight RESTCONF interface.

How to test the RESTCONF Collector

- Install some other feature that has a RESTCONF interface, for example. "odl-tsdr-syslog-collector"
- Issue a RESTCONF command that uses either POST,PUT or DELETE. For example, you could call the register-filter RPC of tsdr-syslog-collector.
- Look up data in TSDR database from Karaf.

```
tsdr:list RESTCONF
```

- You should see the request that you have sent, along with its information (URL, HTTP method, requesting IP address and request body)
- Try to send a GET request, then check again, your request should not be registered, because the collector does not register GET requests by default.
- Open the file: "etc/tsdr.restconf.collector.cfg", and add GET to the list of METHODS_TO_LOG, so that it becomes:

```
METHODS_TO_LOG=POST,PUT,DELETE,GET
```

  - Try again to issue your GET request, and check if it was recorded this time, it should be recorder.
  - Try manipulating the other properties (PATHS_TO_LOG (which URLs do we want to log from), REMOTE_ADDRESSES_TO_LOG (which requesting IP addresses do we want to log from) and CONTENT_TO_LOG (what should be in the request's body in order to log it)), and see if the requests are getting logged.
  - Try providing invalid properties (unknown methods for the METHODS_TO_LOG parameter, or the same method repeated multiple times, and invalid regular expressions for the other parameters), then check karaf's log using "log:display". It should tell you that the value is invalid, and that it will use the default value instead.

VTN Installation Guide

Overview

OpenDaylight Virtual Tenant Network (VTN) is an application that provides multi-tenant virtual network on an SDN controller.

Conventionally, huge investment in the network systems and operating expenses are needed because the network is configured as a silo for each department and system. Therefore various network appliances must be installed for each tenant and those boxes cannot be shared with others. It is a heavy work to design, implement and operate the entire complex network.

The uniqueness of VTN is a logical abstraction plane. This enables the complete separation of logical plane from physical plane. Users can design and deploy any desired network without knowing the physical network topology or bandwidth restrictions.

VTN allows the users to define the network with a look and feel of conventional L2/L3 network. Once the network is designed on VTN, it will automatically be mapped into underlying physical network, and then configured on the individual switch leverage SDN control protocol. The definition of logical plane makes it possible not only to hide the complexity of the underlying network but also to better manage network resources. It achieves reducing reconfiguration time of network services and minimizing network configuration errors. OpenDaylight Virtual Tenant Network (VTN) is an application that provides multi-tenant virtual network on an SDN controller. It provides API for creating a common virtual network irrespective of the physical network.

It is implemented as two major components

- *VTN Manager*
- *VTN Coordinator*

VTN Manager

An OpenDaylight Plugin that interacts with other modules to implement the components of the VTN model. It also provides a REST interface to configure VTN components in OpenDaylight. VTN Manager is implemented as one plugin to the OpenDaylight. This provides a REST interface to create/update/delete VTN components. The user command in VTN Coordinator is translated as REST API to VTN Manager by the OpenDaylight Driver component. In addition to the above mentioned role, it also provides an implementation to the OpenStack L2 Network Functions API.

VTN Coordinator

The VTN Coordinator is an external application that provides a REST interface for an user to use OpenDaylight VTN Virtualization. It interacts with VTN Manager plugin to implement the user configuration. It is also capable of multiple OpenDaylight orchestration. It realizes VTN provisioning in OpenDaylight instances. In the OpenDaylight architecture VTN Coordinator is part of the network application, orchestration and services layer. VTN Coordinator will use the REST interface exposed by the VTN Manger to realize the virtual network using OpenDaylight. It uses OpenDaylight APIs (REST) to construct the virtual network in OpenDaylight instances. It provides REST APIs for northbound VTN applications and supports virtual networks spanning across multiple OpenDaylight by coordinating across OpenDaylight.

Preparing for Installation

VTN Manager

Follow the instructions in *Installing OpenDaylight*.

VTN Coordinator

1. Arrange a physical/virtual server with any one of the supported 64-bit OS environment.
   - RHEL 7
   - CentOS 7
   - Fedora 20 / 21 / 22
2. Install these packages:

```
yum install perl-Digest-SHA uuid libxslt libcurl unixODBC json-c bzip2
rpm -ivh http://yum.postgresql.org/9.3/redhat/rhel-6-x86_64/pgdg-redhat93-9.3-3.
↪noarch.rpm
yum install postgresql93-libs postgresql93 postgresql93-server postgresql93-
↪contrib postgresql93-odbc
```

Installing VTN

VTN Manager

Install Feature:

```
feature:install odl-vtn-manager-neutron odl-vtn-manager-rest
```

Note: The above command will install all features of VTN Manager. You can install only REST or Neutron also.

VTN Coordinator

- To get the Nitrogen distribution for VTN coordinator download the latest "tar.bz2" file from the below link:

```
https://nexus.opendaylight.org/content/repositories/public/org/opendaylight/vtn/
↪distribution.vtn-coordinator/6.5.0-Nitrogen/
```

- Run the below command to extract VTN Coordinator from the tar.bz2 file:

```
tar -C/ -jxvf distribution.vtn-coordinator-6.5.0-Nitrogen-bin.tar.bz2
```

This will install VTN Coordinator to /usr/local/vtn directory. The name of the tar.bz2 file name varies depending on the version. Please give the same tar.bz2 file name which is there in your directory.

- Configuring database for VTN Coordinator:

```
/usr/local/vtn/sbin/db_setup
```

- To start the Coordinator:

```
/usr/local/vtn/bin/vtn_start
```

Using VTN REST API:

Get the version of VTN REST API using the below command, and make sure the setup is working:

```
curl --user admin:adminpass -H 'content-type: application/json' -X GET http://<VTN_
↪COORDINATOR_IP_ADDRESS>:8083/vtn-webapi/api_version.json
```

The response should be like this, but version might differ:

```
{"api_version":{"version":"V1.2"}}
```

Verifying your Installation

VTN Manager

- In the karaf prompt, type the below command to ensure that vtn packages are installed:

```
feature:list | grep vtn
```

- Run any VTN Manager REST API:

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X GET http://localhost:8181/restconf/operational/vtn:vtns
```

VTN Coordinator

```
ps -ef | grep unc will list all the vtn apps
Run any REST API for VTN Coordinator version
```

Uninstalling VTN

VTN Manager

```
feature:uninstall odl-vtnmanager-all
```

VTN Coordinator

1. Stop VTN:

```
/usr/local/vtn/bin/vtn_stop
```

2. Remove the `usr/local/vtn` folder

1.2.11 Common OpenDaylight Features

OpenDaylight User Interface (DLUX)

This section introduces you to the OpenDaylight User Experience (DLUX) application.

Getting Started with DLUX

DLUX provides a number of different Karaf features, which you can enable and disable separately. They are:

1. odl-dlux-core
2. odl-dluxapps-nodes
3. odl-dluxapps-topology
4. odl-dluxapps-yangui
5. odl-dluxapps-yangvisualizer
6. odl-dluxapps-yangman

Logging In

To log in to DLUX, after installing the application:

1. Open a browser and enter the login URL http://<your-karaf-ip>:8181/index.html in your browser (Chrome is recommended).
2. Login to the application with your username and password credentials.

Note: OpenDaylight's default credentials are *admin* for both the username and password.

Working with DLUX

After you login to DLUX, if you enable only odl-dlux-core feature, you will see only topology application available in the left pane.

Note: To make sure topology displays all the details, enable the odl-l2switch-switch feature in Karaf.

DLUX has other applications such as node, yang UI and those apps won't show up, until you enable their features odl-dluxapps-nodes and odl-dluxapps-yangui respectively in the Karaf distribution.

Note: If you install your application in dlux, they will also show up on the left hand navigation after browser page refresh.

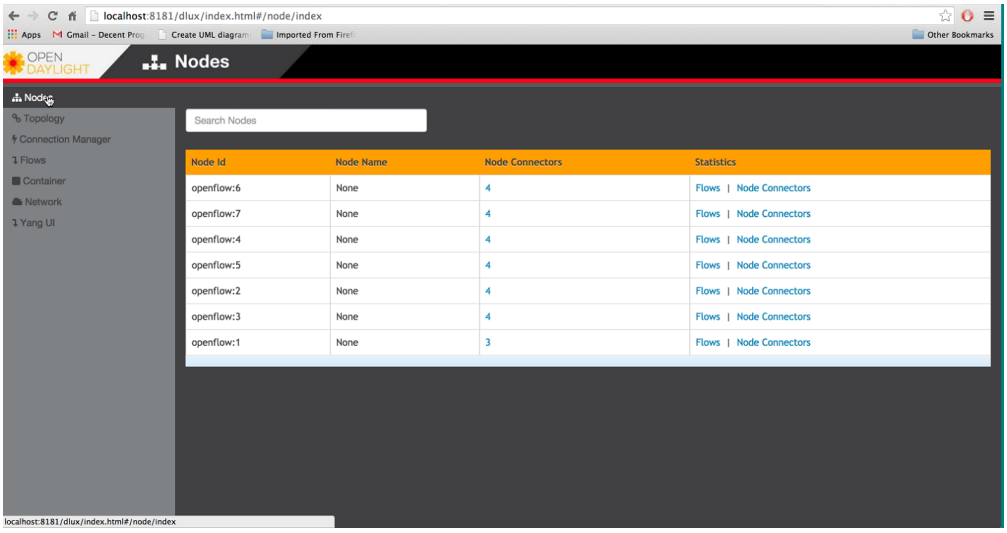


Fig. 1.1: DLUX Modules

Viewing Network Statistics

The *Nodes* module on the left pane enables you to view the network statistics and port information for the switches in the network.

To use the *Nodes* module:

1. Select *Nodes* on the left pane. The right pane displays atable that lists all the nodes, node connectors and the statistics.
2. Enter a node ID in the *Search Nodes* tab to search by node connectors.
3. Click on the *Node Connector* number to view details such as port ID, port name, number of ports per switch, MAC Address, and so on.
4. Click *Flows* in the Statistics column to view Flow Table Statistics for the particular node like table ID, packet match, active flows and so on.
5. Click *Node Connectors* to view Node Connector Statistics for the particular node ID.

Viewing Network Topology

The Topology tab displays a graphical representation of network topology created.

Note: DLUX does not allow for editing or adding topology information. The topology is generated and edited in other modules, e.g., the OpenFlow plugin. OpenDaylight stores this information in the MD-SAL datastore where DLUX can read and display it.

To view network topology:

1. Select *Topology* on the left pane. You will view the graphical representation on the right pane. In the diagram blue boxes represent the switches, the black represents the hosts available, and lines represents how the switches and hosts are connected.
2. Hover your mouse on hosts, links, or switches to view source and destination ports.

3. Zoom in and zoom out using mouse scroll to verify topology for larger topologies.

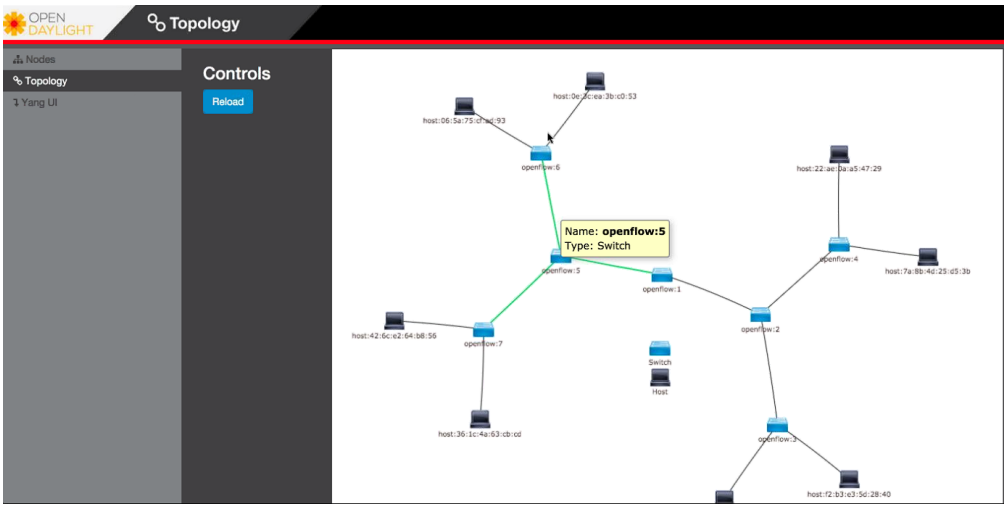


Fig. 1.2: Topology Module

Interacting with the YANG-based MD-SAL datastore

The *Yang UI* module enables you to interact with the YANG-based MD-SAL datastore. For more information about YANG and how it interacts with the MD-SAL datastore, see the *Controller* and *YANG Tools* section of the *OpenDaylight Developer Guide*.

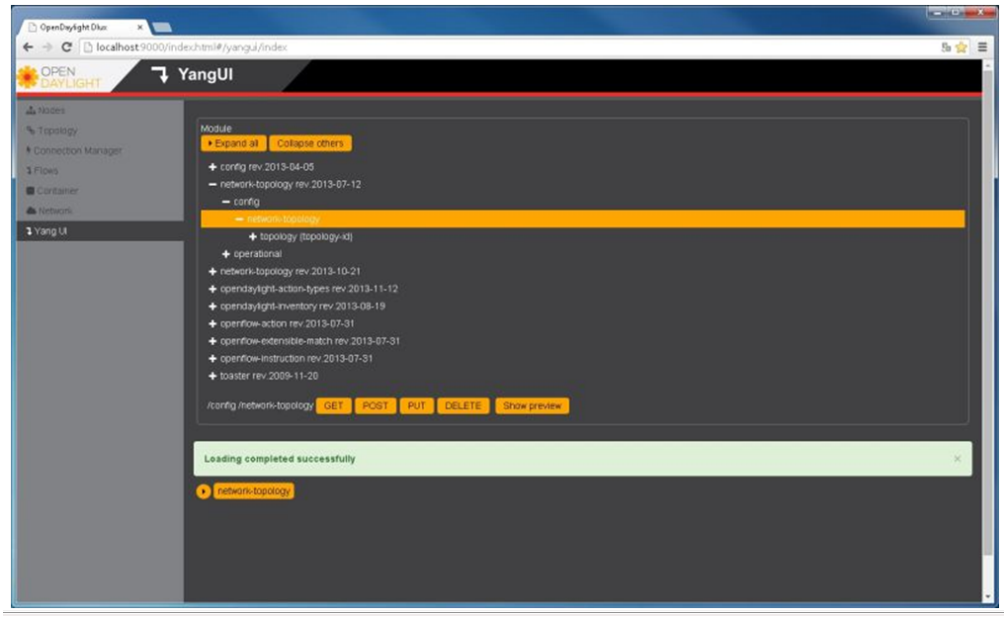


Fig. 1.3: Yang UI

To use Yang UI:

1. Select *Yang UI* on the left pane. The right pane is divided in two parts.

2. The top part displays a tree of APIs, subAPIs, and buttons to call possible functions (GET, POST, PUT, and DELETE).

Note: Not every subAPI can call every function. For example, subAPIs in the *operational* store have GET functionality only.

Inputs can be filled from OpenDaylight when existing data from OpenDaylight is displayed or can be filled by user on the page and sent to OpenDaylight.

Buttons under the API tree are variable. It depends on subAPI specifications. Common buttons are:

- GET to get data from OpenDaylight,
- PUT and POST for sending data to OpenDaylight for saving
- DELETE for sending data to OpenDaylight for deleting.

You must specify the xpath for all these operations. This path is displayed in the same row before buttons and it may include text inputs for specific path element identifiers.

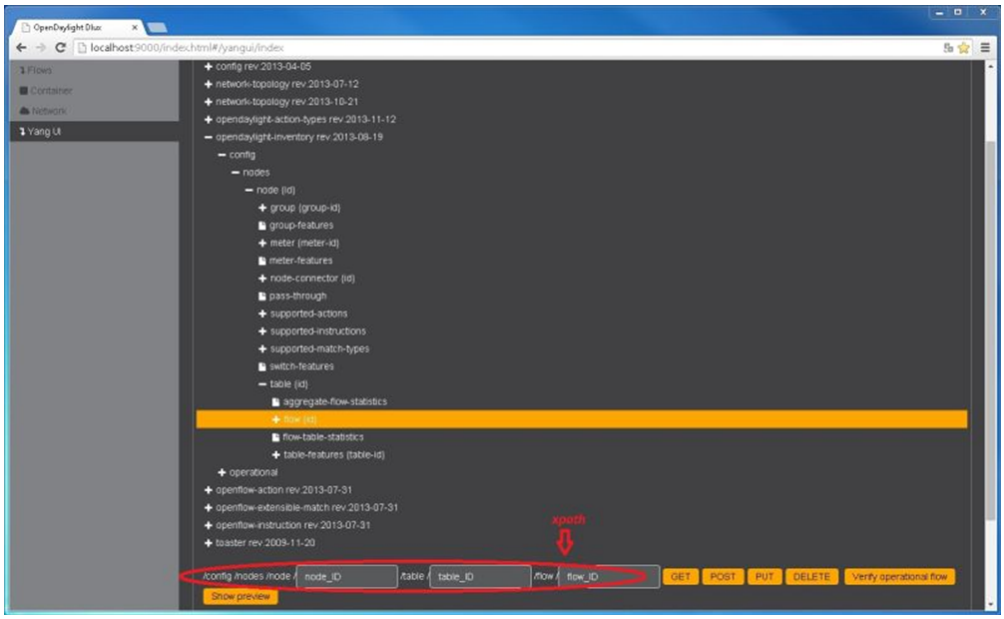


Fig. 1.4: Yang API Specification

3. The bottom part of the right pane displays inputs according to the chosen subAPI.
   - Lists are handled as a special case. For example, a device can store multiple flows. In this case "flow" is name of the list and every list element is identified by a unique key value. Elements of a list can, in turn, contain other lists.
   - In Yang UI, each list element is rendered with the name of the list it belongs to, its key, its value, and a button for removing it from the list.
4. After filling in the relevant inputs, click the *Show Preview* button under the API tree to display request that will be sent to OpenDaylight. A pane is displayed on the right side with text of request when some input is filled.

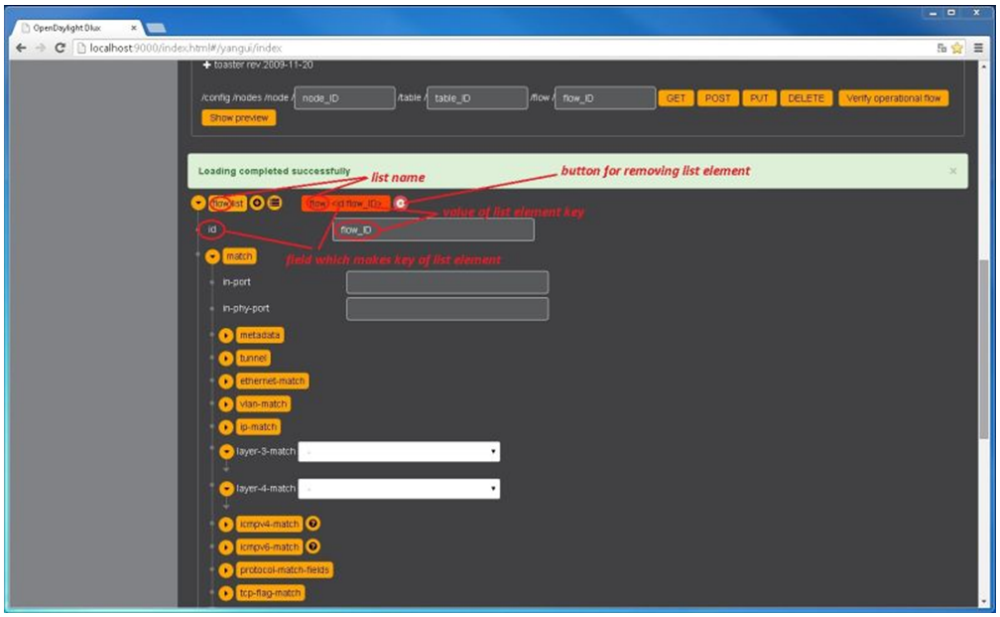


Fig. 1.5: Yang UI API Specification

Displaying Topology on the Yang UI

To display topology:

1. Select subAPI network-topology <topology revision number> == > operational == > network-topology.
2. Get data from OpenDaylight by clicking on the "GET" button.
3. Click *Display Topology*.

Configuring List Elements on the Yang UI

Lists in Yang UI are displayed as trees. To expand or collapse a list, click the arrow before name of the list. To configure list elements in Yang UI:

1. To add a new list element with empty inputs use the plus icon-button **+** that is provided after list name.
2. To remove several list elements, use the *X* button that is provided after every list element.
3. In the YANG-based data store all elements of a list must have a unique key. If you try to assign two or more elements the same key, a warning icon *!* is displayed near their name buttons.
4. When the list contains at least one list element, after the + icon, there are buttons to select each individual list element. You can choose one of them by clicking on it. In addition, to the right of the list name, there is a button which will display a vertically scrollable pane with all the list elements.

Setting Up Clustering

Clustering Overview

Clustering is a mechanism that enables multiple processes and programs to work together as one entity. For example, when you search for something on google.com, it may seem like your search request is processed by only one web

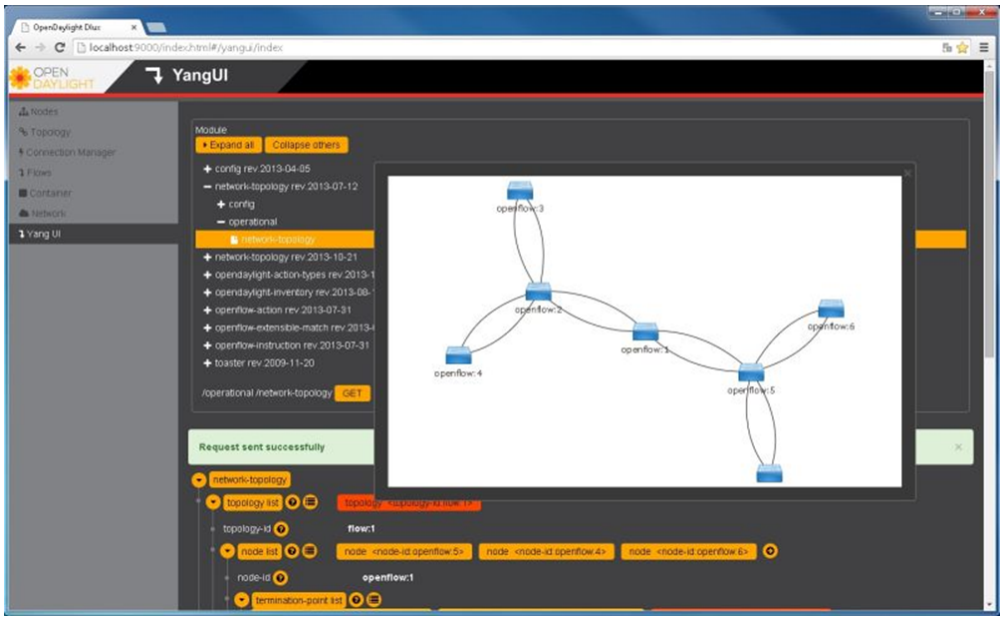


Fig. 1.6: DLUX Yang Topology

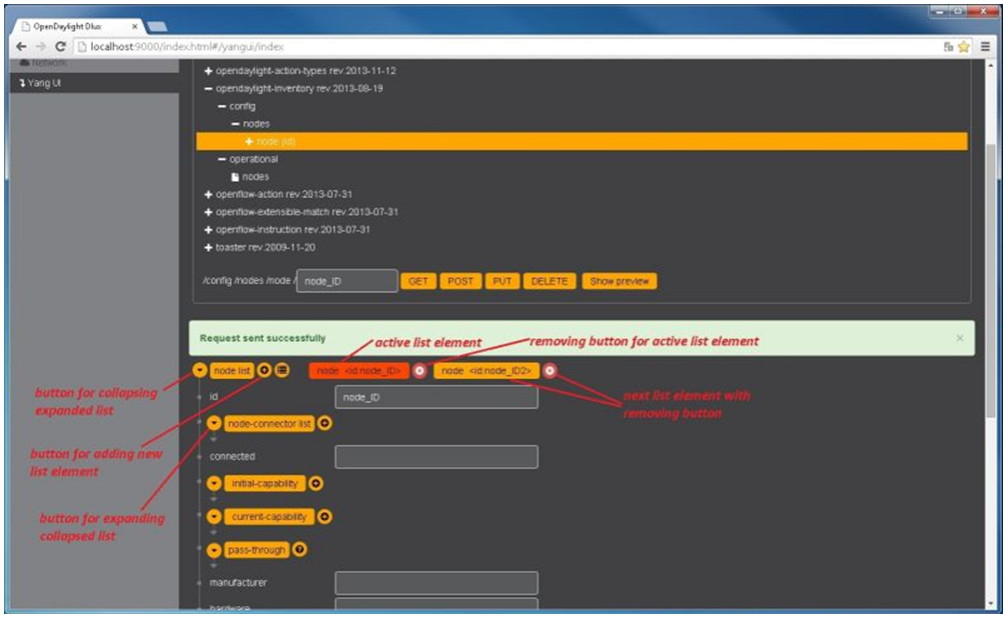


Fig. 1.7: DLUX List Elements

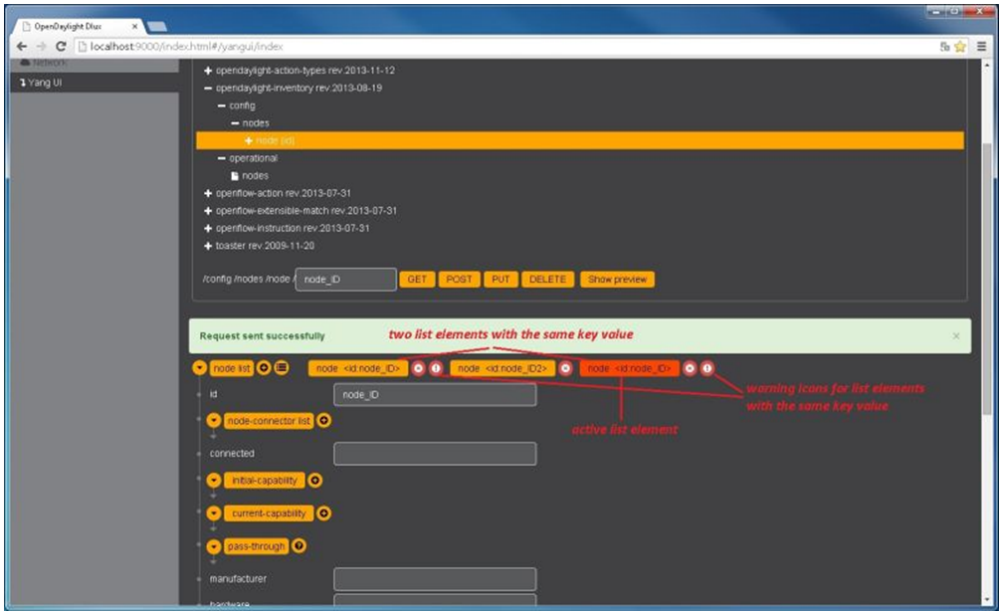


Fig. 1.8: DLUX List Warnings

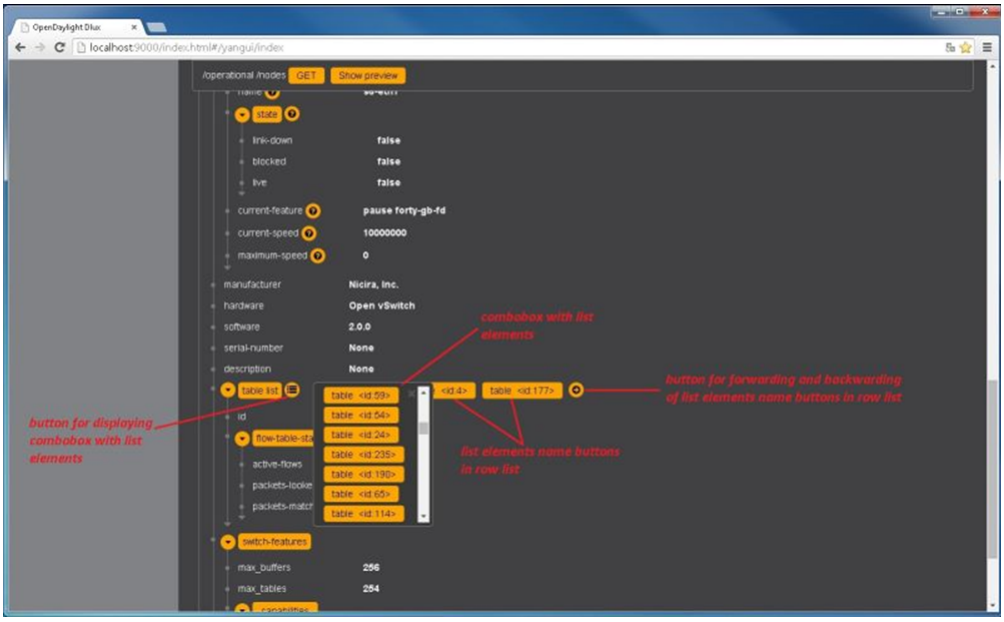


Fig. 1.9: DLUX List Button

server. In reality, your search request is processed by may web servers connected in a cluster. Similarly, you can have multiple instances of OpenDaylight working together as one entity.

Advantages of clustering are:

- Scaling: If you have multiple instances of OpenDaylight running, you can potentially do more work and store more data than you could with only one instance. You can also break up your data into smaller chunks (shards) and either distribute that data across the cluster or perform certain operations on certain members of the cluster.
- High Availability: If you have multiple instances of OpenDaylight running and one of them crashes, you will still have the other instances working and available.
- Data Persistence: You will not lose any data stored in OpenDaylight after a manual restart or a crash.

The following sections describe how to set up clustering on both individual and multiple OpenDaylight instances.

Multiple Node Clustering

The following sections describe how to set up multiple node clusters in OpenDaylight.

Deployment Considerations

To implement clustering, the deployment considerations are as follows:

- To set up a cluster with multiple nodes, we recommend that you use a minimum of three machines. You can set up a cluster with just two nodes. However, if one of the two nodes fail, the cluster will not be operational.

  Note: This is because clustering in OpenDaylight requires a majority of the nodes to be up and one node cannot be a majority of two nodes.

- Every device that belongs to a cluster needs to have an identifier. OpenDaylight uses the node's `role` for this purpose. After you define the first node's role as *member-1* in the `akka.conf` file, OpenDaylight uses *member-1* to identify that node.
- Data shards are used to contain all or a certain segment of a OpenDaylight's MD-SAL datastore. For example, one shard can contain all the inventory data while another shard contains all of the topology data.

  If you do not specify a module in the `modules.conf` file and do not specify a shard in `module-shards.conf`, then (by default) all the data is placed in the default shard (which must also be defined in `module-shards.conf` file). Each shard has replicas configured. You can specify the details of where the replicas reside in `module-shards.conf` file.
- If you have a three node cluster and would like to be able to tolerate any single node crashing, a replica of every defined data shard must be running on all three cluster nodes.

  Note: This is because OpenDaylight's clustering implementation requires a majority of the defined shard replicas to be running in order to function. If you define data shard replicas on two of the cluster nodes and one of those nodes goes down, the corresponding data shards will not function.

- If you have a three node cluster and have defined replicas for a data shard on each of those nodes, that shard will still function even if only two of the cluster nodes are running. Note that if one of those remaining two nodes goes down, the shard will not be operational.
- It is recommended that you have multiple seed nodes configured. After a cluster member is started, it sends a message to all of its seed nodes. The cluster member then sends a join command to the first seed node that

responds. If none of its seed nodes reply, the cluster member repeats this process until it successfully establishes a connection or it is shut down.

- After a node is unreachable, it remains down for configurable period of time (10 seconds, by default). Once a node goes down, you need to restart it so that it can rejoin the cluster. Once a restarted node joins a cluster, it will synchronize with the lead node automatically.

Clustering Scripts

OpenDaylight includes some scripts to help with the clustering configuration.

Note: Scripts are stored in the OpenDaylight distribution/bin folder, and maintained in the distribution project repository in the folder distribution-karaf/src/main/assembly/bin/.

Configure Cluster Script

This script is used to configure the cluster parameters (e.g. akka.conf, module-shards.conf) on a member of the controller cluster. The user should restart the node to apply the changes.

Note: The script can be used at any time, even before the controller is started for the first time.

Usage:

```
bin/configure_cluster.sh <index> <seed_nodes_list>
```

- index: Integer within 1..N, where N is the number of seed nodes. This indicates which controller node (1..N) is configured by the script.
- seed_nodes_list: List of seed nodes (IP address), separated by comma or space.

The IP address at the provided index should belong to the member executing the script. When running this script on multiple seed nodes, keep the seed_node_list the same, and vary the index from 1 through N.

Optionally, shards can be configured in a more granular way by modifying the file "custom_shard_configs.txt" in the same folder as this tool. Please see that file for more details.

Example:

```
bin/configure_cluster.sh 2 192.168.0.1 192.168.0.2 192.168.0.3
```

The above command will configure the member 2 (IP address 192.168.0.2) of a cluster made of 192.168.0.1 192.168.0.2 192.168.0.3.

Setting Up a Multiple Node Cluster

To run OpenDaylight in a three node cluster, perform the following:

First, determine the three machines that will make up the cluster. After that, do the following on each machine:

1. Copy the OpenDaylight distribution zip file to the machine.
2. Unzip the distribution.
3. Open the following .conf files:

- configuration/initial/akka.conf
- configuration/initial/module-shards.conf

4. In each configuration file, make the following changes:

   Find every instance of the following lines and replace _127.0.0.1_ with the hostname or IP address of the machine on which this file resides and OpenDaylight will run:

```
netty.tcp {
  hostname = "127.0.0.1"
```

   Note: The value you need to specify will be different for each node in the cluster.

5. Find the following lines and replace _127.0.0.1_ with the hostname or IP address of any of the machines that will be part of the cluster:

```
cluster {
  seed-nodes = ["akka.tcp://opendaylight-cluster-data@${IP_OF_MEMBER1}:2550",
                <url-to-cluster-member-2>,
                <url-to-cluster-member-3>]
```

6. Find the following section and specify the role for each member node. Here we assign the first node with the *member-1* role, the second node with the *member-2* role, and the third node with the *member-3* role:

```
roles = [
  "member-1"
]
```

   Note: This step should use a different role on each node.

7. Open the configuration/initial/module-shards.conf file and update the replicas so that each shard is replicated to all three nodes:

```
replicas = [
    "member-1",
    "member-2",
    "member-3"
]
```

   For reference, view a sample config files <<_sample_config_files,below>>.

8. Move into the +<karaf-distribution-directory>/bin+ directory.
9. Run the following command:

```
JAVA_MAX_MEM=4G JAVA_MAX_PERM_MEM=512m ./karaf
```

10. Enable clustering by running the following command at the Karaf command line:

```
feature:install odl-mdsal-clustering
```

OpenDaylight should now be running in a three node cluster. You can use any of the three member nodes to access the data residing in the datastore.

Sample Config Files

Sample `akka.conf` file:

```
odl-cluster-data {
  bounded-mailbox {
    mailbox-type = "org.opendaylight.controller.cluster.common.actor.
↪MeteredBoundedMailbox"
    mailbox-capacity = 1000
    mailbox-push-timeout-time = 100ms
  }

  metric-capture-enabled = true

  akka {
    loglevel = "DEBUG"
    loggers = ["akka.event.slf4j.Slf4jLogger"]

    actor {

      provider = "akka.cluster.ClusterActorRefProvider"
      serializers {
                java = "akka.serialization.JavaSerializer"
                proto = "akka.remote.serialization.ProtobufSerializer"
              }

              serialization-bindings {
                  "com.google.protobuf.Message" = proto

              }
    }
    remote {
      log-remote-lifecycle-events = off
      netty.tcp {
        hostname = "10.194.189.96"
        port = 2550
        maximum-frame-size = 419430400
        send-buffer-size = 52428800
        receive-buffer-size = 52428800
      }
    }

    cluster {
      seed-nodes = ["akka.tcp://opendaylight-cluster-data@10.194.189.96:2550",
                    "akka.tcp://opendaylight-cluster-data@10.194.189.98:2550",
                    "akka.tcp://opendaylight-cluster-data@10.194.189.101:2550"]

      auto-down-unreachable-after = 10s

      roles = [
        "member-2"
      ]

    }
  }
}

odl-cluster-rpc {
```

```
  bounded-mailbox {
    mailbox-type = "org.opendaylight.controller.cluster.common.actor.
↪MeteredBoundedMailbox"
    mailbox-capacity = 1000
    mailbox-push-timeout-time = 100ms
  }

  metric-capture-enabled = true

  akka {
    loglevel = "INFO"
    loggers = ["akka.event.slf4j.Slf4jLogger"]

    actor {
      provider = "akka.cluster.ClusterActorRefProvider"

    }
    remote {
      log-remote-lifecycle-events = off
      netty.tcp {
        hostname = "10.194.189.96"
        port = 2551
      }
    }

    cluster {
      seed-nodes = ["akka.tcp://opendaylight-cluster-rpc@10.194.189.96:2551"]

      auto-down-unreachable-after = 10s
    }
  }
}
```

Sample `module-shards.conf` file:

```
module-shards = [
    {
        name = "default"
        shards = [
            {
                name="default"
                replicas = [
                    "member-1",
                    "member-2",
                    "member-3"
                ]
            }
        ]
    },
    {
        name = "topology"
        shards = [
            {
                name="topology"
                replicas = [
                    "member-1",
                    "member-2",
                    "member-3"
```

```
                ]
            }
        ]
    },
    {
        name = "inventory"
        shards = [
            {
                name="inventory"
                replicas = [
                    "member-1",
                    "member-2",
                    "member-3"
                ]
            }
        ]
    },
    {
         name = "toaster"
         shards = [
             {
                 name="toaster"
                 replicas = [
                    "member-1",
                    "member-2",
                    "member-3"
                 ]
             }
         ]
    }
]
```

Cluster Monitoring

OpenDaylight exposes shard information via MBeans, which can be explored with JConsole, VisualVM, or other JMX clients, or exposed via a REST API using Jolokia, provided by the `odl-jolokia` Karaf feature. This is convenient, due to a significant focus on REST in OpenDaylight.

The basic URI that lists a schema of all available MBeans, but not their content itself is:

```
GET  /jolokia/list
```

To read the information about the shards local to the queried OpenDaylight instance use the following REST calls. For the config datastore:

```
GET  /jolokia/read/org.opendaylight.controller:type=DistributedConfigDatastore,
↪Category=ShardManager,name=shard-manager-config
```

For the operational datastore:

```
GET  /jolokia/read/org.opendaylight.controller:type=DistributedOperationalDatastore,
↪Category=ShardManager,name=shard-manager-operational
```

The output contains information on shards present on the node:

```
{
  "request": {
    "mbean": "org.opendaylight.controller:Category=ShardManager,name=shard-manager-
↪operational,type=DistributedOperationalDatastore",
    "type": "read"
  },
  "value": {
    "LocalShards": [
      "member-1-shard-default-operational",
      "member-1-shard-entity-ownership-operational",
      "member-1-shard-topology-operational",
      "member-1-shard-inventory-operational",
      "member-1-shard-toaster-operational"
    ],
    "SyncStatus": true,
    "MemberName": "member-1"
  },
  "timestamp": 1483738005,
  "status": 200
}
```


The exact names from the "LocalShards" lists are needed for further exploration, as they will be used as part of the URI to look up detailed info on a particular shard. An example output for the `member-1-shard-default-operational` looks like this:


```
{
  "request": {
    "mbean": "org.opendaylight.controller:Category=Shards,name=member-1-shard-default-
↪operational,type=DistributedOperationalDatastore",
    "type": "read"
  },
  "value": {
    "ReadWriteTransactionCount": 0,
    "SnapshotIndex": 4,
    "InMemoryJournalLogSize": 1,
    "ReplicatedToAllIndex": 4,
    "Leader": "member-1-shard-default-operational",
    "LastIndex": 5,
    "RaftState": "Leader",
    "LastCommittedTransactionTime": "2017-01-06 13:19:00.135",
    "LastApplied": 5,
    "LastLeadershipChangeTime": "2017-01-06 13:18:37.605",
    "LastLogIndex": 5,
    "PeerAddresses": "member-3-shard-default-operational: akka.tcp://opendaylight-
↪cluster-data@192.168.16.3:2550/user/shardmanager-operational/member-3-shard-default-
↪operational, member-2-shard-default-operational: akka.tcp://opendaylight-cluster-
↪data@192.168.16.2:2550/user/shardmanager-operational/member-2-shard-default-
↪operational",
    "WriteOnlyTransactionCount": 0,
    "FollowerInitialSyncStatus": false,
    "FollowerInfo": [
      {
        "timeSinceLastActivity": "00:00:00.320",
        "active": true,
        "matchIndex": 5,
        "voting": true,
        "id": "member-3-shard-default-operational",
        "nextIndex": 6
```

```
      },
      {
        "timeSinceLastActivity": "00:00:00.320",
        "active": true,
        "matchIndex": 5,
        "voting": true,
        "id": "member-2-shard-default-operational",
        "nextIndex": 6
      }
    ],
    "FailedReadTransactionsCount": 0,
    "StatRetrievalTime": "810.5 μs",
    "Voting": true,
    "CurrentTerm": 1,
    "LastTerm": 1,
    "FailedTransactionsCount": 0,
    "PendingTxCommitQueueSize": 0,
    "VotedFor": "member-1-shard-default-operational",
    "SnapshotCaptureInitiated": false,
    "CommittedTransactionsCount": 6,
    "TxCohortCacheSize": 0,
    "PeerVotingStates": "member-3-shard-default-operational: true, member-2-shard-
↪default-operational: true",
    "LastLogTerm": 1,
    "StatRetrievalError": null,
    "CommitIndex": 5,
    "SnapshotTerm": 1,
    "AbortTransactionsCount": 0,
    "ReadOnlyTransactionCount": 0,
    "ShardName": "member-1-shard-default-operational",
    "LeadershipChangeCount": 1,
    "InMemoryJournalDataSize": 450
  },
  "timestamp": 1483740350,
  "status": 200
}
```


The output helps identifying shard state (leader/follower, voting/non-voting), peers, follower details if the shard is a leader, and other statistics/counters.

The Integration team is maintaining a Python based tool, that takes advantage of the above MBeans exposed via Jolokia, and the *systemmetrics* project offers a DLUX based UI to display the same information.

Geo-distributed Active/Backup Setup

An OpenDaylight cluster works best when the latency between the nodes is very small, which practically means they should be in the same datacenter. It is however desirable to have the possibility to fail over to a different datacenter, in case all nodes become unreachable. To achieve that, the cluster can be expanded with nodes in a different datacenter, but in a way that doesn't affect latency of the primary nodes. To do that, shards in the backup nodes must be in "non-voting" state.

The API to manipulate voting states on shards is defined as RPCs in the cluster-admin.yang file in the *controller* project, which is well documented. A summary is provided below.

Note: Unless otherwise indicated, the below POST requests are to be sent to any single cluster node.

To create an active/backup setup with a 6 node cluster (3 active and 3 backup nodes in two locations) there is an RPC to set voting states of all shards on a list of nodes to a given state:

```
POST  /restconf/operations/cluster-admin:change-member-voting-states-for-all-shards
```

This RPC needs the list of nodes and the desired voting state as input. For creating the backup nodes, this example input can be used:

```
{
  "input": {
    "member-voting-state": [
      {
        "member-name": "member-4",
        "voting": false
      },
      {
        "member-name": "member-5",
        "voting": false
      },
      {
        "member-name": "member-6",
        "voting": false
      }
    ]
  }
}
```

When an active/backup deployment already exists, with shards on the backup nodes in non-voting state, all that is needed for a fail-over from the active "sub-cluster" to backup "sub-cluster" is to flip the voting state of each shard (on each node, active AND backup). That can be easily achieved with the following RPC call (no parameters needed):

```
POST  /restconf/operations/cluster-admin:flip-member-voting-states-for-all-shards
```

If it's an unplanned outage where the primary voting nodes are down, the "flip" RPC must be sent to a backup non-voting node. In this case there are no shard leaders to carry out the voting changes. However there is a special case whereby if the node that receives the RPC is non-voting and is to be changed to voting and there's no leader, it will apply the voting changes locally and attempt to become the leader. If successful, it persists the voting changes and replicates them to the remaining nodes.

When the primary site is fixed and you want to fail back to it, care must be taken when bringing the site back up. Because it was down when the voting states were flipped on the secondary, its persisted database won't contain those changes. If brought back up in that state, the nodes will think they're still voting. If the nodes have connectivity to the secondary site, they should follow the leader in the secondary site and sync with it. However if this does not happen then the primary site may elect its own leader thereby partitioning the 2 clusters, which can lead to undesirable results. Therefore it is recommended to either clean the databases (i.e., `journal` and `snapshots` directory) on the primary nodes before bringing them back up or restore them from a recent backup of the secondary site (see section *Backing Up and Restoring the Datastore*).

If is also possible to gracefully remove a node from a cluster, with the following RPC:

```
POST  /restconf/operations/cluster-admin:remove-all-shard-replicas
```

and example input:

```
{
  "input": {
    "member-name": "member-1"
```

```
  }
}
```

or just one particular shard:

```
POST  /restconf/operations/cluster-admin:remove-shard-replica
```

with example input:

```
{
  "input": {
    "shard-name": "default",
    "member-name": "member-2",
    "data-store-type": "config"
  }
}
```

Now that a (potentially dead/unrecoverable) node was removed, another one can be added at runtime, without changing the configuration files of the healthy nodes (requiring reboot):

```
POST  /restconf/operations/cluster-admin:add-replicas-for-all-shards
```

No input required, but this RPC needs to be sent to the new node, to instruct it to replicate all shards from the cluster.

Note: While the cluster admin API allows adding and removing shards dynamically, the `module-shard.conf` and `modules.conf` files are still used on startup to define the initial configuration of shards. Modifications from the use of the API are not stored to those static files, but to the journal.

Extra Configuration Options

Name	Type	Default	Description
max-shard-data-change-executor-queue-size	uint32 (1..max)	1000	The maximum queue size for each shard's data store data change notification executor.
max-shard-data-change-executor-pool-size	uint32 (1..max)	20	The maximum thread pool size for each shard's data store data change notification executor.
max-shard-data-change-listener-queue-size	uint32 (1..max)	1000	The maximum queue size for each shard's data store data change listener.
max-shard-data-store-executor-queue-size	uint32 (1..max)	5000	The maximum queue size for each shard's data store executor.
shard-transaction-idle-timeout-in-minutes	uint32 (1..max)	10	The maximum amount of time a shard transaction can be idle without receiving any messages before it self-destructs.
shard-snapshot-batch-count	uint32 (1..max)	20000	The minimum number of entries to be present in the in-memory journal log before a snapshot is to be taken.
shard-snapshot-data-threshold-percentage	uint8 (1..100)	12	The percentage of Runtime.totalMemory() used by the in-memory journal log before a snapshot is to be taken
shard-hearbeat-interval-in-millis	uint16 (100..max)	500	The interval at which a shard will send a heart beat message to its remote shard.
operation-timeout-in-seconds	uint16 (5..max)	5	The maximum amount of time for akka operations (remote or local) to complete before failing.
shard-journal-recovery-log-batch-size	uint32 (1..max)	5000	The maximum number of journal log entries to batch on recovery for a shard before committing to the data store.
shard-transaction-commit-timeout-in-seconds	uint32 (1..max)	30	The maximum amount of time a shard transaction three-phase commit can be idle without receiving the next messages before it aborts the transaction
shard-transaction-commit-queue-capacity	uint32 (1..max)	20000	The maximum allowed capacity for each shard's transaction commit queue.
shard-initialization-timeout-in-seconds	uint32 (1..max)	300	The maximum amount of time to wait for a shard to initialize from persistence on startup before failing an operation (eg transaction create and change listener registration).
shard-leader-election-timeout-in-seconds	uint32 (1..max)	30	The maximum amount of time to wait for a shard to elect a leader before failing an operation (eg transaction create).
enable-metric-capture	boolean	false	Enable or disable metric capture.
bounded-mailbox-capacity	uint32 (1..max)	1000	Max queue size that an actor's mailbox can reach
persistent	boolean	true	Enable or disable data persistence
shard-isolated-leader-check-interval-in-millis	uint32 (1..max)	5000	the interval at which the leader of the shard will check if its majority followers are active and term itself as isolated

These configuration options are included in the etc/org.opendaylight.controller.cluster.datastore.cfg configuration file.

Persistence and Backup

Set Persistence Script

This script is used to enable or disable the config datastore persistence. The default state is enabled but there are cases where persistence may not be required or even desired. The user should restart the node to apply the changes.

Note: The script can be used at any time, even before the controller is started for the first time.

Usage:

```
bin/set_persistence.sh <on/off>
```

Example:

```
bin/set_persistence.sh off
```

The above command will disable the config datastore persistence.

Backing Up and Restoring the Datastore

The same cluster-admin API described in the *cluster guide* for managing shard voting states has an RPC allowing backup of the datastore in a single node, taking only the file name as a parameter:

```
POST  /restconf/operations/cluster-admin:backup-datastore
```

RPC input JSON:

```
{
  "input": {
    "file-path": "/tmp/datastore_backup"
  }
}
```

Note: This backup can only be restored if the YANG models of the backed-up data are identical in the backup OpenDaylight instance and restore target instance.

To restore the backup on the target node the file needs to be placed into the `$KARAF_HOME/clustered-datastore-restore` directory, and then the node restarted. If the directory does not exist (which is quite likely if this is a first-time restore) it needs to be created. On startup, ODL checks if the `journal` and `snapshots` directories in `$KARAF_HOME` are empty, and only then tries to read the contents of the `clustered-datastore-restore` directory, if it exists. So for a successful restore, those two directories should be empty. The backup file name itself does not matter, and the startup process will delete it after a successful restore.

The backup is node independent, so when restoring a 3 node cluster, it is best to restore it on each node for consistency. For example, if restoring on one node only, it can happen that the other two empty nodes form a majority and the cluster comes up with no data.

Running XSQL Console Commands and Queries

XSQL Overview

XSQL is an XML-based query language that describes simple stored procedures which parse XML data, query or update database tables, and compose XML output. XSQL allows you to query tree models like a sequential database. For example, you could run a query that lists all of the ports configured on a particular module and their attributes.

The following sections cover the XSQL installation process, supported XSQL commands, and the way to structure queries.

Installing XSQL

To run commands from the XSQL console, you must first install XSQL on your system:

1. Navigate to the directory in which you unzipped OpenDaylight
2. Start Karaf:

```
./bin/karaf
```

3. Install XSQL:

```
feature:install odl-mdsal-xsql
```

XSQL Console Commands

To enter a command in the XSQL console, structure the command as follows:

```
odl:xsql _<XSQL command>_
```

The following table describes the commands supported in this OpenDaylight release.

Supported XSQL Console Commands

Command	*Description*
r	Repeats the last command you executed.
list vtables	Lists the schema node containers that are currently installed. Whenever an OpenDaylight module is installed, its YANG model is placed in the schema context. At that point, the XSQL receives a notification, confirms that the module's YANG model resides in the schema context and then maps the model to XSQL by setting up the necessary vtables and vfields. This command is useful when you need to determine vtable information for a query.
list vfields *<vtable name>*	Lists the vfields present in a specific vtable. This command is useful when you need to determine vfields information for a query.
jdbc *<ip address>*	When the ODL server is behind a firewall, and the JDBC client cannot connect to the JDBC server, run this command to start the client as a server and establish a connection.
exit	Closes the console.
tocsv	Enables or disables the forwarding of query output as a .csv file.
filename *<filename>*	Specifies the .tocsv file to which the query data is exported. If you do not specify a value for this option when the toccsv option is enabled, the filename for the query data file is generated automatically.

XSQL Queries

You can run a query to extract information that meets the criteria you specify using the information provided by the *list vtables* and *list vfields* _<vtable name>_ commands. Any query you run should be structured as follows:

select _<vfields you want to search for, separated by a comma and a space>_ *from* _<vtables you want to search in, separated by a comma and a space>_ *where* _<criteria>_ '*_<*criteria operator*>_';*

For example, if you want to search the nodes/node ID field in the nodes/node-connector table and find every instance of the Hardware-Address object that contains _BA_ in its text string, enter the following query:

```
select nodes/node.ID from nodes/node-connector where Hardware-Address like '%BA%';
```

The following criteria operators are supported:

Supported XSQL Query Criteria Operators

Criteria Operators	Description
=	Lists results that equal the value you specify.
!=	Lists results that do not equal the value you specify.
like	Lists results that contain the substring you specify. For example, if you specify *like %BC%*, every string that contains that particular substring is displayed.
<	Lists results that are less than the value you specify.
>	Lists results that are more than the value you specify.
and	Lists results that match both values you specify.
or	Lists results that match either of the two values you specify.
>=	Lists results that are more than or equal to the value you specify.
<=	Lists results that are less than or equal to the value you specify.
is null	Lists results for which no value is assigned.
not null	Lists results for which any value is assigned.
skip	Use this operator to list matching results from a child node, even if its parent node does not meet the specified criteria. See the following example for more information.

Example: Skip Criteria Operator

If you are looking at the following structure and want to determine all of the ports that belong to a YY type module:

- Network Element 1
  - Module 1, Type XX
    - Module 1.1, Type YY
      - Port 1
      - Port 2
  - Module 2, Type YY
    - Port 1
    - Port 2

If you specify *Module.Type='YY'* in your query criteria, the ports associated with module 1.1 will not be returned since its parent module is type XX. Instead, enter *Module.Type='YY' or skip Module!='YY'*. This tells XSQL to disregard any parent module data that does not meet the type YY criteria and collect results for any matching child modules. In this example, you are instructing the query to skip module 1 and collect the relevant data from module 1.1.

OpenDaylight Version

Overview

This feature allows NETCONF/RESTCONF users to determine the version of OpenDaylight they are communicating with.

Install the Version Feature

Follow these steps to install the version feature:

1. Navigate to the directory in which you installed OpenDaylight
2. Start Karaf:

```
./bin/karaf
```

3. Install Version feature:

```
feature:install odl-distribution-version
```

Note: For RESTCONF access, it is recommended to install odl-restconf and odl-netconf-connector-ssh.

Version Feature Usage

Example of RESTCONF request using curl from bash:

```
$ curl -u 'admin:admin' localhost:8181/restconf/config/network-topology:network-
↪topology/topology/topology-netconf/node/controller-config/yang-ext:mount/
↪config:modules/module/odl-distribution-version:odl-version/odl-distribution-version
```

Example response (formatted):

```
{
 "module": [
  {
   "type": "odl-distribution-version:odl-version",
   "name": "odl-distribution-version",
   "odl-distribution-version:version": "0.5.0-SNAPSHOT"
  }
 ]
}
```

1.2.12 Security Considerations

This document discusses the various security issues that might affect OpenDaylight. The document also lists specific recommendations to mitigate security risks.

This document also contains information about the corrective steps you can take if you discover a security issue with OpenDaylight, and if necessary, contact the Security Response Team, which is tasked with identifying and resolving security threats.

Overview of OpenDaylight Security

There are many different kinds of security vulnerabilities that could affect an OpenDaylight deployment, but this guide focuses on those where (a) the servers, virtual machines or other devices running OpenDaylight have been properly physically (or virtually in the case of VMs) secured against untrusted individuals and (b) individuals who have access, either via remote logins or physically, will not attempt to attack or subvert the deployment intentionally or otherwise.

While those attack vectors are real, they are out of the scope of this document.

What remains in scope is attacks launched from a server, virtual machine, or device other than the one running OpenDaylight where the attack does not have valid credentials to access the OpenDaylight deployment.

The rest of this document gives specific recommendations for deploying OpenDaylight in a secure manner, but first we highlight some high-level security advantages of OpenDaylight.

- Separating the control and management planes from the data plane (both logically and, in many cases, physically) allows possible security threats to be forced into a smaller attack surface.
- Having centralized information and network control gives network administrators more visibility and control over the entire network, enabling them to make better decisions faster. At the same time, centralization of network control can be an advantage only if access to that control is secure.

  Note: While both previous advantages improve security, they also make an OpenDaylight deployment an attractive target for attack making understanding these security considerations even more important.

- The ability to more rapidly evolve southbound protocols and how they are used provides more and faster mechanisms to enact appropriate security mitigations and remediations.
- OpenDaylight is built from OSGi bundles and the Karaf Java container. Both Karaf and OSGi provide some level of isolation with explicit code boundaries, package imports, package exports, and other security-related features.
- OpenDaylight has a history of rapidly addressing known vulnerabilities and a well-defined process for reporting and dealing with them.

OpenDaylight Security Resources

- If you have any security issues, you can send a mail to *security@lists.opendaylight.org*.
- For the list of current OpenDaylight security issues that are either being fixed or resolved, refer to https://wiki.opendaylight.org/view/Security:Advisories.
- To learn more about the OpenDaylight security issues policies and procedure, refer to https://wiki.opendaylight.org/view/Security:Main

Deployment Recommendations

We recommend that you follow the deployment guidelines in setting up OpenDaylight to minimize security threats.

- The default credentials should be changed before deploying OpenDaylight.
- OpenDaylight should be deployed in a private network that cannot be accessed from the internet.
- Separate the data network (that connects devices using the network) from the management network (that connects the network devices to OpenDaylight).

Note: Deploying OpenDaylight on a separate, private management network does not eliminate threats, but only mitigates them. By construction, some messages must flow from the data network to the management network, e.g., OpenFlow *packet_in* messages, and these create an attack surface even if it is a small one.

- Implement an authentication policy for devices that connect to both the data and management network. These are the devices which bridge, likely untrusted, traffic from the data network to the management network.

Securing OSGi bundles

OSGi is a Java-specific framework that improves the way that Java classes interact within a single JVM. It provides an enhanced version of the *java.lang.SecurityManager* (ConditionalPermissionAdmin) in terms of security.

Java provides a security framework that allows a security policy to grant permissions, such as reading a file or opening a network connection, to specific code. The code maybe classes from the jarfile loaded from a specific URL, or a class signed by a specific key. OSGi builds on the standard Java security model to add the following features:

- A set of OSGi-specific permission types, such as one that grants the right to register an OSGi service or get an OSGi service from the service registry.
- The ability to dynamically modify permissions at runtime. This includes the ability to specify permissions by using code rather than a text configuration file.
- A flexible predicate-based approach to determining which rules are applicable to which *ProtectionDomain*. This approach is much more powerful than the standard Java security policy which can only grant rights based on a jarfile URL or class signature. A few standard predicates are provided, including selecting rules based upon bundle symbolic-name.
- Support for bundle *local permissions* policies with optional further constraints such as *DENY* operations. Most of this functionality is accessed by using the *OSGi ConditionalPermissionAdmin* service which is part of the OSGi core and can be obtained from the OSGi service registry. The *ConditionalPermissionAdmin* API replaces the earlier *PermissionAdmin* API.

For more information, refer to http://www.osgi.org/Main/HomePage.

Securing the Karaf container

Apache Karaf is a OSGi-based runtime platform which provides a lightweight container for OpenDaylight and applications. Apache Karaf uses either Apache Felix Framework or Eclipse Equinox OSGi frameworks, and provide additional features on top of the framework.

Apache Karaf provides a security framework based on Java Authentication and Authorization Service (JAAS) in compliance with OSGi recommendations, while providing RBAC (Role-Based Access Control) mechanism for the console and Java Management Extensions (JMX).

The Apache Karaf security framework is used internally to control the access to the following components:

- OSGi services
- console commands
- JMX layer
- WebConsole

The remote management capabilities are present in Apache Karaf by default, however they can be disabled by using various configuration alterations. These configuration options may be applied to the OpenDaylight Karaf distribution.

Note: Refer to the following list of publications for more information on implementing security for the Karaf container.

- For role-based JMX administration, refer to http://karaf.apache.org/manual/latest/users-guide/monitoring.html.
- For remote SSH access configuration, refer to http://karaf.apache.org/manual/latest/users-guide/remote.html.
- For WebConsole access, refer to http://karaf.apache.org/manual/latest/users-guide/webconsole.html.
- For Karaf security features, refer to http://karaf.apache.org/manual/latest/developers-guide/security-framework.html.

Disabling the remote shutdown port

You can lock down your deployment post installation. Set `karaf.shutdown.port=-1` in `etc/custom.properties` or `etc/config.properties` to disable the remote shutdown port.

Securing Southbound Plugins

Many individual southbound plugins provide mechanisms to secure their communication with network devices. For example, the OpenFlow plugin supports TLS connections with bi-directional authentication and the NETCONF plugin supports connecting over SSH. Meanwhile, the Unified Secure Channel plugin provides a way to form secure, remote connections for supported devices.

When deploying OpenDaylight, you should carefully investigate the secure mechanisms to connect to devices using the relevant plugins.

Securing OpenDaylight using AAA

AAA stands for Authentication, Authorization, and Accounting. All three of can help improve the security posture of and OpenDaylight deployment. In this release, only authentication is fully supported, while authorization is an experimental feature and accounting remains a work in progress.

The vast majority of OpenDaylight's northbound APIs (and all RESTCONF APIs) are protected by AAA by default when installing the +odl-restconf+ feature. In the cases that APIs are *not* protected by AAA, this will be noted in the per-project release notes.

By default, OpenDaylight has only one user account with the username and password *admin*. This should be changed before deploying OpenDaylight.

Security Considerations for Clustering

While OpenDaylight clustering provides many benefits including high availability, scale-out performance, and data durability, it also opens a new attack surface in the form of the messages exchanged between the various instances of OpenDaylight in the cluster. In the current OpenDaylight release, these messages are neither encrypted nor authenticated meaning that anyone with access to the management network where OpenDaylight exchanges these clustering messages can forge and/or read the messages. This means that if clustering is enabled, it is even more important that the management network be kept secure from any untrusted entities.

1.2.13 How to Get Help

Users and developers can get support from the OpendayLight community through the mailing lists, IRC and forums.

1. Create your question on ServerFault or Stackoverflow with the tag *#opendaylight*.

   Note: It is important to tag questions correctly to ensure that the questions reach individuals subscribed to the tag.

2. Mail discuss@lists.opendaylight.org or dev@lists.opendaylight.org.
3. Directly mail the PTL as indicated on the specific projects page.
4. IRC: Connect to #opendaylight or #opendaylight-meeting channel on freenode.
5. For infrastructure and release engineering queries, mail helpdesk@opendaylight.org. IRC: Connect to #lf-releng channel on freenode.

1.3 OpenDaylight User Guide

1.3.1 Overview

This first part of the user guide covers the basic user operations of the OpenDaylight Release using the generic base functionality.

OpenDaylight Controller Overview

The OpenDaylight controller is JVM software and can be run from any operating system and hardware as long as it supports Java. The controller is an implementation of the Software Defined Network (SDN) concept and makes use of the following tools:

- **Maven**: OpenDaylight uses Maven for easier build automation. Maven uses pom.xml (Project Object Model) to script the dependencies between bundle and also to describe what bundles to load and start.
- **OSGi**: This framework is the back-end of OpenDaylight as it allows dynamically loading bundles and packages JAR files, and binding bundles together for exchanging information.
- **JAVA interfaces**: Java interfaces are used for event listening, specifications, and forming patterns. This is the main way in which specific bundles implement call-back functions for events and also to indicate awareness of specific state.
- **REST APIs**: These are northbound APIs such as topology manager, host tracker, flow programmer, static routing, and so on.

The controller exposes open northbound APIs which are used by applications. The OSGi framework and bidirectional REST are supported for the northbound APIs. The OSGi framework is used for applications that run in the same address space as the controller while the REST (web-based) API is used for applications that do not run in the same address space (or even the same system) as the controller. The business logic and algorithms reside in the applications. These applications use the controller to gather network intelligence, run its algorithm to do analytics, and then orchestrate the new rules throughout the network. On the southbound, multiple protocols are supported as plugins, e.g. OpenFlow 1.0, OpenFlow 1.3, BGP-LS, and so on. The OpenDaylight controller starts with an OpenFlow 1.0 southbound plugin. Other OpenDaylight contributors begin adding to the controller code. These modules are linked dynamically into a **Service Abstraction Layer** (SAL).

The SAL exposes services to which the modules north of it are written. The SAL figures out how to fulfill the requested service irrespective of the underlying protocol used between the controller and the network devices. This provides investment protection to the applications as OpenFlow and other protocols evolve over time. For the controller to control devices in its domain, it needs to know about the devices, their capabilities, reachability, and so on. This information is stored and managed by the **Topology Manager**. The other components like ARP handler, Host Tracker, Device Manager, and Switch Manager help in generating the topology database for the Topology Manager.

For a more detailed overview of the OpenDaylight controller, see the *OpenDaylight Developer Guide*.

Using the OpenDaylight User Interface (DLUX)

This section introduces you to the OpenDaylight User Experience (DLUX) application.

Getting Started with DLUX

DLUX provides a number of different Karaf features, which you can enable and disable separately. They are:

- odl-dlux-core
- odl-dluxapps-nodes
- odl-dluxapps-topology
- odl-dluxapps-yangui
- odl-dluxapps-yangvisualizer
- odl-dluxapps-yangman

Logging In

To log in to DLUX, after installing the application:

1. Open a browser and enter the login URL http://<your-karaf-ip>:8181/index.html in your browser (Chrome is recommended).
2. Login to the application with your username and password credentials.

Note: OpenDaylight's default credentials are *admin* for both the username and password.

Working with DLUX

After you login to DLUX, if you enable only odl-dlux-core feature, you will see only topology application available in the left pane.

Note: To make sure topology displays all the details, enable the odl-l2switch-switch feature in Karaf.

DLUX has other applications such as node, yang UI and those apps won't show up, until you enable their features odl-dluxapps-nodes and odl-dluxapps-yangui respectively in the Karaf distribution.

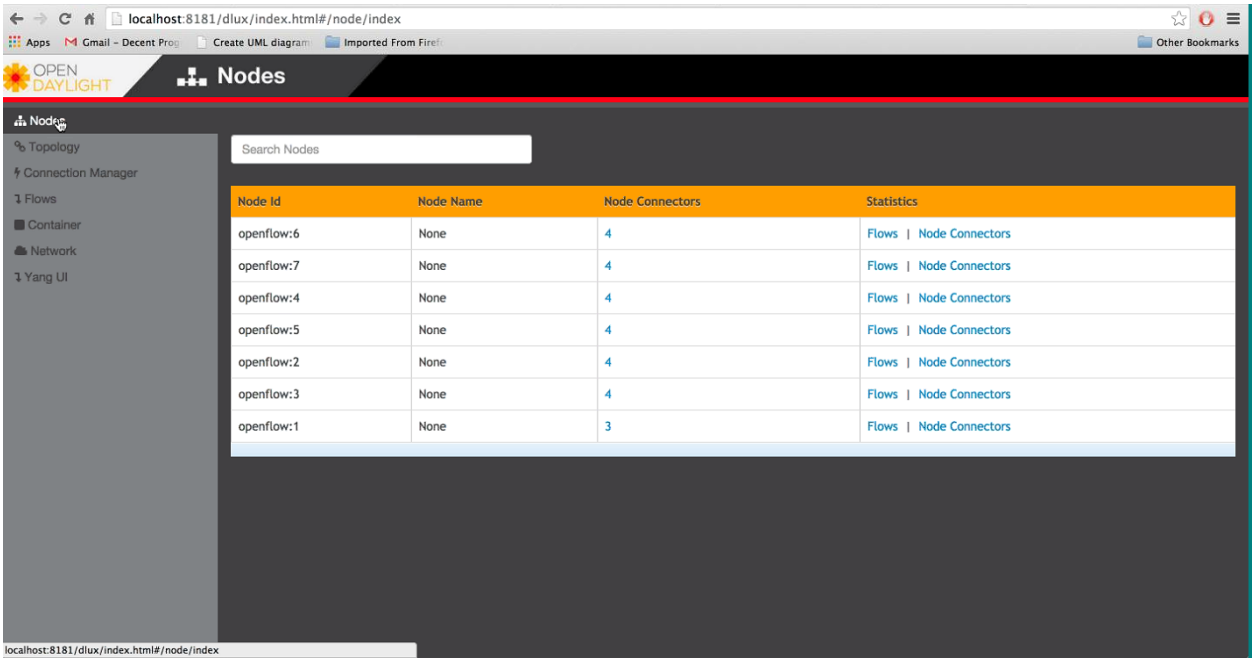


Fig. 1.10: DLUX Modules

Note: If you install your application in dlux, they will also show up on the left hand navigation after browser page refresh.

Viewing Network Statistics

The **Nodes** module on the left pane enables you to view the network statistics and port information for the switches in the network.

To use the **Nodes** module:

1. Select **Nodes** on the left pane. The right pane displays atable that lists all the nodes, node connectors and the statistics.
2. Enter a node ID in the **Search Nodes** tab to search by node connectors.
3. Click on the **Node Connector** number to view details such as port ID, port name, number of ports per switch, MAC Address, and so on.
4. Click **Flows** in the Statistics column to view Flow Table Statistics for the particular node like table ID, packet match, active flows and so on.
5. Click **Node Connectors** to view Node Connector Statistics for the particular node ID.

Viewing Network Topology

The Topology tab displays a graphical representation of network topology created.

Note: DLUX does not allow for editing or adding topology information. The topology is generated and edited in

other modules, e.g., the OpenFlow plugin. OpenDaylight stores this information in the MD-SAL datastore where DLUX can read and display it.

To view network topology:

1. Select **Topology** on the left pane. You will view the graphical representation on the right pane. In the diagram blue boxes represent the switches, the black represents the hosts available, and lines represents how the switches and hosts are connected.
2. Hover your mouse on hosts, links, or switches to view source and destination ports.
3. Zoom in and zoom out using mouse scroll to verify topology for larger topologies.

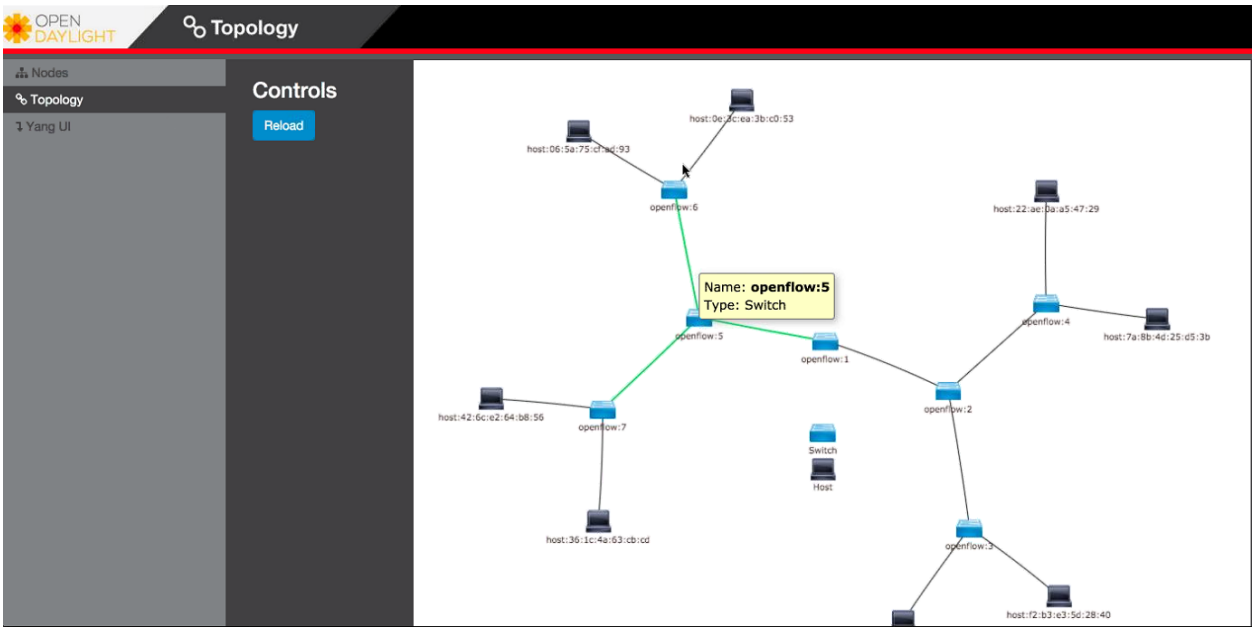


Fig. 1.11: Topology Module

Interacting with the YANG-based MD-SAL datastore

The **Yang UI** module enables you to interact with the YANG-based MD-SAL datastore. For more information about YANG and how it interacts with the MD-SAL datastore, see the *Controller* and *YANG Tools* section of the *OpenDaylight Developer Guide*.

To use Yang UI:

1. Select **Yang UI** on the left pane. The right pane is divided in two parts.
2. The top part displays a tree of APIs, subAPIs, and buttons to call possible functions (GET, POST, PUT, and DELETE).

   Note: every subAPI can call every function. For example, subAPIs in the *operational* store have GET functionality only.

   Inputs can be filled from OpenDaylight when existing data from OpenDaylight is displayed or can be filled by user on the page and sent to OpenDaylight.

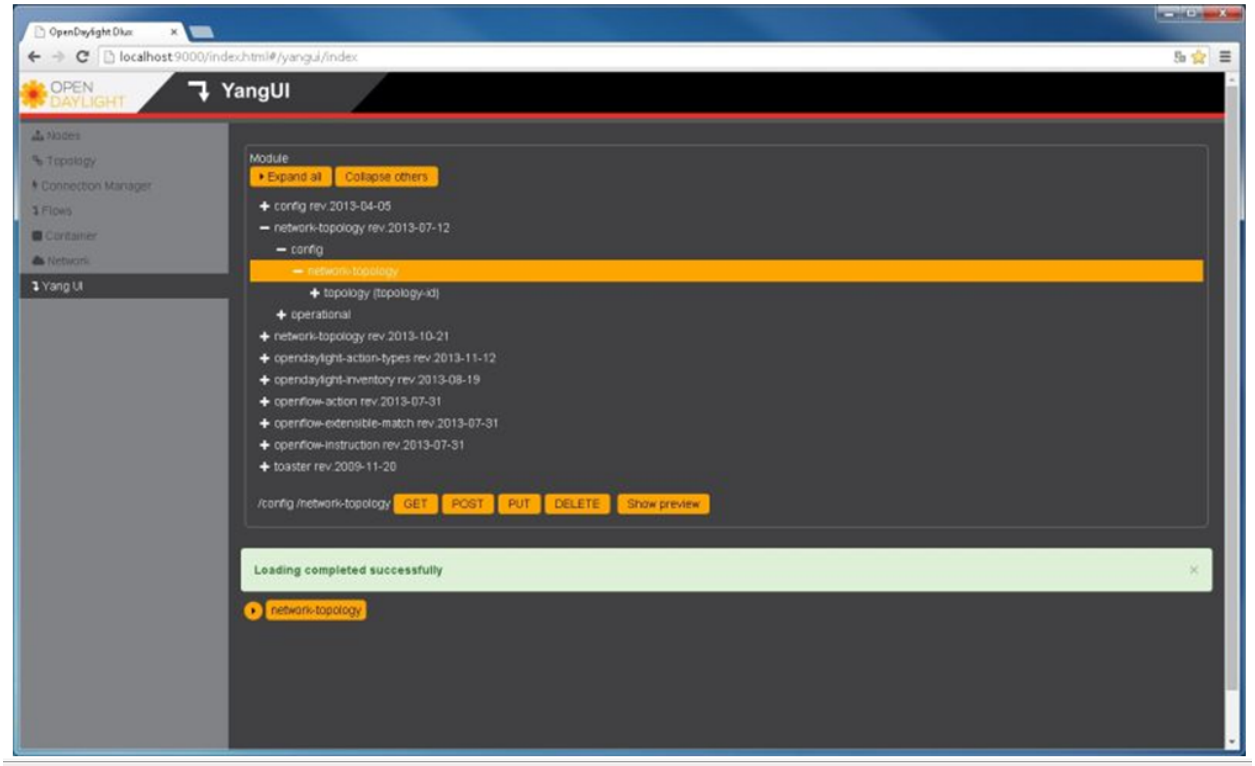


Fig. 1.12: Yang UI

Buttons under the API tree are variable. It depends on subAPI specifications. Common buttons are:

- GET to get data from OpenDaylight,
- PUT and POST for sending data to OpenDaylight for saving
- DELETE for sending data to OpenDaylight for deleting.

  You must specify the xpath for all these operations. This path is displayed in the same row before buttons and it may include text inputs for specific path element identifiers.

3. The bottom part of the right pane displays inputs according to the chosen subAPI.
   - Lists are handled as a special case. For example, a device can store multiple flows. In this case "flow" is name of the list and every list element is identified by a unique key value. Elements of a list can, in turn, contain other lists.
   - In Yang UI, each list element is rendered with the name of the list it belongs to, its key, its value, and a button for removing it from the list.
4. After filling in the relevant inputs, click the **Show Preview** button under the API tree to display request that will be sent to OpenDaylight. A pane is displayed on the right side with text of request when some input is filled.

Displaying Topology on the Yang UI

To display topology:

1. Select subAPI network-topology <topology revision number> == > operational == > network-topology.
2. Get data from OpenDaylight by clicking on the "GET" button.

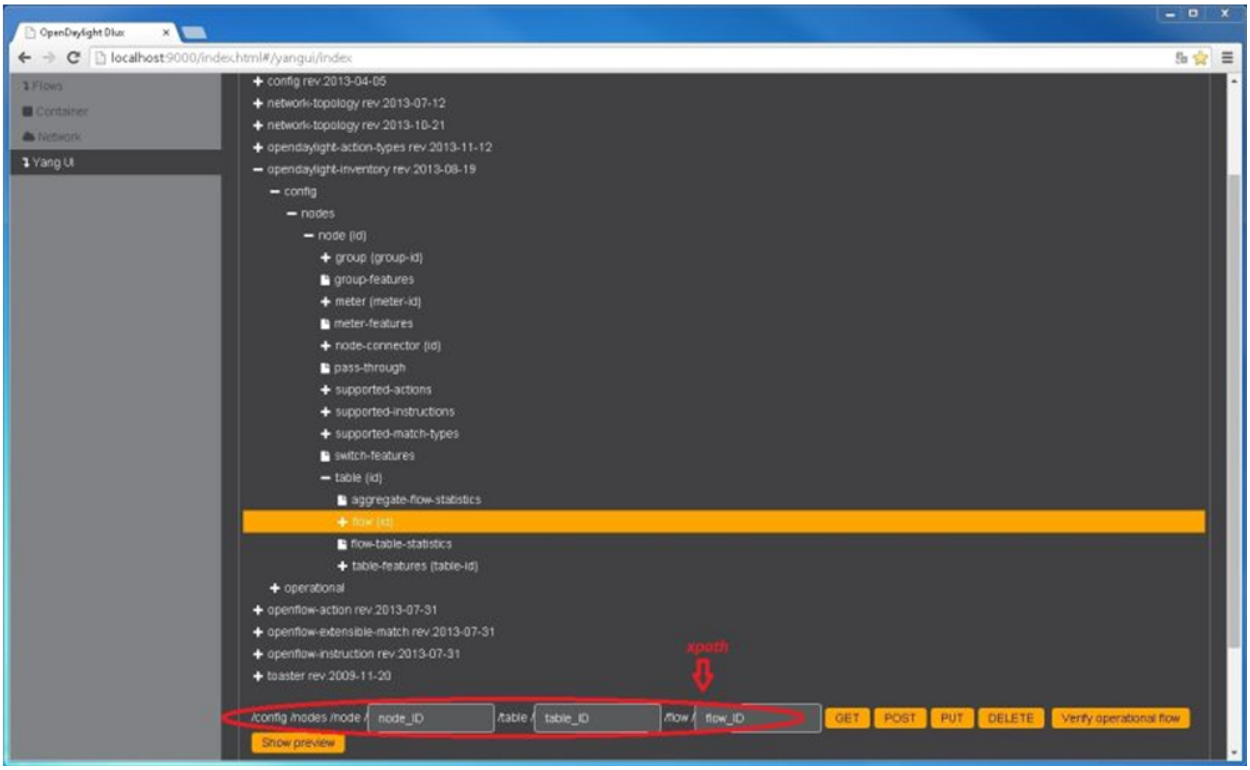


Fig. 1.13: Yang API Specification

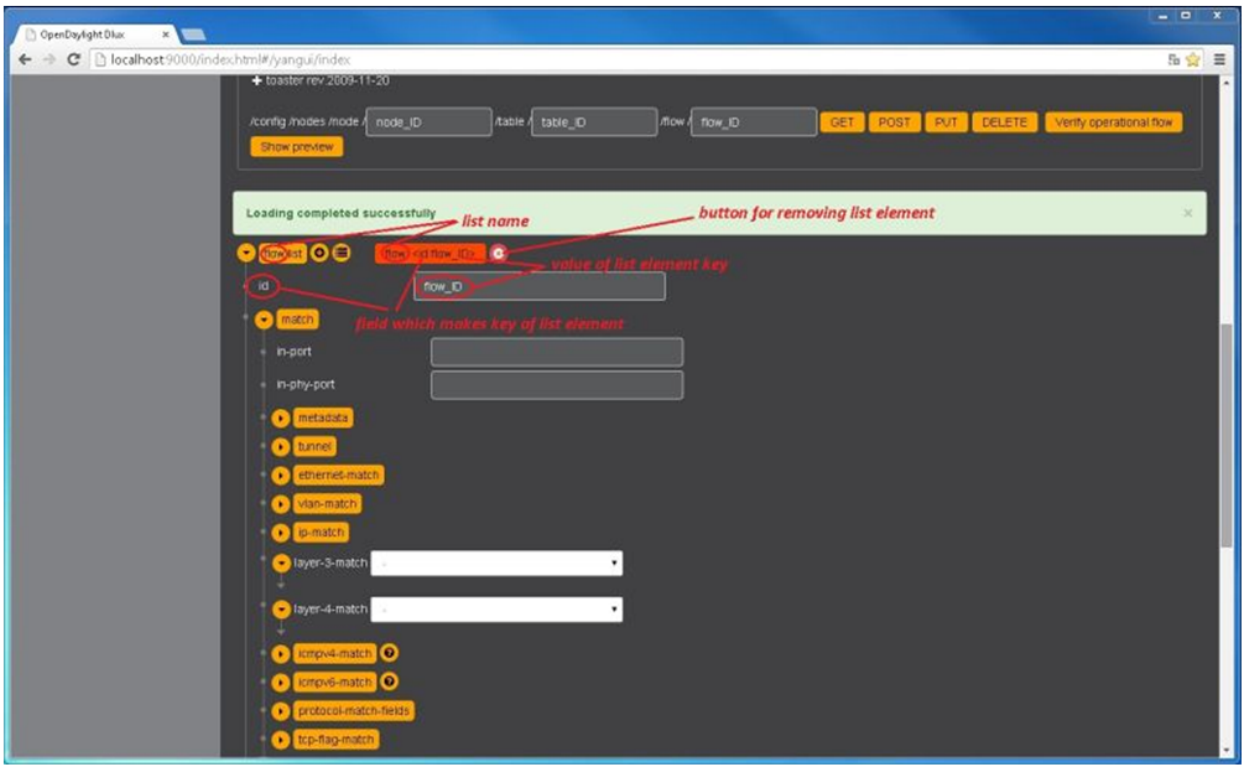


Fig. 1.14: Yang UI API Specification

3. Click **Display Topology**.

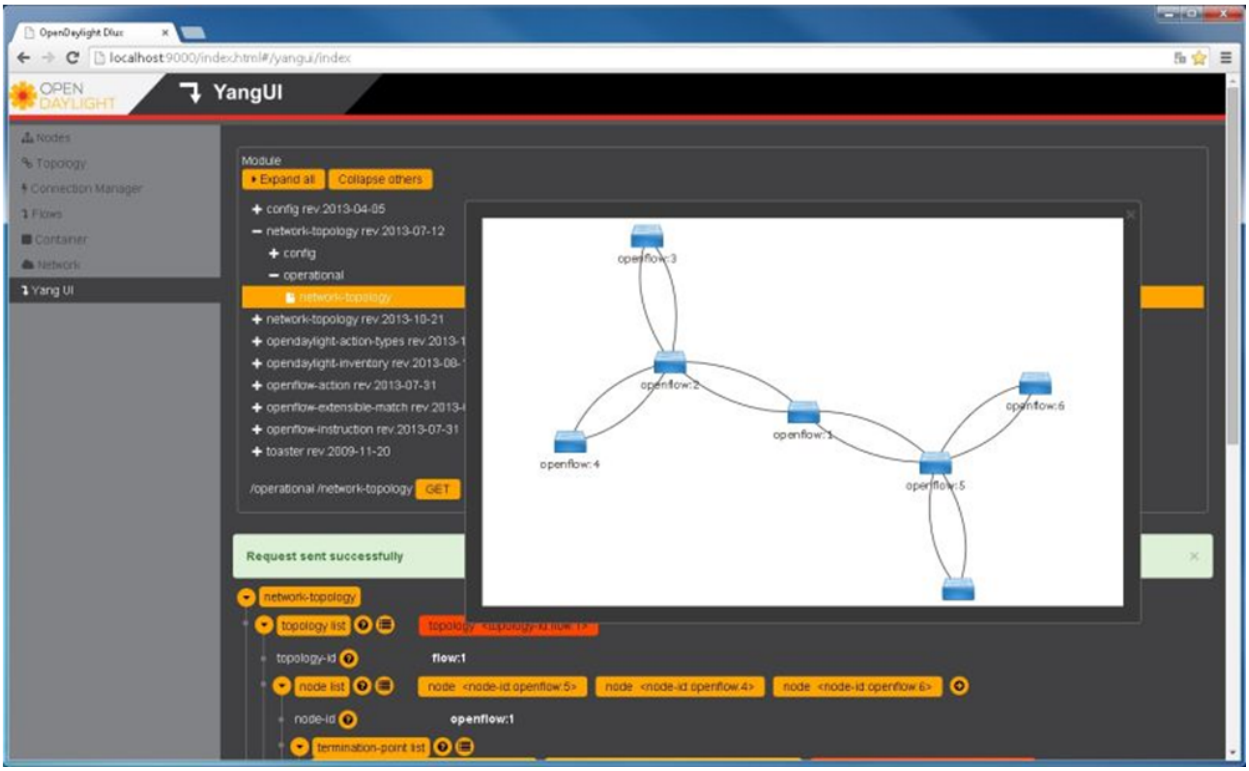


Fig. 1.15: DLUX Yang Topology

Configuring List Elements on the Yang UI

Lists in Yang UI are displayed as trees. To expand or collapse a list, click the arrow before name of the list. To configure list elements in Yang UI:

1. To add a new list element with empty inputs use the plus icon-button **+** that is provided after list name.
2. To remove several list elements, use the **X** button that is provided after every list element.
3. In the YANG-based data store all elements of a list must have a unique key. If you try to assign two or more elements the same key, a warning icon **!** is displayed near their name buttons.
4. When the list contains at least one list element, after the **+** icon, there are buttons to select each individual list element. You can choose one of them by clicking on it. In addition, to the right of the list name, there is a button which will display a vertically scrollable pane with all the list elements.

Running XSQL Console Commands and Queries

XSQL Overview

XSQL is an XML-based query language that describes simple stored procedures which parse XML data, query or update database tables, and compose XML output. XSQL allows you to query tree models like a sequential database. For example, you could run a query that lists all of the ports configured on a particular module and their attributes.

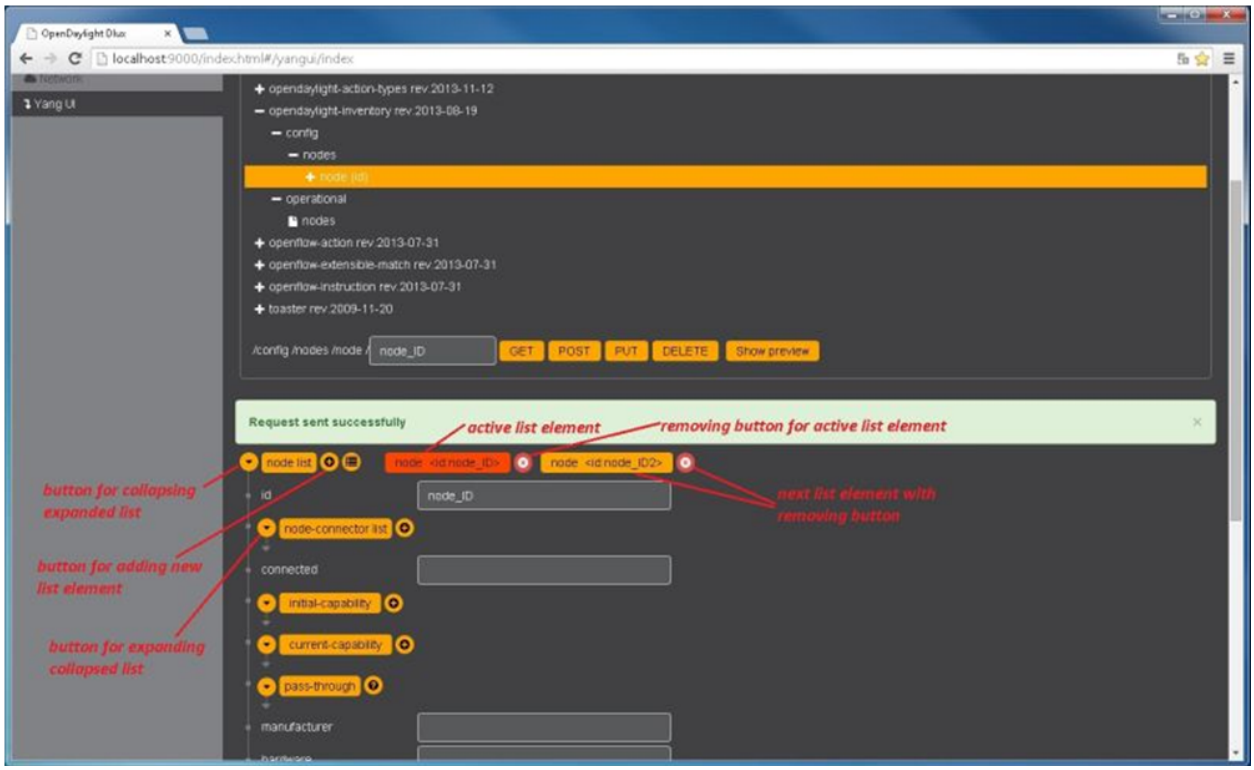


Fig. 1.16: DLUX List Elements

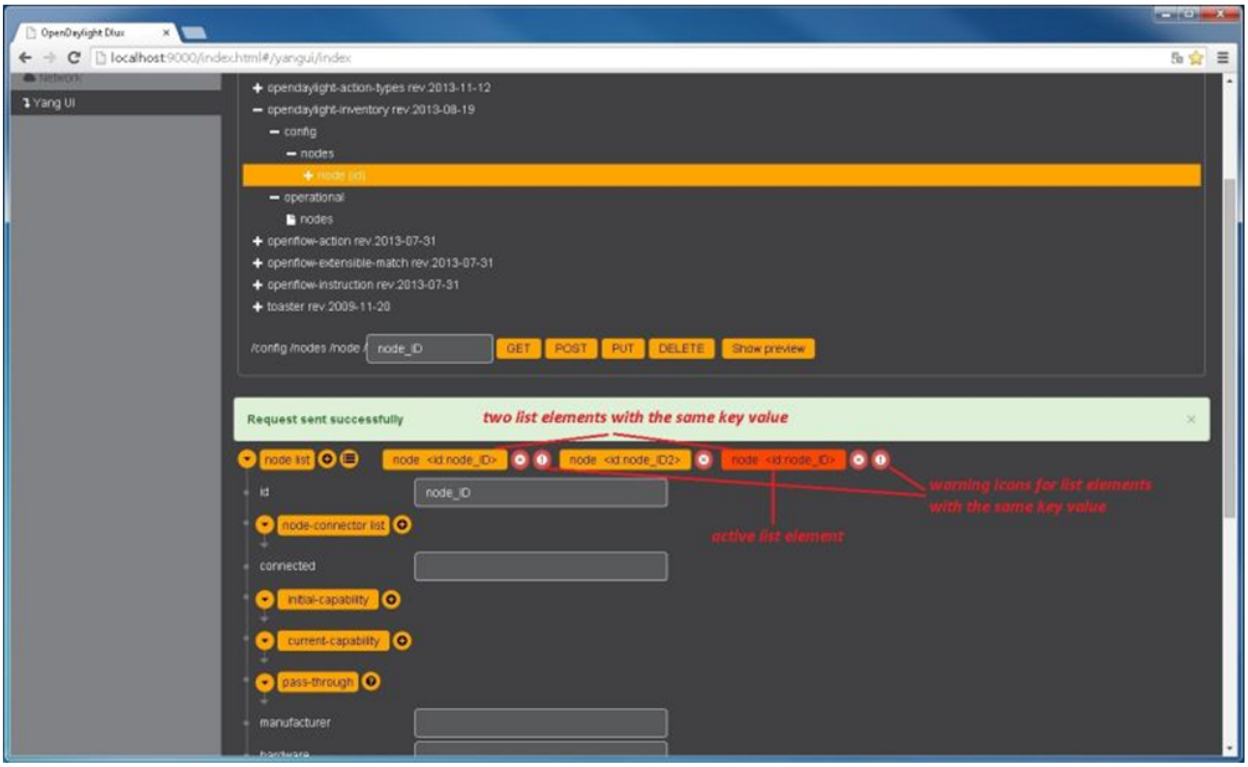


Fig. 1.17: DLUX List Warnings

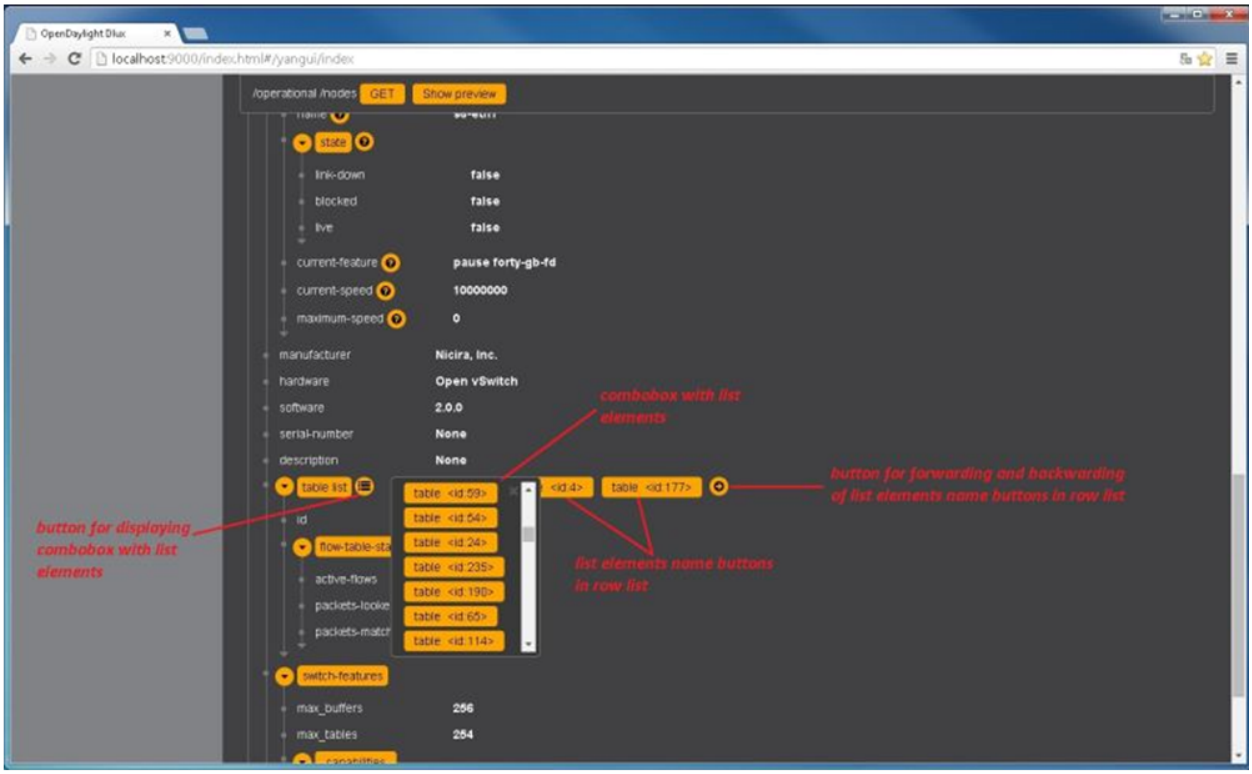


Fig. 1.18: DLUX List Button1

The following sections cover the XSQL installation process, supported XSQL commands, and the way to structure queries.

Installing XSQL

To run commands from the XSQL console, you must first install XSQL on your system:

1. Navigate to the directory in which you unzipped OpenDaylight
2. Start Karaf:

```
./bin/karaf
```

3. Install XSQL:

```
feature:install odl-mdsal-xsql
```

XSQL Console Commands

To enter a command in the XSQL console, structure the command as follows: **odl:xsql** <*XSQL command*>

The following table describes the commands supported in this OpenDaylight release.

Command	Description
r	Repeats the last command you executed.
list vtables	Lists the schema node containers that are currently installed. Whenever an OpenDaylight module is installed, its YANG model is placed in the schema context. At that point, the XSQL receives a notification, confirms that the module's YANG model resides in the schema context and then maps the model to XSQL by setting up the necessary vtables and vfields. This command is useful when you need to determine vtable information for a query.
list vfields <*vtable name*>	Lists the vfields present in a specific vtable. This command is useful when you need to determine vfields information for a query.
jdbc <*ip address*>	When the ODL server is behind a firewall, and the JDBC client cannot connect to the JDBC server, run this command to start the client as a server and establish a connection.
exit	Closes the console.
tocsv	Enables or disables the forwarding of query output as a .csv file.
filename <*filename*>	Specifies the .tocsv file to which the query data is exported. If you do not specify a value for this option when the toccsv option is enabled, the filename for the query data file is generated automatically.

Table: Supported XSQL Console Commands

XSQL Queries

You can run a query to extract information that meets the criteria you specify using the information provided by the **list vtables** and **list vfields** <*vtable name*> commands. Any query you run should be structured as follows:

select <*vfields you want to search for, separated by a comma and a space*> **from** <*vtables you want to search in, separated by a comma and a space*> **where** <*criteria*> ***<criteria operator>**;*

For example, if you want to search the nodes/node ID field in the nodes/node-connector table and find every instance of the Hardware-Address object that contains *BA* in its text string, enter the following query:

```
select nodes/node.ID from nodes/node-connector where Hardware-Address like '%BA%';
```

The following criteria operators are supported:

Criteria Operators	Description
=	Lists results that equal the value you specify.
!=	Lists results that do not equal the value you specify.
like	Lists results that contain the substring you specify. For example, if you specify **like %BC%**, every string that contains that particular substring is displayed.
<	Lists results that are less than the value you specify.
>	Lists results that are more than the value you specify.
and	Lists results that match both values you specify.
or	Lists results that match either of the two values you specify.
>=	Lists results that are more than or equal to the value you specify.
	Lists results that are less than or equal to the value you specify.
is null	Lists results for which no value is assigned.
not null	Lists results for which any value is assigned.
skip	Use this operator to list matching results from a child node, even if its parent node does not meet the specified criteria. See the following example for more information.

Table: Supported XSQL Query Criteria Operators

Example: Skip Criteria Operator

If you are looking at the following structure and want to determine all of the ports that belong to a YY type module:

- Network Element 1
  - Module 1, Type XX
    - Module 1.1, Type YY
      - Port 1
      - Port 2
  - Module 2, Type YY
    - Port 1
    - Port 2

If you specify **Module.Type=*YY*** in your query criteria, the ports associated with module 1.1 will not be returned since its parent module is type XX. Instead, enter **Module.Type=*YY* or skip Module!=*YY***. This tells XSQL to disregard any parent module data that does not meet the type YY criteria and collect results for any matching child modules. In this example, you are instructing the query to skip module 1 and collect the relevant data from module 1.1.

1.3.2 Project-specific User Guides

ALTO User Guide

Overview

The ALTO project is aimed to provide support for **Application Layer Traffic Optimization** services defined in RFC 7285 in OpenDaylight.

This user guide will introduce the three basic services (namely `simple-ird`, `manual-maps` and `host-tracker`) which are implemented since the Beryllium release, and give instructions on how to configure them to provide corresponding ALTO services.

A new feature named `simple-pce` (**Simple Path Computation Engine**) is added into Boron release as an ALTO extension service.

How to Identify ALTO Resources

Each ALTO resource can be uniquely identified by a tuple . For each resource, a *version-tag* is used to support historical look-ups.

The formats of *resource-id* and *version-tag* are defined in section 10.2 and section 10.3 respectively. The *context-id* is not part of the protocol and we choose the same format as a *universal unique identifier* (UUID) which is defined in RFC 4122.

A context is like a namespace for ALTO resources, which eliminates *resource-id* collisions. For simplicity, we also provide a default context with the id **000000000000-0000-0000-0000-00000000**.

How to Use Simple IRD

The simple IRD feature provides a simple *information resource directory* (IRD) service defined in RFC 7285.

Install the Feature

To enable simple IRD, run the following command in the karaf CLI:

```
karaf > feature:install odl-alto-simpleird
```

After the feature is successfully installed, a special context will be created for all simple IRD resources. The id for this context can be seen by executing the following command in a terminal:

```
curl -X GET -u admin:admin http://localhost:8181/restconf/operational/alto-simple-
↪ird:information/
```

Create a new IRD

To create a new IRD resource, two fields MUST be provided:

- Field **instance-id**: the *resource-id* of the IRD resource;
- Field **entry-context**: the context-id for non-IRD entries managed by this IRD resource.

Using the following script, one can create an empty IRD resource:

```
#!/bin/bash
# filename: ird-create
INSTANCE_ID=$1
if [ $2 ]; then
    CONTEXT_ID=$2
else
    CONTEXT_ID="00000000-0000-0000-0000-000000000000"
fi
URL="`http://localhost:8181/restconf/config/alto-simple-ird:ird-instance-
↪configuration/"$INSTANCE_ID"/[`http://localhost:8181/restconf/config/alto-simple-
↪ird:ird-instance-configuration/"$INSTANCE_ID"/`]`"
DATA=$(cat template \
  | sed 's/\$1/'$CONTEXT_ID'/g' \
  | sed 's/\$2/'$INSTANCE_ID'/g')
curl -4 -D - -X PUT -u admin:admin \
  -H "Content-Type: application/json" -d "$(echo $DATA)"\
  $URL
```

For example, the following command will create a new IRD named *ird* which can accept entries with the default context-id:

```
$ ./ird-create ird 000000000000-0000-0000-0000-00000000
```

And below is the what the template file looks like:

```
{
    "ird-instance-configuration": {
        "entry-context": "/alto-resourcepool:context[alto-resourcepool:context-id='$1
↪']",
        "instance-id": "$2"
    }
}
```

Remove an IRD

To remove an existing IRD (and all the entries in it), one can use the following command in a terminal:

```
curl -X DELETE -u admin:admin http://localhost:8181/restconf/config/alto-simple-
↪ird:ird-instance-configuration/$INSTANCE_ID
```

Add a new entry

There are several ways to add entries to an IRD and in this section we introduce only the simplest method. Using the following script, one can add a new entry to the target IRD.

For each new entry, four parameters MUST be provided:

- Parameter *ird-id*: the *resource-id* of the target IRD;
- Parameter *entry-id*: the *resource-id* of the ALTO service to be added;
- Parameter *context-id*: the *context-id* of the ALTO service to be added, which MUST be identical to the target IRD's *entry-context*;
- Parameter *location*: either a URI or a relative path to the ALTO service.

The following script can be used to add one entry to the target IRD, where the relative path is used:

```
#!/bin/bash
# filename: ird-add-entry
IRD_ID=$1
ENTRY_ID=$2
CONTEXT_ID=$3
BASE_URL=$4
URL="`http://localhost:8181/restconf/config/alto-simple-ird:ird-instance-
↪configuration/"$IRD_ID"/ird-configuration-entry/"$ENTRY_ID"/"
DATA=$(cat template \
  | sed 's/\$1/'$ENTRY_ID'/g' \
  | sed 's/\$2/'$CONTEXT_ID'/g' \
  | sed 's/\$3/'$BASE_URL'/g' )
curl -4 -D - -X PUT -u admin:admin \
  -H "Content-Type: application/json" -d "$(echo $DATA)" \
  $URL
```

For example, the following command will add a new resource named *networkmap*, whose context-id is the default context-id and the base URL is /alto/networkmap, to the IRD named *ird*:

```
$ ./ird-add-entry ird networkmap 000000000000-0000-0000-0000-00000000 /alto/networkmap
```

And below is the template file:

```
{
    "ird-configuration-entry": {
        "entry-id": "$1",
        "instance": "/alto-resourcepool:context[alto-resourcepool:context-id='$2']/
↪alto-resourcepool:resource[alto-resourcepool:resource-id='$1']",
        "path": "$3/$1"
    }
}
```

Remove an entry

To remove an entry from an IRD, one can use the following one-line command:

```
curl -X DELETE -u admin:admin http://localhost:8181/restconf/config/alto-simple-
↪ird:ird-instance-configuration/$IRD_ID/ird-configuration-entry/$ENTRY_ID/
```

How to Use Host-tracker-based ECS

As a real instance of ALTO services, ***alto-hosttracker*** reads data from ***l2switch*** and generates a network map with resource id ***hosttracker-network-map*** and a cost map with resource id ***hostracker-cost-map***. It can only work with OpenFlow-enabled networks.

After installing the ***odl-alto-hosttracker*** feature, the corresponding network map and cost map will be inserted into the data store.

Managing Resource with `alto-resourcepool`

After installing `odl-alto-release` feature in Karaf, `alto-resourcepool` feature will be installed automatically. And you can manage all resources in ALTO via RESTCONF APIs provided by `alto-resourcepool`.

With the example bash script below you can get any resource infomation in a given context.

```
#!/bin/bash
RESOURCE_ID=$1
if [ $2 ] ; then
    CONTEXT_ID=$2
else
    CONTEXT_ID="00000000-0000-0000-0000-000000000000"
fi
URL="http://localhost:8181/restconf/operational/alto-resourcepool:context/"$CONTEXT_ID
↪"/alto-resourcepool:resource/"$RESOURCE_ID
curl -X GET -u admin:admin $URL | python -m json.tool | sed -n '/default-tag/p' | sed
↪'s/.*:.*\"\(.*\)\".*/\1/g'
```

Manual Configuration

Using RESTCONF API

After installing `odl-alto-release` feature in Karaf, it is possible to manage network-maps and cost-maps using RESTCONF. Take a look at all the operations provided by `resource-config` at the API service page which can be found at `http://localhost:8181/apidoc/explorer/index.html`.

The easiest method to operate network-maps and cost-maps is to modify data broker via RESTCONF API directly.

Using RPC

The `resource-config` package also provides a query RPC to config the resources. You can CREATE, UPDATE and DELETE **network-maps** and **cost-maps** via query RPC.

Simple Path Computation Engine

The `simple-pce` module provides a simple path computation engine for ALTO and other projects. It supports basic CRUD (create, read, update, delete) operations to manage L2 and L3 routing with/without rate limitation. This module is an independent feature, so you can follow the instruction below to install it independently.

```
karaf > feature:install odl-alto-extenstion
```

Note: The rate limitation meter requires OpenFlow 1.3 support.

Basic Usage with RESTCONF API

You can use the simple path computation engine with RESTCONF API, which is defined in the YANG model here.

Use Case

Server Selection

One of the key use case for ALTO is server selection. For example, a client (with IP address 10.0.0.1) sends a data transferring request to Data Transferring Service (DTS). And there are three data replica servers (with IP address 10.60.0.1, 10.60.0.2 and 10.60.0.3) which can response the request. In this case, DTS can send a query request to ALTO server to make server selection decision.

Following is an example ALTO query:

```
POST /alto/endpointcost HTTP/1.1
Host: localhost:8080
Content-Type: application/alto-endpointcostparams+json
Accept: application/alto-endpointcost+json,application/alto-error+json
{
  "cost-type": {
    "cost-mode": "ordinal",
    "cost-metric": "hopcount"
  },
  "endpoints": {
    "srcs": [ "ipv4:10.0.0.1" ],
    "dsts": [
      "ipv4:10.60.0.1",
      "ipv4:10.60.0.2",
      "ipv4:10.60.0.3"
  ]
  }
}
```

Authentication, Authorization and Accounting (AAA) Services

Overview

Authentication, Authorization and Accounting (AAA) is a term for a framework controlling access to resources, enforcing policies to use those resources and auditing their usage. These processes are the fundamental building

blocks for effective network management and security.

Authentication provides a way of identifying a user, typically by having the user enter a valid user name and valid password before access is granted. The process of authentication is based on each user having a unique set of criteria for gaining access. The AAA framework compares a user's authentication credentials with other user credentials stored in a database. If the credentials match, the user is granted access to the network. If the credentials don't match, authentication fails and access is denied.

Authorization is the process of finding out what an authenticated user is allowed to do within the system, which tasks can do, which API can call, etc. The authorization process determines whether the user has the authority to perform such actions.

Accounting is the process of logging the activity of an authenticated user, for example, the amount of data a user has sent and/or received during a session, which APIs called, etc.

Terms And Definitions

AAA Authentication, Authorization and Accounting.

Token A claim of access to a group of resources on the controller.

Domain A group of resources, direct or indirect, physical, logical, or virtual, for the purpose of access control.

User A person who either owns or has access to a resource or group of resources on the controller.

Role Opaque representation of a set of permissions, which is merely a unique string as admin or guest.

Credential Proof of identity such as user name and password, OTP, biometrics, or others.

Client A service or application that requires access to the controller.

Claim A data set of validated assertions regarding a user, e.g. the role, domain, name, etc.

Grant It is the entity associating a user with his role and domain.

IdP Identity Provider.

TLS Transport Layer Security

CLI Command Line Interface

Security Framework for AAA services

Since Boron release, the OpenDaylight's AAA services are based on the Apache Shiro Java Security Framework. The main configuration file for AAA is located at "etc/shiro.ini" relative to the OpenDaylight Karaf home directory.

How to enable AAA

AAA is enabled through installing the odl-aaa-shiro feature. The vast majority of OpenDaylight's northbound APIs (and all RESTCONF APIs) are protected by AAA by default when installing the +odl-restconf+ feature, since the odl-aaa-shiro is automatically installed as part of them. In the cases that APIs are *not* protected by AAA, this will be noted in the per-project release notes.

How to disable AAA

Edit the "etc/shiro.ini" file and replace the following:

```
/** = authcBasic
```

with

```
/** = anon
```

Then restart the Karaf process.

AAA Realms

AAA plugin utilizes the Shiro Realms to support pluggable authentication & authorization schemes. There are two parent types of realms:

- AuthenticatingRealm
  - Provides no Authorization capability.
  - Users authenticated through this type of realm are treated equally.
- AuthorizingRealm
  - AuthorizingRealm is a more sophisticated AuthenticatingRealm, which provides the additional mechanisms to distinguish users based on roles.
  - Useful for applications in which roles determine allowed capabilities.

OpenDaylight contains five implementations:

- TokenAuthRealm
  - An AuthorizingRealm built to bridge the Shiro-based AAA service with the h2-based AAA implementation.
  - Exposes a RESTful web service to manipulate IdM policy on a per-node basis. If identical AAA policy is desired across a cluster, the backing data store must be synchronized using an out of band method.
  - A python script located at "etc/idmtool" is included to help manipulate data contained in the TokenAuthRealm.
  - Enabled out of the box. This is the realm configured by default.
- ODLJndiLdapRealm
  - An AuthorizingRealm built to extract identity information from IdM data contained on an LDAP server.
  - Extracts group information from LDAP, which is translated into OpenDaylight roles.
  - Useful when federating against an existing LDAP server, in which only certain types of users should have certain access privileges.
  - Disabled out of the box.
- ODLJndiLdapRealmAuthNOnly
  - The same as ODLJndiLdapRealm, except without role extraction. Thus, all LDAP users have equal authentication and authorization rights.
  - Disabled out of the box.
- ODLActiveDirectoryRealm
  - Wraps the generic ActiveDirectoryRealm provided by Shiro. This allows for enhanced logging as well as isolation of all realms in a single package, which enables easier import by consuming servlets.

- KeystoneAuthRealm
  - This realm authenticates OpenDaylight users against the OpenStack's Keystone server.
  - Disabled out of the box.

Note: More than one Realm implementation can be specified. Realms are attempted in order until authentication succeeds or all realm sources are exhausted. Edit the **securityManager.realms = $tokenAuthRealm** property in shiro.ini and add all the realms needed separated by commas.

TokenAuthRealm

How it works

The TokenAuthRealm is the default Authorization Realm deployed in OpenDaylight. TokenAuthRealm uses a direct authentication mechanism as shown in the following picture:

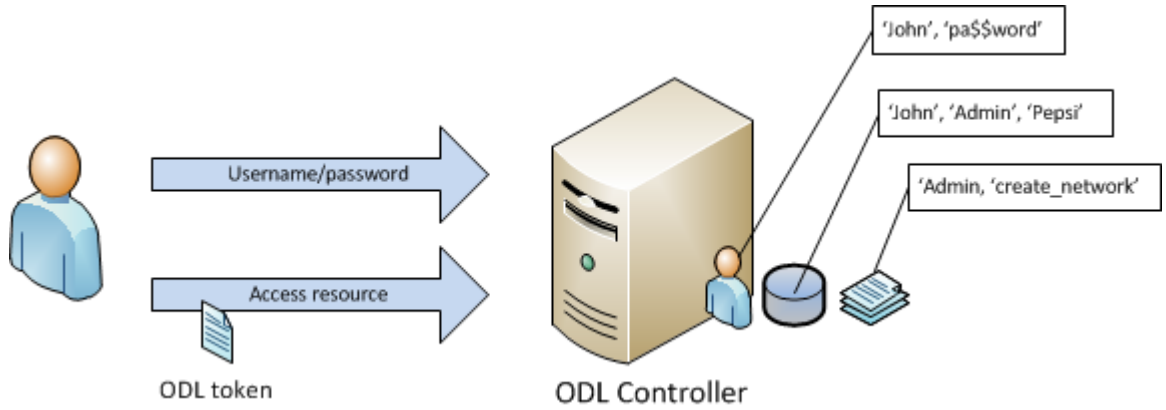


Fig. 1.19: TokenAuthRealm direct authentication mechanism

A user presents some credentials (e.g., username/password) directly to the OpenDaylight controller token endpoint /oauth2/token and receives an access token, which then can be used to access protected resources on the controller.

Configuring TokenAuthRealm

The TokenAuthRealm stores IdM data in an h2 database on each node. Thus, configuration of a cluster currently requires configuring the desired IdM policy on each node. There are two supported methods to manipulate the TokenAuthRealm IdM configuration:

- idmtool configuration tool
- RESTful Web Service configuration

Idmtool

A utility script located at "etc/idmtool" is used to manipulate the TokenAuthRealm IdM policy. idmtool assumes a single domain, the default one (sdn), since multiple domains are not supported in the Boron release. General usage information for idmtool is derived through issuing the following command:

```
$ python etc/idmtool -h
usage: idmtool [-h] [--target-host TARGET_HOST]
               user
               {list-users,add-user,change-password,delete-user,list-domains,list-
↪roles,add-role,delete-role,add-grant,get-grants,delete-grant}
               ...

positional arguments:
  user                  username for BSC node
  {list-users,add-user,change-password,delete-user,list-domains,list-roles,add-role,
↪delete-role,add-grant,get-grants,delete-grant}
                        sub-command help
    list-users          list all users
    add-user            add a user
    change-password     change a password
    delete-user         delete a user
    list-domains        list all domains
    list-roles          list all roles
    add-role            add a role
    delete-role         delete a role
    add-grant           add a grant
    get-grants          get grants for userid on sdn
    delete-grant        delete a grant

optional arguments:
  -h, --help            show this help message and exit
  --target-host TARGET_HOST
                        target host node
```

Add a user

```
python etc/idmtool admin add-user newUser
Password:
Enter new password:
Re-enter password:
add_user(admin)

command succeeded!

json:
{
    "description": "",
    "domainid": "sdn",
    "email": "",
    "enabled": true,
    "name": "newUser",
    "password": "**********",
    "salt": "**********",
    "userid": "newUser@sdn"
}
```

Note: AAA redacts the password and salt fields for security purposes.

Delete a user

```
$ python etc/idmtool admin delete-user newUser@sdn
Password:
delete_user(newUser@sdn)

command succeeded!
```

List all users

```
$ python etc/idmtool admin list-users
Password:
list_users

command succeeded!

json:
{
    "users": [
        {
            "description": "user user",
            "domainid": "sdn",
            "email": "",
            "enabled": true,
            "name": "user",
            "password": "**********",
            "salt": "**********",
            "userid": "user@sdn"
        },
        {
            "description": "admin user",
            "domainid": "sdn",
            "email": "",
            "enabled": true,
            "name": "admin",
            "password": "**********",
            "salt": "**********",
            "userid": "admin@sdn"
        }
    ]
}
```

Change a user's password

```
$ python etc/idmtool admin change-password admin@sdn
Password:
Enter new password:
Re-enter password:
change_password(admin)

command succeeded!

json:
```

```
{
    "description": "admin user",
    "domainid": "sdn",
    "email": "",
    "enabled": true,
    "name": "admin",
    "password": "**********",
    "salt": "**********",
    "userid": "admin@sdn"
}
```

Add a role

```
$ python etc/idmtool admin add-role network-admin
Password:
add_role(network-admin)

command succeeded!

json:
{
    "description": "",
    "domainid": "sdn",
    "name": "network-admin",
    "roleid": "network-admin@sdn"
}
```

Delete a role

```
$ python etc/idmtool admin delete-role network-admin@sdn
Password:
delete_role(network-admin@sdn)

command succeeded!
```

List all roles

```
$ python etc/idmtool admin list-roles
Password:
list_roles

command succeeded!

json:
{
    "roles": [
        {
            "description": "a role for admins",
            "domainid": "sdn",
            "name": "admin",
            "roleid": "admin@sdn"
```

```
        },
        {
            "description": "a role for users",
            "domainid": "sdn",
            "name": "user",
            "roleid": "user@sdn"
        }
    ]
}
```

List all domains

```
$ python etc/idmtool admin list-domains
Password:
list_domains

command succeeded!

json:
{
    "domains": [
        {
            "description": "default odl sdn domain",
            "domainid": "sdn",
            "enabled": true,
            "name": "sdn"
        }
    ]
}
```

Add a grant

```
$ python etc/idmtool admin add-grant user@sdn admin@sdn
Password:
add_grant(userid=user@sdn,roleid=admin@sdn)

command succeeded!

json:
{
    "domainid": "sdn",
    "grantid": "user@sdn@admin@sdn@sdn",
    "roleid": "admin@sdn",
    "userid": "user@sdn"
}
```

Delete a grant

```
$ python etc/idmtool admin delete-grant user@sdn admin@sdn
Password:
http://localhost:8181/auth/v1/domains/sdn/users/user@sdn/roles/admin@sdn
```

```
delete_grant(userid=user@sdn,roleid=admin@sdn)

command succeeded!
```

Get grants for a user

```
python etc/idmtool admin get-grants admin@sdn
Password:
get_grants(admin@sdn)

command succeeded!

json:
{
    "roles": [
        {
            "description": "a role for users",
            "domainid": "sdn",
            "name": "user",
            "roleid": "user@sdn"
        },
        {
            "description": "a role for admins",
            "domainid": "sdn",
            "name": "admin",
            "roleid": "admin@sdn"
        }
    ]
}
```

Configuration using the RESTful Web Service

The TokenAuthRealm IdM policy is fully configurable through a RESTful web service. Full documentation for manipulating AAA IdM data is located online (https://wiki.opendaylight.org/images/0/00/AAA_Test_Plan.docx), and a few examples are included in this guide:

Get All Users

```
curl -u admin:admin http://localhost:8181/auth/v1/users
OUTPUT:
{
    "users": [
        {
            "description": "user user",
            "domainid": "sdn",
            "email": "",
            "enabled": true,
            "name": "user",
            "password": "**********",
            "salt": "**********",
            "userid": "user@sdn"
        },
```

```
        {
            "description": "admin user",
            "domainid": "sdn",
            "email": "",
            "enabled": true,
            "name": "admin",
            "password": "**********",
            "salt": "**********",
            "userid": "admin@sdn"
        }
    ]
}
```

Create a User

```
curl -u admin:admin -X POST -H "Content-Type: application/json" --data-binary @./user.
↪json http://localhost:8181/auth/v1/users
PAYLOAD:
{
    "name": "ryan",
    "userid": "ryan@sdn",
    "password": "ryan",
    "domainid": "sdn",
    "description": "Ryan's User Account",
    "email": "ryandgoulding@gmail.com"
}

OUTPUT:
{
    "userid":"ryan@sdn",
    "name":"ryan",
    "description":"Ryan's User Account",
    "enabled":true,
    "email":"ryandgoulding@gmail.com",
    "password":"**********",
    "salt":"**********",
    "domainid":"sdn"
}
```

Create an OAuth2 Token For Admin Scoped to SDN

```
curl -d 'grant_type=password&username=admin&password=a&scope=sdn' http://
↪localhost:8181/oauth2/token

OUTPUT:
{
    "expires_in":3600,
    "token_type":"Bearer",
    "access_token":"5a615fbc-bcad-3759-95f4-ad97e831c730"
}
```

Use an OAuth2 Token

```
curl -H "Authorization: Bearer 5a615fbc-bcad-3759-95f4-ad97e831c730" http://
↪localhost:8181/auth/v1/domains
{
    "domains":
    [
        {
            "domainid":"sdn",
            "name":"sdn",
            "description":"default odl sdn domain",
            "enabled":true
        }
    ]
}
```

Token Store Configuration Parameters

Edit the file "etc/opendaylight/karaf/08-authn-config.xml" and edit the following: .**timeToLive**: Configure the maximum time, in milliseconds, that tokens are to be cached. Default is 360000. Save the file.

ODLJndiLdapRealm

How it works

LDAP integration is provided in order to externalize identity management. This configuration allows federation with an external LDAP server. The user's OpenDaylight role parameters are mapped to corresponding LDAP attributes as specified by the groupRolesMap. Thus, an LDAP operator can provision attributes for LDAP users that support different OpenDaylight role structures.

Configuring ODLJndiLdapRealm

To configure LDAP parameters, modify "etc/shiro.ini" parameters to include the ODLJndiLdapRealm:

```
# OpenDaylight provides a few LDAP implementations, which are disabled out of the box.
# ODLJndiLdapRealm includes authorization functionality based on LDAP elements
# extracted through and LDAP search.  This requires a bit of knowledge about
# how your LDAP system is setup.  An example is provided below:
ldapRealm = org.opendaylight.aaa.shiro.realm.ODLJndiLdapRealm
ldapRealm.userDnTemplate = uid={0},ou=People,dc=DOMAIN,dc=TLD
ldapRealm.contextFactory.url = ldap://<URL>:389
ldapRealm.searchBase = dc=DOMAIN,dc=TLD
ldapRealm.ldapAttributeForComparison = objectClass
ldapRealm.groupRolesMap = "Person":"admin"
# ...
# further down in the file...
# Stacked realm configuration;  realms are round-robbined until authentication␣
↪succeeds or realm sources are exhausted.
securityManager.realms = $tokenAuthRealm, $ldapRealm
```

ODLJndiLdapRealmAuthNOnly

How it works

This is useful for setups where all LDAP users are allowed equal access.

Configuring ODLJndiLdapRealmAuthNOnly

Edit the "etc/shiro.ini" file and modify the following:

```
ldapRealm = org.opendaylight.aaa.shiro.realm.ODLJndiLdapRealm
ldapRealm.userDnTemplate = uid={0},ou=People,dc=DOMAIN,dc=TLD
ldapRealm.contextFactory.url = ldap://<URL>:389
# ...
# further down in the file...
# Stacked realm configuration;  realms are round-robbined until authentication␣
↪succeeds or realm sources are exhausted.
securityManager.realms = $tokenAuthRealm, $ldapRealm
```

KeystoneAuthRealm

How it works

This realm authenticates OpenDaylight users against the OpenStack's Keystone server. This realm uses the Keystone's Identity API v3 or later.

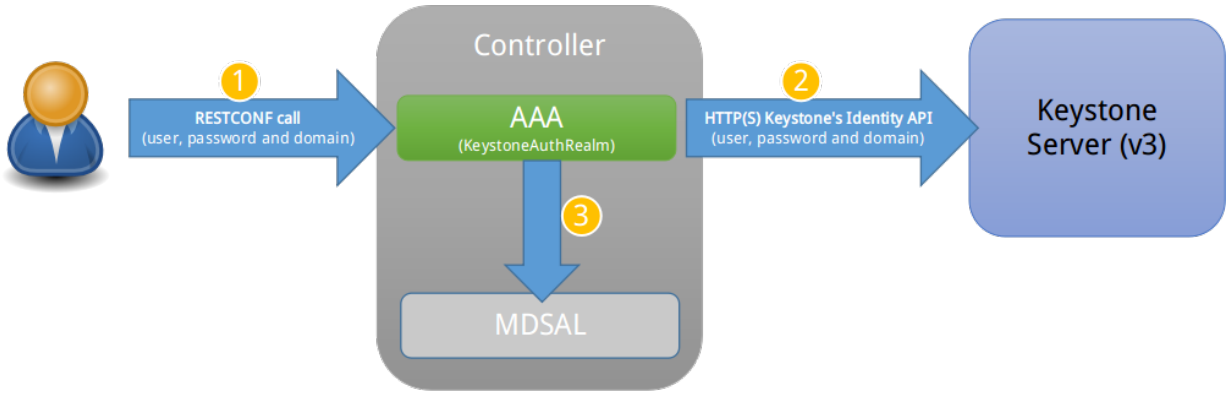


Fig. 1.20: KeystoneAuthRealm authentication/authorization mechanism

As can shown on the above diagram, once configured, all the RESTCONF APIs calls will require sending **user**, **password** and optionally **domain** (1). Those credentials are used to authenticate the call against the Keystone server (2) and, if the authentication succeeds, the call will proceed to the MDSAL (3). The credentials must be provisioned in advance within the Keystone Server. The user and password are mandatory, while the domain is optional, in case it is not provided within the REST call, the realm will default to (**Default**), which is hard-coded. The default domain can be also configured through the *shiro.ini* file (see the *AAA User Guide*).

The protocol between the Controller and the Keystone Server (2) can be either HTTPS or HTTP. In order to use HTTPS the Keystone Server's certificate must be exported and imported on the Controller (see the *Certificate Management* section).

Configuring KeystoneAuthRealm

Edit the "etc/shiro.ini" file and modify the following:

```
# The KeystoneAuthRealm allows for authentication/authorization against an
# OpenStack's Keystone server. It uses the Identity's API v3 or later.
keystoneAuthRealm = org.opendaylight.aaa.shiro.realm.KeystoneAuthRealm
# The URL where the Keystone server exposes the Identity's API v3 the URL
# can be either HTTP or HTTPS and it is mandatory for this realm.
keystoneAuthRealm.url = https://<host>:<port>
# Optional parameter to make the realm verify the certificates in case of HTTPS
#keystoneAuthRealm.sslVerification = true
# Optional parameter to set up a default domain for requests using credentials
# without domain, uncomment in case you want a different value from the hard-coded
# one "Default"
#keystoneAuthRealm.defaultDomain = Default
```

Once configured the realm, the mandatory fields are the fully quallified name of the class implementing the realm *keystoneAuthRealm* and the endpoint where the Keystone Server is listening *keystoneAuthRealm.url*.

The optional parameter *keystoneAuthRealm.sslVerification* specifies whether the realm has to verify the SSL certificate or not. The optional parameter *keystoneAuthRealm.defaultDomain* allows to use a different default domain from the hard-coded one *"Default"*.

Authorization Configuration

OpenDaylight supports two authorization engines at present, both of which are roughly similar in behavior:

- Shiro-Based Authorization
- MDSAL-Based Dynamic Authorization

Note: The preferred mechanism for configuring AAA Authentication is the MDSAL-Based Dynamic Authorization. Read the following section.

Shiro-Based Static Authorization

OpenDaylight AAA has support for Role Based Access Control (RBAC) based on the Apache Shiro permissions system. Configuration of the authorization system is done off-line; authorization currently cannot be configured after the controller is started. The Authorization provided by this mechanism is aimed towards supporting coarse-grained security policies, the MDSAL-Based mechanism allows for a more robust configuration capabilities. Shiro-based Authorization describes how to configure the Authentication feature in detail.

Enable "admin" Role Based Access to the IdMLight RESTful web service

Edit the "etc/shiro.ini" configuration file and add "/auth/v1/**= authcBasic, roles[admin]" above the line "/** = authcBasic" within the "urls" section.

```
/auth/v1/** = authcBasic, roles[admin]
/** = authcBasic
```

This will restrict the idmlight rest endpoints so that a grant for admin role must be present for the requesting user.

Note: The ordering of the authorization rules above is important!

MDSAL-Based Dynamic Authorization

The MDSAL-Based Dynamic authorization uses the MDSALDynamicAuthorizationFilter engine to restrict access to particular URL endpoint patterns. Users may define a list of policies that are insertion-ordered. Order matters for that list of policies, since the first matching policy is applied. This choice was made to emulate behavior of the Shiro-Based Authorization mechanism.

A **policy** is a key/value pair, where the key is a **resource** (i.e., a "URL pattern") and the value is a list of **permissions** for the resource. The following describes the various elements of a policy:

- **Resource**: the resource is a string URL pattern as outlined by Apache Shiro. For more information, see http://shiro.apache.org/web.html.
- **Description**: an optional description of the URL endpoint and why it is being secured.
- **Permissions list**: a list of permissions for a particular policy. If more than one permission exists in the permissions list they are evaluated using logical "OR". A permission describes the prerequisites to perform HTTP operations on a particular endpoint. The following describes the various elements of a permission:
  - **Role**: the role required to access the target URL endpoint.
  - **Actions list**: a leaf-list of HTTP permissions that are allowed for a Subject possessing the required role.

This an example on how to limit access to the modules endpoint:

```
HTTP Operation:
put URL: /restconf/config/aaa:http-authorization/policies

headers: Content-Type: application/json Accept: application/json

body:
  { "aaa:policies":
    { "aaa:policies":
      [ { "aaa:resource": "/restconf/modules/**",
        "aaa:permissions": [ { "aaa:role": "admin",
                               "aaa:actions": [ "get",
                                                "post",
                                                "put",
                                                "patch",
                                                "delete"
                                              ]
                             }
                           ]
        }
      ]
    }
  }
```

The above example locks down access to the modules endpoint (and any URLS available past modules) to the "admin" role. Thus, an attempt from the OOB *admin* user will succeed with 2XX HTTP status code, while an attempt from the OOB *user* user will fail with HTTP status code 401, as the user *user* is not granted the "admin" role.

Accounting Configuration

Accounting is handled through the standard slf4j logging mechanisms used by the rest of OpenDaylight. Thus, one can control logging verbosity through manipulating the log levels for individual packages and classes directly through the Karaf console, JMX, or etc/org.ops4j.pax.logging.cfg. In normal operations, the default levels exposed do not provide much information about AAA services; this is due to the fact that logging can severely degrade performance.

All AAA logging is output to the standard karaf.log file. For debugging purposes (i.e., to enable maximum verbosity), issue the following command:

```
log:set TRACE org.opendaylight.aaa
```

Enable Successful/Unsuccessful Authentication Attempts Logging

By default, successful/unsuccessful authentication attempts are NOT logged. This is due to the fact that logging can severely decrease REST performance. To enable logging of successful/unsuccessful REST attempts, issue the following command in Karaf's console:

```
log:set DEBUG org.opendaylight.aaa.shiro.filters.AuthenticationListener
```

It is possible to add custom AuthenticationListener(s) to the Shiro-based configuration, allowing different ways to listen for successful/unsuccessful authentication attempts. Custom AuthenticationListener(s) must implement the org.apache.shiro.authc.AuthenticationListener interface.

Certificate Management

The **Certificate Management Service** is used to manage the keystores and certificates at the OpenDaylight distribution to easily provides the TLS communication.

The Certificate Management Service managing two keystores:

1. **OpenDaylight Keystore** which holds the OpenDaylight distribution certificate self sign certificate or signed certificate from a root CA based on generated certificate request.
2. **Trust Keystore** which holds all the network nodes certificates that shall to communicate with the OpenDaylight distribution through TLS communication.

The Certificate Management Service stores the keystores (OpenDaylight & Trust) as *.jks* files under configuration/ssl/ directory. Also the keystores could be stored at the MD-SAL datastore in case OpenDaylight distribution running at cluster environment. When the keystores are stored at MD-SAL, the Certificate Management Service rely on the **Encryption-Service** to encrypt the keystore data before storing it to MD-SAL and decrypted at runtime.

How to use the Certificate Management Service to manage the TLS communication

The following are the steps to configure the TLS communication:

1. After starting the distribution, the *odl-aaa-cert* feature has to get installed. Use the following command at Karaf CLI to check.

```
opendaylight-user@root>feature:list -i | grep aaa-cert
odl-aaa-cert | 0.5.0-SNAPSHOT | x | odl-aaa-0.5.0-SNAPSHOT | OpenDaylight :: AAA ::␣
↪aaa certificate Service
```

2. The initial configuration of the Certificate Manager Service exists under the distribution directory etc/opendaylight/datastore/initial/config/aaa-cert-config.xml.

```
<aaa-cert-service-config xmlns="urn:opendaylight:yang:aaa:cert">
  <use-config>false</use-config>
  <use-mdsal>false</use-mdsal>
  <bundle-name>opendaylight</bundle-name>
  <ctlKeystore>
    <name>ctl.jks</name>
    <alias>controller</alias>
    <store-password/>
    <dname>CN=ODL, OU=Dev, O=LinuxFoundation, L=QC Montreal, C=CA</dname>
    <validity>365</validity>
    <key-alg>RSA</key-alg>
    <sign-alg>SHA1WithRSAEncryption</sign-alg>
     <keysize>1024</keysize>
     <cipher-suites>
       <suite-name />
     </cipher-suites>
  </ctlKeystore>
  <trustKeystore>
    <name>truststore.jks</name>
    <store-password/>
  </trustKeystore>
</aaa-cert-service-config>
```

Now as it is explained above, the Certificate Manager Service support two mode of operations; cluster mode and single mode. To use the single mode change the use-config to true and it is recommended as long as there is no need for cluster environment. To use the cluster mode change the use-config and use-mdsal configurations to true and the keystores will be stored and shard across the cluster nodes within the MD-SAL datastore.

The initial password become randomly generated when the *aaa-cert* feature is installed.

The cipher suites can be restricted by changing the **<cipher-suites>** configuration, however, the JDK has to be upgraded by installing the Java Cryptography Extension policy.

```
<cipher-suites>
  <suite-name>TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384</suite-name>
</cipher-suites>
  <cipher-suites>
<suite-name>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384</suite-name>
  </cipher-suites>
<cipher-suites>
  <suite-name>TLS_DHE_RSA_WITH_AES_256_GCM_SHA384</suite-name>
</cipher-suites>
```

3. The new configurations will take affect after restarting the distribution.

4. Now to add or get certificate to the OpenDaylight and Trust keystores, the Certificate Manager Service provides the following RPCs.

```
a) Set the node certificate that will communicate with OpeDaylight through TLS
connection.
POST /operations/aaa-cert-rpc:setNodeCertifcate
{
  "input": {
    "node-cert": "string",
    "node-alias": "string"
  }
```

```
}
```

```
b) Get the node certificate based on node alias.
POST /operations/aaa-cert-rpc:getNodeCertifcate
{
  "input": {
    "node-alias": "string"
  }
}
```

```
c) Get the OpeDaylight keystore certificate.
POST /operations/aaa-cert-rpc:getODLCertificate
{
  output {
    odl-cert "string"
  }
}
```

```
d) Generate a certificate request from the OpeDaylight keystore to be signed
by a CA.
POST /operations/aaa-cert-rpc:getODLCertificateReq
{
  output {
    odl-cert-req "string"
  }
}
```

```
e) Set the OpeDaylight certificate, the certificate should be generated
based on a certificate request generated from the ODL keystore otherwise the
certificated will not be added.
POST /operations/aaa-cert-rpc:setODLCertificate
{
  "input": {
    "odl-cert-alias": "string",
    "odl-cert": "string"
  }
}
```

Note: The Certificate Manager Service RPCs are allowed only to the Role Admin Users and it could be completely disabled through the shiro.ini config file. Check the URL section at the shiro.ini.

Encryption Service

The **AAA Encryption Service** is used to encrypt the OpenDaylight's users' passwords and TLS communication certificates. This section shows how to use the AAA Encryption Service with an OpenDaylight distribution project to encrypt data.

The following are the steps to configure the Encryption Service:

1. After starting the distribution, the *aaa-encryption-service* feature has to get installed. Use the following command at Karaf CLI to check.

```
opendaylight-user@root>feature:list -i | grep aaa-encryption-service
odl-aaa-encryption-service | 0.5.0-SNAPSHOT | x | odl-aaa-0.5.0-SNAPSHOT |␣
↪OpenDaylight :: AAA :: Encryption Service
```

2. The initial configuration of the Encryption Service exists under the distribution directory etc/opendaylight/datastore/initial/config/aaa-encrypt-service-config.xml

```
<aaa-encrypt-service-config xmlns="config:aaa:authn:encrypt:service:config">
  <encrypt-key/>
  <encrypt-salt/>
  <encrypt-method>PBKDF2WithHmacSHA1</encrypt-method>
  <encrypt-type>AES</encrypt-type>
  <encrypt-iteration-count>32768</encrypt-iteration-count>
  <encrypt-key-length>128</encrypt-key-length>
  <cipher-transforms>AES/CBC/PKCS5Padding</cipher-transforms>
</aaa-encrypt-service-config>
```

Note: Both the initial encryption key and encryption salt become randomly generated when the *aaa-encryption-service* feature is installed.

3. Finally the new configurations will take affect after restarting the distribution.

Using the AAA Command Line Interface (CLI)

The AAA offers a CLI through the Karaf's console. This CLI allows the user to configure and use some of the functionalities provided by AAA.

The AAA CLI exists under the **odl-aaa-cli** feature. This feature can be installed by executing the following command.

```
feature:install odl-aaa-cli
```

To check that the installation of the feature succeeded type "aaa" and press *tab* to see the list of available commands under the *aaa* scope.

```
opendaylight-user@root>aaa:
aaa:add-domain           aaa:add-grant            aaa:add-role             aaa:add-
↪user
aaa:change-user-pwd      aaa:export-keystores     aaa:gen-cert-req         aaa:get-
↪cipher-suites
aaa:get-domains          aaa:get-node-cert        aaa:get-odl-cert         aaa:get-
↪roles
aaa:get-tls-protocols    aaa:get-users            aaa:import-keystores     aaa:remove-
↪domain
aaa:remove-grant         aaa:remove-role          aaa:remove-user
```

Add a User

The *add-user* command allows for adding an OpenDaylight user. The following user parameters can be specified.

```
aaa:add-user --name <user name>
             --roleName <role>
             --userDescription <user description>
```

```
            --email <user email>
            --domainName <domain name>
```

List available Users

The *get-users* command list all the available users within the Controller.

```
aaa:get-users

  user
  admin
```

Remove a User

The *remove-user* command allows for removing an OpenDaylight user. The command needs the user name as parameter.

```
aaa:remove-user --name <user name>
```

Change the OpenDaylight user password

The *change-user-pwd* command allows for changing the OpenDaylight user's password. It takes the user name as argument then will ask for the given user current password.

```
aaa:change-user-pwd -user admin
  Enter current password:
  Enter new password:
  admin's password has been changed
```

Add a Role

The *add-role* command allows for adding a role to the Controller.

```
aaa:add-role --name <role name>
             --desc <role description>
             --domainName <domain name>
```

List available Roles

The *get-roles* command list all the available roles within the controller.

```
aaa:get-roles

  user
  admin
```

Remove a Role

The *remove-role* command allows for removing an OpenDaylight role. The command needs the role name as parameter. The role will be removed from those users who have it.

```
aaa:remove-role --name <role name>
```

Add a Domain

The *add-domain* command allows for adding a domain to the Controller.

```
aaa:add-domain --name <domain name>
               --desc <domain description>
```

List available Domains

The *get-domains* command list all the available domains within the controller. The system asks for the administrator credentials to execute this command.

```
aaa:get-domains

  sdn
```

Remove a Domain

The *remove-domain* command allows for removing an OpenDaylight role. The command needs the domain name as parameter.

```
aaa:remove-domain --name <domain name>
```

Add a Grant

The *add-grant* command allows for creating a grant for an existing user. The command returns a grant id for that user.

```
aaa:add-grant --userName <user name>
              --domainName <domain name>
              --roleName <role name>
```

Remove a Grant

The *remove-grant* command allows for removing an OpenDaylight grant. This command needs the user name, domain and and role as parameters.

```
aaa:remove-grant --userName <user name>
                 --domainName <domain name>
                 --roleName <role name>
```

Generate Certificate Request

Generate certificate request command will generate a certificate request based on the generated OpenDaylight keystore and print it on the Karaf CLI. The system asks for the keystore password.

```
aaa:gen-cert-req

-----BEGIN CERTIFICATE REQUEST-----
MIIBlzCCAQACAQAwWTELMAkGA1UEBhMCQ0ExFDASBgNVBAcMC1FDIE1vbnRyZWFsMRgwFgYDVQQKDA
9MaW51eEZvdW5kYXRpb24xDDAKBgNVBAsMA0RldjEMMAoGA1UEAwwDT0RMMIGfMA0GCSqGSIb3DQEB
AQUAA4GNADCBiQKBgQCCmLW6j+JLYJM5yAMwscw/CHqPnp5elPa1YtQsHKEAvp1I+mLVtHKZeXeteA
kyp6ORxw6KQ515fcDyQVrRJiSM15jUd27UaFq5ku0+qJeG+Qh2btx+cvNSE7/+cgUWWosKz4Aff5F5
FqR62jLUTNzqCvoaTbZaOnLYVq+O2dYyZwIDAQABMA0GCSqGSIb3DQEBBQUAA4GBADhDr4Jm7gVm/o
p861/FShyw1ZZscxOEl2TprJZiTO6sn3sLptQZv8v52Z+Jm5dAgr7L46c97Xfa+0j6Y4LXNb0f88lL
RG8PxGbk6Tqbjqc0WS+U1Ibc/rcPK4HEN/bcYCn+Na1gLBaFXUPg08ozG6MwqFNeS5Z0jz1W0D9/oiao
-----END CERTIFICATE REQUEST-----
```

Get OpenDaylight Certificate

The *get-odl-certificate* command will print the OpenDaylight certificate at the Karaf CLI. The system asks for the keystore password.

```
aaa:get-odl-cert -storepass <store_password>

-----BEGIN CERTIFICATE-----
MIICKTCCAZKgAwIBAgIEI75RWDANBgkqhkiG9w0BAQUFADBZMQwwCgYDVQQDDANPREwxDDAKBgNVBA
sMA0RldjEYMBYGA1UECgwPTGludXhGb3VuZGF0aW9uMRQwEgYDVQQHDAtRQyBNb250cmVhbDELMAkG
A1UEBhMCQ0EwHhcNMTYxMTMwMTYyNDE3WhcNMTcxMTMwMTYyNDE3WjBZMQwwCgYDVQQDDANPREwxDD
AKBgNVBAsMA0RldjEYMBYGA1UECgwPTGludXhGb3VuZGF0aW9uMRQwEgYDVQQHDAtRQyBNb250cmVh
bDELMAkGA1UEBhMCQ0EwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAIKYtbqP4ktgkznIAzCxzD
8Ieo+enl6U9rVi1CwcoQC+nUj6YtW0cpl5d614CTKno5HHDopDnXl9wPJBWtEmJIzXmNR3btRoWrmS
7T6ol4b5CHZu3H5y81ITv/5yBRZaiwrPgB9/kXkWpHraMtRM3OoK+hpNtlo6cthWr47Z1jJnAgMBAA
EwDQYJKoZIhvcNAQEFBQADgYEAL9DK/P/yEBre3Mg3bICAUAvSvZic+ydDmigWLsY4J3UzKdV2f1jI
s+rQTEgtlHShBf/ed546D49cp3XEzYrcxgILhGXDziCrUK0K1TiYqPTp6FLijjdydGlPpwuMyyV5Y0
iDiRclWuPz2fHbs8WQOWNs6VQ+WaRExtEsEC4qgSo=
-----END CERTIFICATE-----
```

Get Cipher Suites

The *get-cipher-suites* command shows the cipher suites supported by the JVM used by the OpenDaylight controller in TLS communication. For example, here are the Default Ciphers Suites in JDK 8.

```
aaa:get-cipher-suites

  TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
  TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
  TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
```

Get TLS Protocols

The *get-tls-protocols* command shows the TLS protocols supported by the JVM used by the OpenDaylight controller. For example, the JDK 8 supports the following TLS protocols: TLSv1.2 (default), TLSv1.1, TLSv1 and SSLv3.

```
aaa:get-tls-protocols

  TLS_KRB5_WITH_RC4_128_SHA
  TLS_KRB5_WITH_RC4_128_MD5
  TLS_KRB5_WITH_3DES_EDE_CBC_SHA
  TLS_KRB5_WITH_3DES_EDE_CBC_MD5
  TLS_KRB5_WITH_DES_CBC_SHA
```

Get Node Certificate

The *get-node-cert* command prints a certificate for a given network node alias. This command is useful to check if the network node certificate has been added properly to the truest keystore. It takes the certificate alias as arguments.

```
aaa:get-node-cert -alias ovs1
-----BEGIN CERTIFICATE-----
MIICKTCCAZKgAwIBAgIEI75RWDANBgkqhkiG9w0BAQUFADBZMQwwCgYDVQQDDANPREwxDDAKBgNVBA
sMA0RldjEYMBYGA1UECgwPTGludXhGb3VuZGF0aW9uMRQwEgYDVQQHDAtRQyBNb250cmVhbDELMAkG
A1UEBhMCQ0EwHhcNMTYxMTMwMTYyNDE3WhcNMTcxMTMwMTYyNDE3WjBZMQwwCgYDVQQDDANPREwxDD
AKBgNVBAsMA0RldjEYMBYGA1UECgwPTGludXhGb3VuZGF0aW9uMRQwEgYDVQQHDAtRQyBNb250cmVh
bDELMAkGA1UEBhMCQ0EwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAIKYtbqP4ktgkznIAzCxzD
8Ieo+enl6U9rVi1CwcoQC+nUj6YtW0cpl5d614CTKno5HHDopDnXl9wPJBWtEmJIzXmNR3btRoWrmS
7T6ol4b5CHZu3H5y81ITv/5yBRZaiwrPgB9/kXkWpHraMtRM3OoK+hpNtlo6cthWr47Z1jJnAgMBAA
EwDQYJKoZIhvcNAQEFBQADgYEAL9DK/P/yEBre3Mg3bICAUAvSvZic+ydDmigWLsY4J3UzKdV2f1jI
s+rQTEgtlHShBf/ed546D49cp3XEzYrcxgILhGXDziCrUK0K1TiYqPTp6FLijjdydGlPpwuMyyV5Y0
iDiRclWuPz2fHbs8WQOWNs6VQ+WaREXtEsEC4qgSo=
-----END CERTIFICATE-----
```

Export Keystores

The *export-keystores* command exports the default MD-SAL Keystores to .jks files in the default directory for keystores (configuration/ssl/).

```
aaa:export-keystores

  Default directory for keystores is configuration/ssl/
```

Import Keystores

The *import-keystores* command imports the default MD-SAL Keystores. The keystores (odl and trust) should exist under default SSL directory (configuration/ssl/).

```
aaa:import-keystores --trustKeystoreName <name of the trust keystore>
                     --trustKeystorePwd <password for the trust keystore>
                     --odlKeystoreName <name of the ODL keystore>
                     --odlKeystorePwd <password for the ODL keystore>
                     --odlKeystoreAlias <alias of the ODL keystore>
                     --tlsProtocols <list of TLS protocols separated by ','>
                     --cipherSuites <list of Cipher suites separated by ','>
```

Warning: It is strongly recommended to run the history clear command after you execute all the AAA CLI commands so Karaf logs stay clean from any adversary.

```
history -c
```

BGP User Guide

This guide contains information on how to use OpenDaylight Border Gateway Protocol (BGP) plugin. The user should learn about BGP basic concepts, supported capabilities, configuration and usage.

Contents

Overview

This section provides high-level overview of the Border Gateway Protocol, OpenDaylight implementation and BGP usage in SDN era.

Contents

- *List of supported capabilities*

Border Gateway Protocol

The Border Gateway Protocol (BGP) is an inter-Autonomous System (AS) routing protocol. The primary role of the BGP is an exchange of routes among other BGP systems. The route is an unit of information which pairs destination (IP address prefix) with attributes to the path with the destination. One of the most interesting attributes is a list of ASes that the route traversed - essential when avoiding loop routing. Advertised routes are stored in the Routing Information Bases (RIBs). Routes are later used to forward packets, stored in Routing Table for this purpose. The main advantage of the BGP over other routing protocols is its scalability, thus it has become the standardized Internet routing protocol (Internet is a set of ASes).

BGP in SDN

However BGP evolved long time before SDN was born, it plays a significant role in many SDN use-cases. Also, continuous evolution of the protocol brings extensions that are very well suited for SDN. Nowadays, BGP can carry various types of routing information - L3VPN, L2VPN, IP multicast, linkstate, etc. Here is a brief list of software-based/legacy-network technologies where BGP-based SDN solution get into an action:

- SDN WAN - WAN orchestration and optimization
- SDN router - Turns switch into an Internet router
- Virtual Route Reflector - High-performance server-based BGP Route Reflector
- SDX - A Software Defined Internet Exchange controller
- Large-Scale Data Centers - BGP Data Center Routing, MPLS/SR in DCs, DC interconnection
- DDoS mitigation - Traffic Filtering distribution with BGP

OpenDaylight BGP plugin

The OpenDaylight controller provides an implementation of BGP (RFC 4271) as a south-bound protocol plugin. The implementation renders all basic *BGP speaker capabilities*:

- inter/intra-AS peering
- routes advertising
- routes originating
- routes storage

The plugin's **north-bound API** (`REST/Java`) provides to user:

- fully dynamic runtime standardized BGP configuration
- read-only access to all RIBs
- read-write programmable RIBs
- read-only reachability/linkstate topology view

Note: The BGP plugin is NOT a virtual router - does not construct Routing Tables, nor forward traffic.

List of supported capabilities

In addition to the base protocol implementation, the plugin provides many extensions to BGP, all based on IETF standards.

- RFC4271 - A Border Gateway Protocol 4 (BGP-4)
- RFC4456 - BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP)
- RFC1997 - BGP Communities Attribute
- RFC4360 - BGP Extended Communities Attribute
- RFC4486 - Subcodes for BGP Cease Notification Message
- RFC5492 - Capabilities Advertisement with BGP-4
- RFC5004 - Avoid BGP Best Path Transitions from One External to Another
- RFC6286 - Autonomous-System-Wide Unique BGP Identifier for BGP-4
- RFC6793 - BGP Support for Four-Octet Autonomous System (AS) Number Space
- RFC7311 - The Accumulated IGP Metric Attribute for BGP
- RFC5668 - 4-Octet AS Specific BGP Extended Community
- draft-ietf-idr-link-bandwidth - BGP Link Bandwidth Extended Community
- draft-ietf-idr-bgp-extended-messages - Extended Message support for BGP
- **RFC4760 - Multiprotocol Extensions for BGP-4**
  - **RFC7752 - North-Bound Distribution of Link-State and TE Information using BGP**
    - draft-gredler-idr-bgp-ls-segment-routing-ext - BGP Link-State extensions for Segment Routing
    - draft-ietf-idr-bgpls-segment-routing-epe - Segment Routing Egress Peer Engineering BGP-LS Extensions
  - **RFC5575 - Dissemination of Flow Specification Rules**
    - RFC7674 - Clarification of the Flowspec Redirect Extended Community
    - draft-ietf-idr-flow-spec-v6 - Dissemination of Flow Specification Rules for IPv6
    - draft-ietf-idr-flowspec-redirect-ip - BGP Flow-Spec Redirect to IP Action
  - **RFC3107 - Carrying Label Information in BGP-4**
    - draft-ietf-idr-bgp-prefix-sid - Segment Routing Prefix SID extensions for BGP
  - **RFC4364 - BGP/MPLS IP Virtual Private Networks (VPNs)**
    - RFC4659 - BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN
  - **RFC7432 - BGP MPLS-Based Ethernet VPN**
    - draft-ietf-bess-evpn-overlay - A Network Virtualization Overlay Solution using EVPN
    - draft-ietf-bess-evpn-vpws - VPWS support in EVPN
- RFC7911 - Advertisement of Multiple Paths in BGP
- RFC2918 - Route Refresh Capability for BGP-4

Running BGP

This section explains how to install BGP plugin.

1. Install BGP feature - `odl-bgpcep-bgp`. Also, for sake of this sample, it is required to install RESTCONF. In the Karaf console, type command:

```
feature:install odl-restconf odl-bgpcep-bgp
```

2. The BGP plugin contains a default configuration, which is applied after the feature starts up. One instance of BGP plugin is created (named *example-bgp-rib*), and its presence can be verified via REST:

   URL: `/restconf/operational/bgp-rib:bgp-rib`

   Method: `GET`

   Response Body:

```
<bgp-rib xmlns="urn:opendaylight:params:xml:ns:yang:bgp-rib">
   <rib>
       <id>example-bgp-rib</id>
       <loc-rib>
       ....
       </loc-rib>
   </rib>
</bgp-rib>
```

Basic Configuration & Concepts

The following section shows how to configure BGP basics, how to verify functionality and presents essential components of the plugin. Next samples demonstrate the plugin's runtime configuration capability. It shows the way to configure the plugin via REST, using standardized OpenConfig BGP APIs.

Contents

- *References*

BGP RIB API

This tree illustrates the BGP RIBs organization in datastore.

```
bgp-rib
  +--ro rib* [id]
     +--ro id          rib-id
     +--ro peer* [peer-id]
     |  +--ro peer-id                  peer-id
     |  +--ro peer-role                peer-role
     |  +--ro simple-routing-policy?   simple-routing-policy
     |  +--ro supported-tables* [afi safi]
     |  |  +--ro afi             identityref
     |  |  +--ro safi            identityref
     |  |  +--ro send-receive?   send-receive
     |  +--ro adj-rib-in
     |  |  +--ro tables* [afi safi]
     |  |     +--ro afi           identityref
     |  |     +--ro safi          identityref
     |  |     +--ro attributes
     |  |     |  +--ro uptodate?   boolean
     |  |     +--ro (routes)?
     |  +--ro effective-rib-in
     |  |  +--ro tables* [afi safi]
     |  |     +--ro afi           identityref
     |  |     +--ro safi          identityref
     |  |     +--ro attributes
     |  |     |  +--ro uptodate?   boolean
     |  |     +--ro (routes)?
     |  +--ro adj-rib-out
     |     +--ro tables* [afi safi]
     |        +--ro afi           identityref
     |        +--ro safi          identityref
     |        +--ro attributes
     |        |  +--ro uptodate?   boolean
     |        +--ro (routes)?
     +--ro loc-rib
        +--ro tables* [afi safi]
           +--ro afi           identityref
           +--ro safi          identityref
           +--ro attributes
           |  +--ro uptodate?   boolean
           +--ro (routes)?
```

Protocol Configuration

As a first step, a new protocol instance needs to be configured. It is a very basic configuration conforming with RFC4271.

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
1  <protocol xmlns="http://openconfig.net/yang/network-instance">
2      <name>bgp-example</name>
3      <identifier xmlns:x="http://openconfig.net/yang/policy-types">x:BGP</identifier>
4      <bgp xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
5          <global>
6              <config>
7                  <router-id>192.0.2.2</router-id>
8                  <as>65000</as>
9              </config>
10         </global>
11     </bgp>
12 </protocol>
```

@line 2: The unique protocol instance identifier.

@line 7: BGP Identifier of the speaker.

@line 8: Local autonomous system number of the speaker. Note that, OpenDaylight BGP implementation supports four-octet AS numbers only.

The new instance presence can be verified via REST:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example`

Method: `GET`

Response Body:

```
1  <rib xmlns="urn:opendaylight:params:xml:ns:yang:bgp-rib">
2      <id>bgp-example</id>
3      <loc-rib>
4          <tables>
5              <afi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:ipv4-
   ↪address-family</afi>
6              <safi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:unicast-
   ↪subsequent-address-family</safi>
7              <ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet"></ipv4-
   ↪routes>
8              <attributes>
9                  <uptodate>true</uptodate>
10             </attributes>
11         </tables>
12     </loc-rib>
13 </rib>
```

@line 3: Loc-RIB - Per-protocol instance RIB, which contains the routes that have been selected by local BGP speaker's decision process.

@line 4: The BGP-4 supports carrying IPv4 prefixes, such routes are stored in *ipv4-address-family/unicast-subsequent-address-family* table.

BGP Server

BGP uses TCP as its transport protocol, by default listens on port 179. OpenDaylight BGP plugin is configured to listen on port *1790*, due to privileged ports restriction for non-root users. One of the workarounds is to use port

redirection. In case other port is desired to be used instead, we can reconfigure it.

Here is a sample of bgp port listening re-configuration:

URL: `/restconf/config/odl-bgp-peer-acceptor-config:bgp-peer-acceptor-config/default`

Method: `PUT`

Content-Type: `application/xml`

Request Body:

```
<bgp-peer-acceptor-config xmlns="urn:opendaylight:params:xml:ns:yang:odl-bgp-peer-acceptor-config">
    <config-name>default</config-name>
    <binding-address>0.0.0.0</binding-address>
    <binding-port>1791</binding-port>
</bgp-peer-acceptor-config>
```

@line 3: Binding address: By default is 0.0.0.0, so it is not a mandatory field.

@line 4: Binding Port: Port were BGP Server will listen.

BGP Peering

To exchange routing information between two BGP systems (peers), it is required to configure a peering on both BGP speakers first. This mean that each BGP speaker has a white list of neighbors, representing remote peers, with which the peering is allowed. The TCP connection is established between two peers and they exchange messages to open and confirm the connection parameters followed by routes exchange.

Here is a sample basic neighbor configuration:

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<neighbor xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <neighbor-address>192.0.2.1</neighbor-address>
    <timers>
        <config>
            <hold-time>90</hold-time>
            <connect-retry>10</connect-retry>
        </config>
    </timers>
    <transport>
        <config>
            <remote-port>179</remote-port>
            <passive-mode>false</passive-mode>
        </config>
    </transport>
    <config>
        <peer-type>INTERNAL</peer-type>
    </config>
</neighbor>
```

@line 2: IP address of the remote BGP peer. Also serves as an unique identifier of a neighbor in a list of neighbors.

@line 5: Proposed number of seconds for value of the Hold Timer. Default value is **90**.

@line 6: Time interval in seconds between attempts to establish session with the peer. Effective in active mode only. Default value is **30**.

@line 11: Remote port number to which the local BGP is connecting. Effective in active mode only. Default value **179**.

@line 12: Wait for peers to issue requests to open a BGP session, rather than initiating sessions from the local router. Default value is **false**.

@line 16: Explicitly designate the peer as internal or external. Default value is **INTERNAL**.

Once the remote peer is connected and it advertised routes to local BGP system, routes are stored in peer's RIBs. The RIBs can be checked via REST:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/peer/bgp:%2F%2F192.0.2.1`

Method: `GET`

Response Body:

```
<peer xmlns="urn:opendaylight:params:xml:ns:yang:bgp-rib">
    <peer-id>bgp://192.0.2.1</peer-id>
    <supported-tables>
        <afi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:ipv4-address-family</afi>
        <safi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:unicast-subsequent-address-family</safi>
    </supported-tables>
    <peer-role>ibgp</peer-role>
    <adj-rib-in>
        <tables>
            <afi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:ipv4-address-family</afi>
            <safi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:unicast-subsequent-address-family</safi>
            <ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet">
                <ipv4-route>
                    <path-id>0</path-id>
                    <prefix>10.0.0.10/32</prefix>
                    <attributes>
                        <as-path></as-path>
                        <origin>
                            <value>igp</value>
                        </origin>
                        <local-pref>
                            <pref>100</pref>
                        </local-pref>
                        <ipv4-next-hop>
                            <global>10.10.1.1</global>
                        </ipv4-next-hop>
                    </attributes>
                </ipv4-route>
            </ipv4-routes>
```

```
30              <attributes>
31                  <uptodate>true</uptodate>
32              </attributes>
33          </tables>
34      </adj-rib-in>
35      <effective-rib-in>
36          <tables>
37              <afi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:ipv4-
    ↪address-family</afi>
38              <safi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:unicast-
    ↪subsequent-address-family</safi>
39              <ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet">
40                  <ipv4-route>
41                      <path-id>0</path-id>
42                      <prefix>10.0.0.10/32</prefix>
43                      <attributes>
44                          <as-path></as-path>
45                          <origin>
46                              <value>igp</value>
47                          </origin>
48                          <local-pref>
49                              <pref>100</pref>
50                          </local-pref>
51                          <ipv4-next-hop>
52                              <global>10.10.1.1</global>
53                          </ipv4-next-hop>
54                      </attributes>
55                  </ipv4-route>
56              </ipv4-routes>
57              <attributes>
58                  <uptodate>true</uptodate>
59              </attributes>
60          </tables>
61      </effective-rib-in>
62      <adj-rib-out>
63          <tables>
64              <afi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:ipv4-
    ↪address-family</afi>
65              <safi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:unicast-
    ↪subsequent-address-family</safi>
66              <ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet"></ipv4-
    ↪routes>
67              <attributes></attributes>
68          </tables>
69      </adj-rib-out>
70  </peer>
```

@line 8: **Adj-RIB-In** - Per-peer RIB, which contains unprocessed routes that has been advertised to local BGP speaker by the remote peer.

@line 13: Here is the reported route with destination *10.0.0.10/32* in Adj-RIB-In.

@line 35: **Effective-RIB-In** - Per-peer RIB, which contains processed routes as a result of applying inbound policy to Adj-RIB-In routes.

@line 40: Here is the reported route with destination *10.0.0.10/32*, same as in Adj-RIB-In, as it was not touched by import policy.

@line 62: **Adj-RIB-Out** - Per-peer RIB, which contains routes for advertisement to the peer by means of the local

speaker's UPDATE message.

@line 66: The peer's Adj-RIB-Out is empty as there are no routes to be advertise from local BGP speaker.

Also the same route should appeared in Loc-RIB now:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/loc-rib/tables/bgp-types:ipv4-address-family/bgp-types:unicast-subsequent-address-family/ipv4-routes`

Method: `GET`

Response Body:

```
<ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet">
    <ipv4-route>
        <path-id>0</path-id>
        <prefix>10.0.0.10/32</prefix>
        <attributes>
            <as-path></as-path>
            <origin>
                <value>igp</value>
            </origin>
            <local-pref>
                <pref>100</pref>
            </local-pref>
            <ipv4-next-hop>
                <global>10.10.1.1</global>
            </ipv4-next-hop>
        </attributes>
    </ipv4-route>
</ipv4-routes>
```

@line 4: **Destination** - IPv4 Prefix Address.

@line 6: **AS_PATH** - mandatory attribute, contains a list of the autonomous system numbers through that routing information has traversed.

@line 8: **ORIGIN** - mandatory attribute, indicates an origin of the route - **ibgp**, **egp**, **incomplete**.

@line 11: **LOCAL_PREF** - indicates a degree of preference for external routes, higher value is preferred.

@line 14: **NEXT_HOP** - mandatory attribute, defines IP address of the router that should be used as the next hop to the destination.

There are much more attributes that may be carried along with the destination:

BGP-4 Path Attributes

- **MULTI_EXIT_DISC (MED)** Optional attribute, to be used to discriminate among multiple exit/entry points on external links, lower number is preferred.

```
<multi-exit-disc>
 <med>0</med>
</multi-exit-disc>
```

- **ATOMIC_AGGREGATE** Indicates whether AS_SET was excluded from AS_PATH due to routes aggregation.

```
<atomic-aggregate/>
```

- **AGGREGATOR** Optional attribute, contains AS number and IP address of a BGP speaker which performed routes aggregation.

```
<aggregator>
    <as-number>65000</as-number>
    <network-address>192.0.2.2</network-address>
</aggregator>
```

- **Unrecognised** Optional attribute, used to store optional attributes, unrecognized by a local BGP speaker.

```
<unrecognized-attributes>
    <partial>true</partial>
    <transitive>true</transitive>
    <type>101</type>
    <value>0101010101010101</value>
</unrecognized-attributes>
```

Route Reflector Attributes

- **ORIGINATOR_ID** Optional attribute, carries BGP Identifier of the originator of the route.

```
<originator-id>
    <originator>41.41.41.41</originator>
</originator-id>
```

- **CLUSTER_LIST** Optional attribute, contains a list of CLUSTER_ID values representing the path that the route has traversed.

```
<cluster-id>
    <cluster>40.40.40.40</cluster>
</cluster-id>
```

- **Communities** Optional attribute, may be used for policy routing.

```
<communities>
    <as-number>65000</as-number>
    <semantics>30740</semantics>
</communities>
```

Extended Communities

- **Route Target** Identifies one or more routers that may receive a route.

```
<extended-communities>
    <transitive>true</transitive>
    <route-target-ipv4>
        <global-administrator>192.0.2.2</global-administrator>
        <local-administrator>123</local-administrator>
    </route-target-ipv4>
</extended-communities>
<extended-communities>
    <transitive>true</transitive>
    <as-4-route-target-extended-community>
            <as-4-specific-common>
            <as-number>65000</as-number>
            <local-administrator>123</local-administrator>
```

```
        </as-4-specific-common>
    </as-4-route-target-extended-community>
</extended-communities>
```

- **Route Origin** Identifies one or more routers that injected a route.

```
<extended-communities>
    <transitive>true</transitive>
    <route-origin-ipv4>
        <global-administrator>192.0.2.2</global-administrator>
        <local-administrator>123</local-administrator>
    </route-origin-ipv4>
</extended-communities>
<extended-communities>
    <transitive>true</transitive>
    <as-4-route-origin-extended-community>
        <as-4-specific-common>
            <as-number>65000</as-number>
            <local-administrator>123</local-administrator>
        </as-4-origin-common>
    </as-4-route-target-extended-community>
</extended-communities>
```

- **Link Bandwidth** Carries the cost to reach external neighbor.

```
<extended-communities>
    <transitive>true</transitive>
    <link-bandwidth-extended-community>
        <bandwidth>BH9CQAA=</bandwidth>
    </link-bandwidth-extended-community>
</extended-communities>
```

- **AIGP** Optional attribute, carries accumulated IGP metric.

```
<aigp>
    <aigp-tlv>
        <metric>120</metric>
    </aigp-tlv>
</aigp>
```

Note: When the remote peer disconnects, it disappear from operational state of local speaker instance and advertised routes are removed too.

External peering configuration

An example above provided configuration for internal peering only. Following configuration sample is intended for external peering:

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<neighbor xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <neighbor-address>192.0.2.3</neighbor-address>
    <config>
        <peer-type>EXTERNAL</peer-type>
        <peer-as>64999</peer-as>
    </config>
</neighbor>
```

@line 5: AS number of the remote peer.

Route reflector configuration

The local BGP speaker can be configured with a specific *cluster ID*. Following example adds the cluster ID to the existing speaker instance:

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/global/config`

Method: `PUT`

Content-Type: `application/xml`

Request Body:

```
<config>
    <router-id>192.0.2.2</router-id>
    <as>65000</as>
    <route-reflector-cluster-id>192.0.2.1</route-reflector-cluster-id>
</config>
```

@line 4: Route-reflector cluster id to use when local router is configured as a route reflector. The *router-id* is used as a default value.

Following configuration sample is intended for route reflector client peering:

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<neighbor xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <neighbor-address>192.0.2.4</neighbor-address>
    <config>
        <peer-type>INTERNAL</peer-type>
    </config>
    <route-reflector>
        <config>
            <route-reflector-client>true</route-reflector-client>
        </config>
    </route-reflector>
</neighbor>
```

@line 8: Configure the neighbor as a route reflector client. Default value is *false*.

MD5 authentication configuration

The OpenDaylight BGP implementation is supporting TCP MD5 for authentication. Sample configuration below shows how to set authentication password for a peer:

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<neighbor xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <neighbor-address>192.0.2.5</neighbor-address>
    <config>
        <auth-password>topsecret</auth-password>
    </config>
</neighbor>
```

@line 4: Configures an MD5 authentication password for use with neighboring devices.

Simple Routing Policy configuration

The OpenDaylight BGP implementation is supporting *Simple Routing Policy*. Sample configuration below shows how to set *Simple Routing Policy* for a peer:

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<neighbor xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <neighbor-address>192.0.2.7</neighbor-address>
    <config>
        <simple-routing-policy>learn-none</simple-routing-policy>
    </config>
</neighbor>
```

@line 4: *Simple Routing Policy*:

- `learn-none` - routes advertised by the peer are not propagated to Effective-RIB-In and Loc-RIB
- `announce-none` - routes from local Loc-RIB are not advertised to the peer

---

Note: Existing neighbor configuration can be reconfigured (change configuration parameters) anytime. As a result, established connection is dropped, peer instance is recreated with a new configuration settings and connection re-established.

Note: The BGP configuration is persisted on OpendDaylight shutdown and restored after the re-start.

BGP Application Peer and programmable RIB

The OpenDaylight BGP implementation also supports routes injection via *Application Peer*. Such peer has its own programmable RIB, which can be modified by user. This concept allows user to originate new routes and advertise them to all connected peers.

Application Peer configuration

Following configuration sample show a way to configure the *Application Peer*:

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
1 <neighbor xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
2     <neighbor-address>10.25.1.9</neighbor-address>
3     <config>
4         <peer-group>application-peers</peer-group>
5     </config>
6 </neighbor>
```

@line 2: IP address is uniquely identifying *Application Peer* and its programmable RIB. Address is also used in local BGP speaker decision process.

@line 4: Indicates that peer is associated with *application-peers* group. It serves to distinguish *Application Peer's* from regular neighbors.

The *Application Peer* presence can be verified via REST:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/peer/bgp:%2F%2F10.25.1.9`

Method: `GET`

Response Body:

```
1 <peer xmlns="urn:opendaylight:params:xml:ns:yang:bgp-rib">
2     <peer-id>bgp://10.25.1.9</peer-id>
3     <peer-role>internal</peer-role>
4     <adj-rib-in>
5         <tables>
6             <afi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:ipv4-
  ↪address-family</afi>
7             <safi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:unicast-
  ↪subsequent-address-family</safi>
8             <ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet"></ipv4-
  ↪routes>
```

```
            <attributes>
                <uptodate>false</uptodate>
            </attributes>
        </tables>
    </adj-rib-in>
    <effective-rib-in>
        <tables>
            <afi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:ipv4-address-family</afi>
            <safi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:unicast-subsequent-address-family</safi>
            <ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet"></ipv4-routes>
            <attributes></attributes>
        </tables>
    </effective-rib-in>
</peer>
```

@line 3: Peer role for *Application Peer* is *internal*.

@line 8: Adj-RIB-In is empty, as no routes were originated yet.

Note: There is no Adj-RIB-Out for *Application Peer*.

Programmable RIB

Next example shows how to inject a route into the programmable RIB.

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/bgp-types:unicast-subsequent-address-family/bgp-inet:ipv4-routes`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<ipv4-route xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet">
    <path-id>0</path-id>
    <prefix>10.0.0.11/32</prefix>
    <attributes>
        <as-path></as-path>
        <origin>
            <value>igp</value>
        </origin>
        <local-pref>
            <pref>100</pref>
        </local-pref>
        <ipv4-next-hop>
            <global>10.11.1.1</global>
        </ipv4-next-hop>
    </attributes>
</ipv4-route>
```

Now the injected route appears in *Application Peer's* RIBs and in local speaker's Loc-RIB:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/peer/bgp:%2F%2F10.25.1.9`

Method: `GET`

Response Body:

```
<peer xmlns="urn:opendaylight:params:xml:ns:yang:bgp-rib">
    <peer-id>bgp://10.25.1.9</peer-id>
    <peer-role>internal</peer-role>
    <adj-rib-in>
        <tables>
            <afi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:ipv4-address-family</afi>
            <safi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:unicast-subsequent-address-family</safi>
            <ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet">
                <ipv4-route>
                    <path-id>0</path-id>
                    <prefix>10.0.0.11/32</prefix>
                    <attributes>
                        <as-path></as-path>
                        <origin>
                            <value>igp</value>
                        </origin>
                        <local-pref>
                            <pref>100</pref>
                        </local-pref>
                        <ipv4-next-hop>
                            <global>10.11.1.1</global>
                        </ipv4-next-hop>
                    </attributes>
                </ipv4-route>
            </ipv4-routes>
            <attributes>
                <uptodate>false</uptodate>
            </attributes>
        </tables>
    </adj-rib-in>
    <effective-rib-in>
        <tables>
            <afi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:ipv4-address-family</afi>
            <safi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:unicast-subsequent-address-family</safi>
            <ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet">
                <ipv4-route>
                    <path-id>0</path-id>
                    <prefix>10.0.0.11/32</prefix>
                    <attributes>
                        <as-path></as-path>
                        <origin>
                            <value>igp</value>
                        </origin>
                        <local-pref>
                            <pref>100</pref>
                        </local-pref>
                        <ipv4-next-hop>
```

```
                            <global>10.11.1.1</global>
                        </ipv4-next-hop>
                    </attributes>
                </ipv4-route>
            </ipv4-routes>
            <attributes></attributes>
        </tables>
    </effective-rib-in>
</peer>
```

@line 9: Injected route is present in *Application Peer's* Adj-RIB-In and Effective-RIB-In.

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/loc-rib/tables/bgp-types:ipv4-address-family/bgp-types:unicast-subsequent-address-family/ipv4-routes`

Method: `GET`

Response Body:

```
<ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet">
    <ipv4-route>
        <path-id>0</path-id>
        <prefix>10.0.0.10/32</prefix>
        <attributes>
            <as-path></as-path>
            <origin>
                <value>igp</value>
            </origin>
            <local-pref>
                <pref>100</pref>
            </local-pref>
            <ipv4-next-hop>
                <global>10.11.1.1</global>
            </ipv4-next-hop>
        </attributes>
    </ipv4-route>
    <ipv4-route>
        <path-id>0</path-id>
        <prefix>10.0.0.10/32</prefix>
        <attributes>
            <as-path></as-path>
            <origin>
                <value>igp</value>
            </origin>
            <local-pref>
                <pref>100</pref>
            </local-pref>
            <ipv4-next-hop>
                <global>10.10.1.1</global>
            </ipv4-next-hop>
        </attributes>
    </ipv4-route>
</ipv4-routes>
```

@line 2: The injected route is now present in Loc-RIB along with a route (destination *10.0.0.10/32*) advertised by remote peer.

This route is also advertised to the remote peer (*192.0.2.1*), hence route appears in its Adj-RIB-Out:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/peer/bgp:%2F%2F192.0.2.1/adj-rib-out/tables/bgp-types:ipv4-address-family/bgp-types:unicast-subsequent-address-family/bgp-inet:ipv4-routes`

Method: `GET`

Response Body:

```
<ipv4-route xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet">
    <path-id>0</path-id>
    <prefix>10.0.0.11/32</prefix>
    <attributes>
        <as-path></as-path>
        <origin>
            <value>igp</value>
        </origin>
        <local-pref>
            <pref>100</pref>
        </local-pref>
        <ipv4-next-hop>
            <global>10.11.1.1</global>
        </ipv4-next-hop>
    </attributes>
</ipv4-route>
```

The injected route can be modified (i.e. different path attribute):

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/bgp-types:unicast-subsequent-address-family/bgp-inet:ipv4-routes/ipv4-route/10.0.0.11%2F32/0`

Method: `PUT`

Content-Type: `application/xml`

Request Body:

```
<ipv4-route xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet">
    <path-id>0</path-id>
    <prefix>10.0.0.11/32</prefix>
    <attributes>
        <as-path></as-path>
        <origin>
            <value>igp</value>
        </origin>
        <local-pref>
            <pref>50</pref>
        </local-pref>
        <ipv4-next-hop>
            <global>10.11.1.2</global>
        </ipv4-next-hop>
    </attributes>
</ipv4-route>
```

The route can be removed from programmable RIB in a following way:

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/bgp-types:unicast-subsequent-address-family/bgp-inet:ipv4-routes/ipv4-route/10.0.0.11%2F32/0`

Method: `DELETE`

Also it is possible to remove all routes from a particular table at once:

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/bgp-types:unicast-subsequent-address-family/bgp-inet:ipv4-routes/`

Method: `DELETE`

Consequently, route disappears from programmable RIB, *Application Peer's* RIBs, Loc-RIB and peer's Adj-RIB-Out (UPDATE message with prefix withdrawal is send).

Note: Routes stored in programmable RIB are persisted on OpendDaylight shutdown and restored after the re-start.

BGP Protocol Configuration Loader

BGP Protocol Configuration Loader allows user to define static initial configuration for a BGP protocol instance. This service will detect the creation of new configuration files following the pattern "protocols-*.xml" under the path "etc/opendaylight/bgp". Once the file is processed, the defined configuration will be available from the configuration Data Store.

Note: If the BGP instance is already present, no update or configuration will be applied.

When installing BGP an example will be provided and a default configuration loaded.

PATH: `etc/opendaylight/bgp/protocols-config.xml`

```
<protocols xmlns="http://openconfig.net/yang/network-instance">
    <protocol>
        <name>example-bgp-rib</name>
        <identifier xmlns:x="http://openconfig.net/yang/policy-types">x:BGP</
↪identifier>
        <bgp xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
            <global>
                <config>
                    <router-id>192.0.2.2</router-id>
                    <as>64496</as>
                    <!-- if cluster-id is not present, it's value is the same as bgp-
↪id -->
                    <!-- <route-reflector-cluster-id>192.0.2.3</route-reflector-
↪cluster-id> -->
                    <!-- <read-only-limit>120</read-only-limit>-->
                </config>
```

```
                <afi-safis>
                    <afi-safi>
                        <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">
↪x:IPV4-UNICAST</afi-safi-name>
                        <!--Advertise N Paths
                        <receive>true</receive>
                        <send-max>2</send-max>-->
                    </afi-safi>
                    <afi-safi>
                        <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">
↪x:IPV6-UNICAST</afi-safi-name>
                    </afi-safi>
                    <afi-safi>
                        <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">
↪x:IPV4-LABELLED-UNICAST</afi-safi-name>
                    </afi-safi>
                    <afi-safi>
                        <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">
↪x:IPV6-LABELLED-UNICAST</afi-safi-name>
                    </afi-safi>
                    <afi-safi>
                        <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">
↪x:L3VPN-IPV4-UNICAST</afi-safi-name>
                    </afi-safi>
                    <afi-safi>
                        <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">
↪x:L3VPN-IPV6-UNICAST</afi-safi-name>
                    </afi-safi>
                    <afi-safi>
                        <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">
↪x:L2VPN-EVPN</afi-safi-name>
                    </afi-safi>
                    <afi-safi>
                        <afi-safi-name>LINKSTATE</afi-safi-name>
                    </afi-safi>
                    <afi-safi>
                        <afi-safi-name>IPV4-FLOW</afi-safi-name>
                    </afi-safi>
                    <afi-safi>
                        <afi-safi-name>IPV6-FLOW</afi-safi-name>
                    </afi-safi>
                    <afi-safi>
                        <afi-safi-name>IPV4-L3VPN-FLOW</afi-safi-name>
                    </afi-safi>
                    <afi-safi>
                        <afi-safi-name>IPV6-L3VPN-FLOW</afi-safi-name>
                    </afi-safi>
                </afi-safis>
            </global>
            <neighbors xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-
↪extensions">
                <neighbor xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-
↪extensions">
                    <neighbor-address>192.0.2.1</neighbor-address>
                    <config>
                        <peer-type>INTERNAL</peer-type>
                        <peer-as>64496</peer-as>
                    </config>
```

```
                    <transport>
                        <config>
                            <remote-port>179</remote-port>
                            <passive-mode>true</passive-mode>
                        </config>
                    </transport>
                    <timers>
                        <config>
                            <hold-time>180</hold-time>
                            <connect-retry>10</connect-retry>
                        </config>
                    </timers>
                    <route-reflector>
                        <config>
                            <route-reflector-client>false</route-reflector-client>
                        </config>
                    </route-reflector>
                    <afi-safis>
                        <afi-safi>
                            <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-
↪types">x:IPV4-UNICAST</afi-safi-name>
                            <!--Advertise N Paths
                            <receive>true</receive>
                            <send-max>0</send-max>-->
                        </afi-safi>
                        <afi-safi>
                            <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-
↪types">x:IPV6-UNICAST</afi-safi-name>
                        </afi-safi>
                        <afi-safi>
                            <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-
↪types">x:IPV4-LABELLED-UNICAST</afi-safi-name>
                        </afi-safi>
                        <afi-safi>
                            <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-
↪types">x:IPV6-LABELLED-UNICAST</afi-safi-name>
                        </afi-safi>
                        <afi-safi>
                            <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-
↪types">x:L3VPN-IPV4-UNICAST</afi-safi-name>
                        </afi-safi>
                        <afi-safi>
                            <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-
↪types">x:L3VPN-IPV6-UNICAST</afi-safi-name>
                        </afi-safi>
                        <afi-safi>
                            <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-
↪types">x:L2VPN-EVPN</afi-safi-name>
                        </afi-safi>
                        <afi-safi>
                            <afi-safi-name>LINKSTATE</afi-safi-name>
                        </afi-safi>
                        <afi-safi>
                            <afi-safi-name>IPV4-FLOW</afi-safi-name>
                        </afi-safi>
                        <afi-safi>
                            <afi-safi-name>IPV6-FLOW</afi-safi-name>
                        </afi-safi>
```

```
                        <afi-safi>
                            <afi-safi-name>IPV4-L3VPN-FLOW</afi-safi-name>
                        </afi-safi>
                        <afi-safi>
                            <afi-safi-name>IPV6-L3VPN-FLOW</afi-safi-name>
                        </afi-safi>
                    </afi-safis>
                </neighbor>
                <neighbor xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-
↪extensions">
                    <neighbor-address>192.0.2.6</neighbor-address>
                    <config>
                        <peer-group>application-peers</peer-group>
                    </config>
                </neighbor>
            </neighbors>
        </bgp>
    </protocol>
</protocols>
```

BGP pipeline

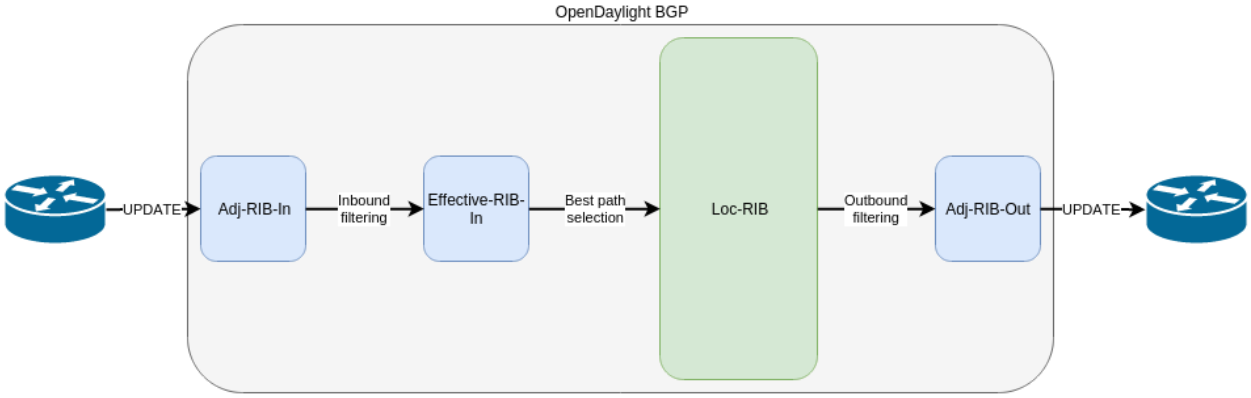


Fig. 1.21: BGP pipeline - routes re-advertisement.

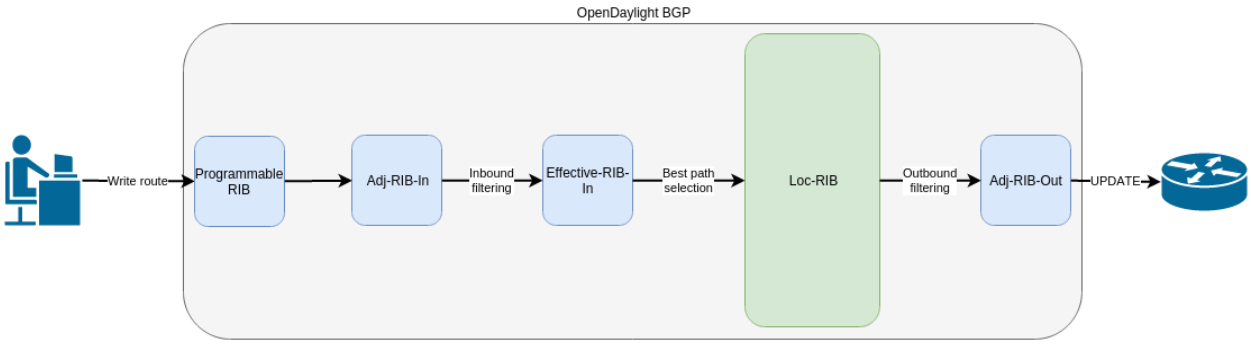


Fig. 1.22: BGP applcaition peer pipeline - routes injection.

References

- A Border Gateway Protocol 4 (BGP-4)
- BGP Route Reflection
- BGP Communities Attribute
- BGP Support for Four-Octet Autonomous System (AS) Number Space
- The Accumulated IGP Metric Attribute for BGP
- 4-Octet AS Specific BGP Extended Community
- BGP Link Bandwidth Extended Community
- Use of BGP for Routing in Large-Scale Data Centers

IP Unicast Family

The BGP-4 allows to carry IPv4 specific information only. The basic BGP Multiprotocol extension brings *Unicast* Subsequent Address Family (SAFI) - intended to be used for IP unicast forwarding. The combination of IPv4 and IPv6 Address Family (AF) and Unicast SAFI is essential for Internet routing. The IPv4 Unicast routes are interchangeable with BGP-4 routes, as they can carry the same type of routing information.

Contents

Configuration

This section shows a way to enable IPv4 and IPv6 Unicast family in BGP speaker and peer configuration.

BGP Speaker

To enable IPv4 and IPv6 Unicast support in BGP plugin, first configure BGP speaker instance:

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<protocol xmlns="http://openconfig.net/yang/network-instance">
    <name>bgp-example</name>
    <identifier xmlns:x="http://openconfig.net/yang/policy-types">x:BGP</identifier>
    <bgp xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
        <global>
            <config>
                <router-id>192.0.2.2</router-id>
                <as>65000</as>
            </config>
            <afi-safis>
                <afi-safi>
                    <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">
↪x:IPV4-UNICAST</afi-safi-name>
                </afi-safi>
                <afi-safi>
                    <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">
↪x:IPV6-UNICAST</afi-safi-name>
                </afi-safi>
            </afi-safis>
        </global>
    </bgp>
</protocol>
```

BGP Peer

Here is an example for BGP peer configuration with enabled IPv4 and IPv6 Unicast family.

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<neighbor xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <neighbor-address>192.0.2.1</neighbor-address>
    <afi-safis>
        <afi-safi>
            <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">x:IPV4-
↪UNICAST</afi-safi-name>
        </afi-safi>
        <afi-safi>
            <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">x:IPV6-
↪UNICAST</afi-safi-name>
```

```
        </afi-safi>
    </afi-safis>
</neighbor>
```

IP Unicast API

Following trees illustrate the BGP IP Unicast routes structures.

IPv4 Unicast Route

```
:(ipv4-routes-case)
   +--ro ipv4-routes
     +--ro ipv4-route* [prefix path-id]
        +--ro prefix        inet:ipv4-prefix
        +--ro path-id       path-id
        +--ro attributes
           +--ro origin
           |  +--ro value    bgp-t:bgp-origin
           +--ro as-path
           |  +--ro segments*
           |     +--ro as-sequence*   inet:as-number
           |     +--ro as-set*        inet:as-number
           +--ro (c-next-hop)?
           |  +--:(ipv4-next-hop-case)
           |  |  +--ro ipv4-next-hop
           |  |     +--ro global?   inet:ipv4-address
           |  +--:(ipv6-next-hop-case)
           |  |  +--ro ipv6-next-hop
           |  |     +--ro global?       inet:ipv6-address
           |  |     +--ro link-local?   inet:ipv6-address
           |  +--:(empty-next-hop-case)
           |     +--ro empty-next-hop?            empty
           +--ro multi-exit-disc
           |  +--ro med?   uint32
           +--ro local-pref
           |  +--ro pref?   uint32
           +--ro atomic-aggregate!
           +--ro aggregator
           |  +--ro as-number?         inet:as-number
           |  +--ro network-address?   inet:ipv4-address
           +--ro communities*
           |  +--ro as-number?   inet:as-number
           |  +--ro semantics?   uint16
           +--ro extended-communities*
           |  +--ro transitive?                              boolean
           |  +--ro (extended-community)?
           |     +--:(as-specific-extended-community-case)
           |     |  +--ro as-specific-extended-community
           |     |     +--ro global-administrator?   short-as-number
           |     |     +--ro local-administrator?    binary
           |     +--:(inet4-specific-extended-community-case)
           |     |  +--ro inet4-specific-extended-community
           |     |     +--ro global-administrator?   inet:ipv4-address
           |     |     +--ro local-administrator?    binary
```

```
|      +--:(opaque-extended-community-case)
|      |  +--ro opaque-extended-community
|      |     +--ro value?   binary
|      +--:(route-target-extended-community-case)
|      |  +--ro route-target-extended-community
|      |     +--ro global-administrator?   short-as-number
|      |     +--ro local-administrator?    binary
|      +--:(route-origin-extended-community-case)
|      |  +--ro route-origin-extended-community
|      |     +--ro global-administrator?   short-as-number
|      |     +--ro local-administrator?    binary
|      +--:(route-target-ipv4-case)
|      |  +--ro route-target-ipv4
|      |     +--ro global-administrator?   inet:ipv4-address
|      |     +--ro local-administrator?    uint16
|      +--:(route-origin-ipv4-case)
|      |  +--ro route-origin-ipv4
|      |     +--ro global-administrator?   inet:ipv4-address
|      |     +--ro local-administrator?    uint16
|      +--:(link-bandwidth-case)
|      |  +--ro link-bandwidth-extended-community
|      |     +--ro bandwidth    netc:bandwidth
|      +--:(as-4-generic-spec-extended-community-case)
|      |  +--ro as-4-generic-spec-extended-community
|      |     +--ro as-4-specific-common
|      |        +--ro as-number              inet:as-number
|      |        +--ro local-administrator    uint16
|      +--:(as-4-route-target-extended-community-case)
|      |  +--ro as-4-route-target-extended-community
|      |     +--ro as-4-specific-common
|      |        +--ro as-number              inet:as-number
|      |        +--ro local-administrator    uint16
|      +--:(as-4-route-origin-extended-community-case)
|      |  +--ro as-4-route-origin-extended-community
|      |     +--ro as-4-specific-common
|      |        +--ro as-number              inet:as-number
|      |        +--ro local-administrator    uint16
|      +--:(encapsulation-case)
|         +--ro encapsulation-extended-community
|            +--ro tunnel-type    encapsulation-tunnel-type
+--ro originator-id
|  +--ro originator?   inet:ipv4-address
+--ro cluster-id
|  +--ro cluster*   bgp-t:cluster-identifier
+--ro aigp
|  +--ro aigp-tlv
|     +--ro metric?   netc:accumulated-igp-metric
+--ro unrecognized-attributes* [type]
   +--ro partial       boolean
   +--ro transitive    boolean
   +--ro type          uint8
   +--ro value         binary
```

IPv6 Unicast Route

```
:(ipv6-routes-case)
   +--ro ipv6-routes
      +--ro ipv6-route* [prefix path-id]
         +--ro prefix        inet:ipv6-prefix
         +--ro path-id       path-id
         +--ro attributes
         ...
```

Usage

IPv4 Unicast

The IPv4 Unicast table in an instance of the speaker's Loc-RIB can be verified via REST:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/loc-rib/tables/bgp-types:ipv4-address-family/bgp-types:unicast-subsequent-address-family/ipv4-routes`

Method: `GET`

Response Body:

```
<ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet">
    <ipv4-route>
        <path-id>0</path-id>
        <prefix>193.0.2.1/32</prefix>
        <attributes>
            <as-path></as-path>
            <origin>
                <value>igp</value>
            </origin>
            <local-pref>
                <pref>100</pref>
            </local-pref>
            <ipv4-next-hop>
                <global>10.0.0.1</global>
            </ipv4-next-hop>
        </attributes>
    </ipv4-route>
</ipv4-routes>
```

IPv6 Unicast

The IPv6 Unicast table in an instance of the speaker's Loc-RIB can be verified via REST:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/loc-rib/tables/bgp-types:ipv4-address-family/bgp-types:unicast-subsequent-address-family/ipv6-routes`

Method: `GET`

Response Body:

```
<ipv6-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet">
    <ipv6-route>
        <path-id>0</path-id>
        <prefix>2a02:b80:0:1::/64</prefix>
        <attributes>
            <as-path></as-path>
            <origin>
                <value>igp</value>
            </origin>
            <local-pref>
                <pref>200</pref>
            </local-pref>
            <ipv6-next-hop>
                <global>2a02:b80:0:2::1</global>
            </ipv6-next-hop>
        </attributes>
    </ipv6-route>
</ipv6-routes>
```

Note: IPv4/6 routes mapping to topology nodes is supported by BGP Topology Provider.

Programming

IPv4 Unicast

This examples show how to originate and remove IPv4 route via programmable RIB. Make sure the *Application Peer* is configured first.

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/bgp-types:unicast-subsequent-address-family/bgp-inet:ipv4-routes`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<ipv4-route xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet">
    <path-id>0</path-id>
    <prefix>10.0.0.11/32</prefix>
    <attributes>
        <as-path></as-path>
        <origin>
            <value>igp</value>
        </origin>
        <local-pref>
            <pref>100</pref>
        </local-pref>
        <ipv4-next-hop>
            <global>10.11.1.1</global>
        </ipv4-next-hop>
    </attributes>
</ipv4-route>
```

To remove the route added above, following request can be used:

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/bgp-types:unicast-subsequent-address-family/bgp-inet:ipv4-routes/ipv4-route/10.0.0.11%2F32/0`

Method: `DELETE`

IPv6 Unicast

This examples show how to originate and remove IPv6 route via programmable RIB:

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv6-address-family/bgp-types:unicast-subsequent-address-family/bgp-inet:ipv6-routes`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<ipv6-route xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet">
    <prefix>2001:db8:30::3/128</prefix>
    <path-id>0</path-id>
    <attributes>
        <ipv6-next-hop>
            <global>2001:db8:1::6</global>
        </ipv6-next-hop>
        <as-path/>
        <origin>
            <value>igp</value>
        </origin>
        <local-pref>
            <pref>100</pref>
        </local-pref>
    </attributes>
</ipv6-route>
```

To remove the route added above, following request can be used:

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv6-address-family/bgp-types:unicast-subsequent-address-family/bgp-inet:ipv6-routes/ipv6-route/2001:db8:30::3%2F128/0`

Method: `DELETE`

References

- Multiprotocol Extensions for BGP-4

IP Labeled Unicast Family

The BGP Labeled Unicast (BGP-LU) Multiprotocol extension is used to distribute a MPLS label that is mapped to a particular route. It can be used to advertise a MPLS transport path between IGP regions and Autonomous Systems. Also, BGP-LU can help to solve the Inter-domain traffic-engineering problem and can be deployed in large-scale data centers along with MPLS and Spring. In addition, IPv6 Labeled Unicast can be used to interconnect IPv6 islands over IPv4/MPLS networks using 6PE.

Contents

Configuration

This section shows a way to enable IPv4 and IPv6 Labeled Unicast family in BGP speaker and peer configuration.

BGP Speaker

To enable IPv4 and IPv6 Labeled Unicast support in BGP plugin, first configure BGP speaker instance:

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<protocol xmlns="http://openconfig.net/yang/network-instance">
    <name>bgp-example</name>
    <identifier xmlns:x="http://openconfig.net/yang/policy-types">x:BGP</identifier>
    <bgp xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
        <global>
            <config>
                <router-id>192.0.2.2</router-id>
                <as>65000</as>
            </config>
```

```
            <afi-safis>
                <afi-safi>
                    <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">
↪x:IPV4-LABELLED-UNICAST</afi-safi-name>
                </afi-safi>
                <afi-safi>
                    <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">
↪x:IPV6-LABELLED-UNICAST</afi-safi-name>
                </afi-safi>
            </afi-safis>
        </global>
    </bgp>
</protocol>
```

BGP Peer

Here is an example for BGP peer configuration with enabled IPv4 and IPv6 Labeled Unicast family.

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<neighbor xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <neighbor-address>192.0.2.1</neighbor-address>
    <afi-safis>
        <afi-safi>
            <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">x:IPV4-
↪LABELLED-UNICAST</afi-safi-name>
        </afi-safi>
        <afi-safi>
            <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">x:IPV6-
↪LABELLED-UNICAST</afi-safi-name>
        </afi-safi>
    </afi-safis>
</neighbor>
```

IP Labeled Unicast API

Following trees illustrate the BGP IP Labeled Unicast routes structures.

IPv4 Labeled Unicast Route

```
:(labeled-unicast-routes-case)
  +--ro labeled-unicast-routes
     +--ro labeled-unicast-route* [route-key path-id]
        +--ro route-key      string
        +--ro label-stack*
        |  +--ro label-value?   netc:mpls-label
```

```
        +--ro prefix?        inet:ip-prefix
        +--ro path-id        path-id
        +--ro attributes
        ...
```

IPv6 Labeled Unicast Route

```
:(labeled-unicast-ipv6-routes-case)
   +--ro labeled-unicast-ipv6-routes
      +--ro labeled-unicast-route* [route-key path-id]
         +--ro route-key      string
         +--ro label-stack*
         |  +--ro label-value?   netc:mpls-label
         +--ro prefix?        inet:ip-prefix
         +--ro path-id        path-id
         +--ro attributes
         ...
```

Usage

The IPv4 Labeled Unicast table in an instance of the speaker's Loc-RIB can be verified via REST:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/loc-rib/tables/bgp-types:ipv4-address-family/bgp-labeled-unicast:labeled-unicast-subsequent-address-family-bgp-labeled-unicast:labeled-unicast-routes`

Method: `GET`

Response Body:

```
<labeled-unicast-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-labeled-unicast
↪">
    <labeled-unicast-route>
        <path-id>0</path-id>
        <route-key>MAA+gRQAAA==</route-key>
        <attributes>
            <local-pref>
                <pref>100</pref>
            </local-pref>
            <ipv4-next-hop>
                <global>200.10.0.101</global>
            </ipv4-next-hop>
            <as-path></as-path>
            <origin>
                <value>igp</value>
            </origin>
        </attributes>
        <label-stack>
            <label-value>1000</label-value>
        </label-stack>
        <prefix>20.0.0.0/24</prefix>
    </labeled-unicast-route>
</labeled-unicast-routes>
```

Programming

IPv4 Labeled

This examples show how to originate and remove IPv4 labeled route via programmable RIB. Make sure the *Application Peer* is configured first.

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/bgp-labeled-unicast:labeled-unicast-subsequent-address-family-bgp-labeled-unicast:labeled-unicast-routes`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<labeled-unicast-route xmlns="urn:opendaylight:params:xml:ns:yang:bgp-labeled-unicast
↪">
    <route-key>label1</route-key>
    <prefix>1.1.1.1/32</prefix>
    <path-id>0</path-id>
    <label-stack>
        <label-value>800322</label-value>
    </label-stack>
    <attributes>
        <ipv4-next-hop>
            <global>199.20.160.41</global>
        </ipv4-next-hop>
        <origin>
            <value>igp</value>
        </origin>
        <as-path/>
        <local-pref>
            <pref>100</pref>
        </local-pref>
    </attributes>
</labeled-unicast-route>
```

In addition, BGP-LU Spring extension allows to attach BGP Prefix SID attribute to the route, in order to signal the BGP-Prefix-SID, where the SR is applied to MPLS dataplane.

```
<bgp-prefix-sid>
    <bgp-prefix-sid-tlvs>
        <label-index-tlv xmlns="urn:opendaylight:params:xml:ns:yang:bgp-labeled-
↪unicast">322</label-index-tlv>
    </bgp-prefix-sid-tlvs>
    <bgp-prefix-sid-tlvs>
        <srgb-value xmlns="urn:opendaylight:params:xml:ns:yang:bgp-labeled-unicast">
            <base>800000</base>
            <range>4095</range>
        </srgb-value>
    </bgp-prefix-sid-tlvs>
</bgp-prefix-sid>
```

To remove the route added above, following request can be used:

URL: /restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/bgp-labeled-unicast:labeled-unicast-subsequent-address-family/bgp-labeled-unicast:labeled-unicast-routes/bgp-labeled-unicast:labeled-unicast-route/label1/0

Method: DELETE

IPv6 Labeled

This examples show how to originate and remove IPv6 labeled route via programmable RIB.

URL: /restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/bgp-labeled-unicast:labeled-unicast-subsequent-address-family/bgp-labeled-unicast:labeled-unicast-ipv6-routes

Method: POST

Content-Type: application/xml

Request Body:

```
<labeled-unicast-route xmlns="urn:opendaylight:params:xml:ns:yang:bgp-labeled-unicast
↪">
    <route-key>label1</route-key>
    <prefix>2001:db8:30::3/128</prefix>
    <path-id>0</path-id>
    <label-stack>
        <label-value>123</label-value>
    </label-stack>
    <attributes>
        <ipv6-next-hop>
            <global>2003:4:5:6::7</global>
        </ipv6-next-hop>
        <origin>
            <value>igp</value>
        </origin>
        <as-path/>
        <local-pref>
            <pref>100</pref>
        </local-pref>
    </attributes>
</labeled-unicast-route>
```

To remove the route added above, following request can be used:

URL: /restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/bgp-labeled-unicast:labeled-unicast-subsequent-address-family/bgp-labeled-unicast:labeled-unicast-ipv6-routes/bgp-labeled-unicast:labeled-unicast-route/label1/0

Method: DELETE

References

- Carrying Label Information in BGP-4
- Segment Routing Prefix SID extensions for BGP

- Connecting IPv6 Islands over IPv4 MPLS Using IPv6 Provider Edge Routers (6PE)
- BGP-Prefix Segment in large-scale data centers
- Egress Peer Engineering using BGP-LU

IP L3VPN Family

The BGP/MPLS IP Virtual Private Networks (BGP L3VPN) Multiprotocol extension can be used to exchange particular VPN (customer) routes among the provider's routers attached to that VPN. Also, routes are distributed to specific VPN remote sites.

Contents

Configuration

This section shows a way to enable IPv4 and IPv6 L3VPN family in BGP speaker and peer configuration.

BGP Speaker

To enable IPv4 and IPv6 L3VPN support in BGP plugin, first configure BGP speaker instance:

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<protocol xmlns="http://openconfig.net/yang/network-instance">
    <name>bgp-example</name>
    <identifier xmlns:x="http://openconfig.net/yang/policy-types">x:BGP</identifier>
    <bgp xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
        <global>
```

```
            <config>
                <router-id>192.0.2.2</router-id>
                <as>65000</as>
            </config>
            <afi-safis>
                <afi-safi>
                    <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">
↪x:L3VPN-IPV4-UNICAST</afi-safi-name>
                </afi-safi>
                <afi-safi>
                    <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">
↪x:L3VPN-IPV6-UNICAST</afi-safi-name>
                </afi-safi>
            </afi-safis>
        </global>
    </bgp>
</protocol>
```

BGP Peer

Here is an example for BGP peer configuration with enabled IPv4 and IPv6 L3VPN family.

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<neighbor xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <neighbor-address>192.0.2.1</neighbor-address>
    <afi-safis>
        <afi-safi>
            <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">x:L3VPN-
↪IPV4-UNICAST</afi-safi-name>
        </afi-safi>
        <afi-safi>
            <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">x:L3VPN-
↪IPV6-UNICAST</afi-safi-name>
        </afi-safi>
    </afi-safis>
</neighbor>
```

IP L3VPN API

Following trees illustrate the BGP IP L3VPN routes structures.

IPv4 L3VPN Route

```
:(vpn-ipv4-routes-case)
   +--ro vpn-ipv4-routes
      +--ro vpn-route* [route-key]
         +--ro route-key               string
         +--ro label-stack*
         |  +--ro label-value?   netc:mpls-label
         +--ro prefix?                 inet:ip-prefix
         +--ro path-id?                path-id
         +--ro route-distinguisher?    bgp-t:route-distinguisher
         +--ro attributes
         ...
```

IPv6 L3VPN Route

```
:(vpn-ipv6-routes-case)
   +--ro vpn-ipv6-routes
      +--ro vpn-route* [route-key]
         +--ro route-key               string
         +--ro label-stack*
         |  +--ro label-value?   netc:mpls-label
         +--ro prefix?                 inet:ip-prefix
         +--ro path-id?                path-id
         +--ro route-distinguisher?    bgp-t:route-distinguisher
         +--ro attributes
         ...
```

Usage

IPv4 L3VPN

The IPv4 L3VPN table in an instance of the speaker's Loc-RIB can be verified via REST:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/loc-rib/tables/bgp-types:ipv4-address-family/bgp-types:mpls-labeled-vpn-subsequent-address-family/bgp-vpn-ipv4:vpn-ipv4-routes`

Method: `GET`

Response Body:

```
<vpn-ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-vpn-ipv4">
    <vpn-route>
        <route-key>cAXdYQABrBAALABlCgIi</route-key>
        <label-stack>
            <label-value>24022</label-value>
        </label-stack>
        <attributes>
            <extended-communities>
                <transitive>true</transitive>
                <route-target-extended-community>
                    <global-administrator>65000</global-administrator>
                    <local-administrator>AAAAZQ==</local-administrator>
                </route-target-extended-community>
            </extended-communities>
```

```
            <origin>
                <value>igp</value>
            </origin>
            <as-path></as-path>
            <local-pref>
                <pref>100</pref>
            </local-pref>
            <ipv4-next-hop>
                <global>127.16.0.44</global>
            </ipv4-next-hop>
        </attributes>
        <route-distinguisher>172.16.0.44:101</route-distinguisher>
        <prefix>10.2.34.0/24</prefix>
    </vpn-route>
</vpn-ipv4-routes>
```

IPv6 L3VPN

The IPv6 L3VPN table in an instance of the speaker's Loc-RIB can be verified via REST:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/loc-rib/tables/bgp-types:ipv6-address-family/bgp-types:mpls-labeled-vpn-subsequent-address-family/bgp-vpn-ipv6:vpn-ipv6-routes`

Method: `GET`

Response Body:

```
<vpn-ipv6-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-vpn-ipv6">
    <vpn-route>
        <route-key>mAXdcQABrBAALABlKgILgAAAAAE=</route-key>
        <label-stack>
            <label-value>24023</label-value>
        </label-stack>
        <attributes>
            <local-pref>
                <pref>100</pref>
            </local-pref>
            <extended-communities>
                <route-target-extended-community>
                    <global-administrator>65000</global-administrator>
                    <local-administrator>AAAAZQ==</local-administrator>
                </route-target-extended-community>
                <transitive>true</transitive>
            </extended-communities>
            <ipv6-next-hop>
                <global>2a02:b80:0:2::1</global>
            </ipv6-next-hop>
            <origin>
                <value>igp</value>
            </origin>
            <as-path></as-path>
        </attributes>
        <route-distinguisher>172.16.0.44:101</route-distinguisher>
        <prefix>2a02:b80:0:1::/64</prefix>
    </vpn-route>
</vpn-ipv6-routes>
```

Programming

This examples show how to originate and remove IPv4 L3VPN route via programmable RIB. Make sure the *Application Peer* is configured first.

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/bgp-types:mpls-labeled-vpn-subsequent-address-family/bgp-vpn-ipv4:vpn-ipv4-routes`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<vpn-route xmlns="urn:opendaylight:params:xml:ns:yang:bgp-vpn-ipv4">
    <route-key>vpn1</route-key>
    <label-stack>
        <label-value>123</label-value>
    </label-stack>
    <route-distinguisher>429496729:1</route-distinguisher>
    <prefix>2.2.2.2/32</prefix>
    <attributes>
        <ipv4-next-hop>
            <global>199.20.166.41</global>
        </ipv4-next-hop>
        <as-path/>
        <origin>
            <value>igp</value>
        </origin>
        <extended-communities>
            <route-target-extended-community>
                <global-administrator>65000</global-administrator>
                <local-administrator>AAAAZQ==</local-administrator>
            </route-target-extended-community>
            <transitive>true</transitive>
        </extended-communities>
    </attributes>
</vpn-route>
```

To remove the route added above, following request can be used:

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/bgp-types:mpls-labeled-vpn-subsequent-address-family/bgp-vpn-ipv4:vpn-ipv4-routes/vpn-route/vpn1`

Method: `DELETE`

References

- BGP/MPLS IP Virtual Private Networks (VPNs)
- BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN
- BGP/MPLS VPN Virtual PE

Link-State Family

The BGP Link-State (BGP-LS) Multiprotocol extension allows to distribute Link-State and Traffic Engineering (TE) information. This information is typically distributed by IGP routing protocols with in the network, limiting LSDB or TED visibility to the IGP area. The BGP-LS-enabled routers are capable to collect such information from networks (multiple IGP areas, inter-AS) and share with external components (i.e. OpenDaylight BGP). The information is applicable in ALTO servers and PCEs, as both need to gather information about topologies. In addition, link-state information is extended to carry segment information (Spring).

Contents

Configuration

This section shows a way to enable IPv4 and IPv6 Labeled Unicast family in BGP speaker and peer configuration.

BGP Speaker

To enable BGP-LS support in BGP plugin, first configure BGP speaker instance:

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<protocol xmlns="http://openconfig.net/yang/network-instance">
    <name>bgp-example</name>
    <identifier xmlns:x="http://openconfig.net/yang/policy-types">x:BGP</identifier>
    <bgp xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
        <global>
            <config>
                <router-id>192.0.2.2</router-id>
                <as>65000</as>
            </config>
            <afi-safis>
                <afi-safi>
                    <afi-safi-name>LINKSTATE</afi-safi-name>
                </afi-safi>
            </afi-safis>
```

```
        </global>
    </bgp>
</protocol>
```

Linkstate path attribute

IANA allocation for BGP-LS path attribute is TYPE 29. Some older BGP-LS implementations might still require earliest asigned allocation TYPE 99. To use TYPE = 99, you need to set value bellow to false.

URL: `/restconf/config/bgp-linkstate-app-config:bgp-linkstate-app-config`

Method: `PUT`

Content-Type: `application/xml`

Request Body:

```
<bgp-linkstate-app-config xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:bgp:linkstate-app-config">
    <iana-linkstate-attribute-type>false</iana-linkstate-attribute-type>
</bgp-linkstate-app-config>
```

BGP Peer

Here is an example for BGP peer configuration with enabled BGP-LS family.

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<neighbor xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <neighbor-address>192.0.2.1</neighbor-address>
    <afi-safis>
        <afi-safi>
            <afi-safi-name>LINKSTATE</afi-safi-name>
        </afi-safi>
    </afi-safis>
</neighbor>
```

Link-State Route API

Following tree illustrate the BGP Link-State route structure.

```
:(linkstate-routes-case)
   +--ro linkstate-routes
      +--ro linkstate-route* [route-key]
         +--ro route-key                      binary
         +--ro protocol-id                    protocol-id
         +--ro identifier                     identifier
```

```
         +--ro (object-type)?
         |  +--:(node-case)
         |  |  +--ro node-descriptors
         |  |     +--ro as-number?         inet:as-number
         |  |     +--ro area-id?           area-identifier
         |  |     +--ro domain-id?         domain-identifier
         |  |     +--ro (c-router-identifier)?
         |  |        +--:(isis-node-case)
         |  |        |  +--ro isis-node
         |  |        |     +--ro iso-system-id    netc:iso-system-identifier
         |  |        +--:(isis-pseudonode-case)
         |  |        |  +--ro isis-pseudonode
         |  |        |     +--ro is-is-router-identifier
         |  |        |     |  +--ro iso-system-id    netc:iso-system-identifier
         |  |        |     +--ro psn                         uint8
         |  |        +--:(ospf-node-case)
         |  |        |  +--ro ospf-node
         |  |        |     +--ro ospf-router-id    uint32
         |  |        +--:(ospf-pseudonode-case)
         |  |           +--ro ospf-pseudonode
         |  |              +--ro ospf-router-id    uint32
         |  |              +--ro lan-interface     ospf-interface-identifier
         |  +--:(link-case)
         |  |  +--ro local-node-descriptors
         |  |  |  +--ro as-number?         inet:as-number
         |  |  |  +--ro area-id?           area-identifier
         |  |  |  +--ro domain-id?         domain-identifier
         |  |  |  +--ro (c-router-identifier)?
         |  |  |  |  +--:(isis-node-case)
         |  |  |  |  |  +--ro isis-node
         |  |  |  |  |     +--ro iso-system-id    netc:iso-system-identifier
         |  |  |  |  +--:(isis-pseudonode-case)
         |  |  |  |  |  +--ro isis-pseudonode
         |  |  |  |  |     +--ro is-is-router-identifier
         |  |  |  |  |     |  +--ro iso-system-id    netc:iso-system-identifier
         |  |  |  |  |     +--ro psn                         uint8
         |  |  |  |  +--:(ospf-node-case)
         |  |  |  |  |  +--ro ospf-node
         |  |  |  |  |     +--ro ospf-router-id    uint32
         |  |  |  |  +--:(ospf-pseudonode-case)
         |  |  |  |     +--ro ospf-pseudonode
         |  |  |  |        +--ro ospf-router-id    uint32
         |  |  |  |        +--ro lan-interface     ospf-interface-identifier
         |  |  |  +--ro bgp-router-id?     inet:ipv4-address
         |  |  |  +--ro member-asn?        inet:as-number
         |  |  +--ro remote-node-descriptors
         |  |  |  +--ro as-number?         inet:as-number
         |  |  |  +--ro area-id?           area-identifier
         |  |  |  +--ro domain-id?         domain-identifier
         |  |  |  +--ro (c-router-identifier)?
         |  |  |  |  +--:(isis-node-case)
         |  |  |  |  |  +--ro isis-node
         |  |  |  |  |     +--ro iso-system-id    netc:iso-system-identifier
         |  |  |  |  +--:(isis-pseudonode-case)
         |  |  |  |  |  +--ro isis-pseudonode
         |  |  |  |  |     +--ro is-is-router-identifier
         |  |  |  |  |     |  +--ro iso-system-id    netc:iso-system-identifier
         |  |  |  |  |     +--ro psn                         uint8
```

```
|  |  |  |  +--:(ospf-node-case)
|  |  |  |  |  +--ro ospf-node
|  |  |  |  |     +--ro ospf-router-id    uint32
|  |  |  |  +--:(ospf-pseudonode-case)
|  |  |  |     +--ro ospf-pseudonode
|  |  |  |        +--ro ospf-router-id    uint32
|  |  |  |        +--ro lan-interface     ospf-interface-identifier
|  |  |  +--ro bgp-router-id?     inet:ipv4-address
|  |  |  +--ro member-asn?        inet:as-number
|  |  +--ro link-descriptors
|  |     +--ro link-local-identifier?    uint32
|  |     +--ro link-remote-identifier?   uint32
|  |     +--ro ipv4-interface-address?   ipv4-interface-identifier
|  |     +--ro ipv6-interface-address?   ipv6-interface-identifier
|  |     +--ro ipv4-neighbor-address?    ipv4-interface-identifier
|  |     +--ro ipv6-neighbor-address?    ipv6-interface-identifier
|  |     +--ro multi-topology-id?        topology-identifier
|  +--:(prefix-case)
|  |  +--ro advertising-node-descriptors
|  |  |  +--ro as-number?         inet:as-number
|  |  |  +--ro area-id?           area-identifier
|  |  |  +--ro domain-id?         domain-identifier
|  |  |  +--ro (c-router-identifier)?
|  |  |     +--:(isis-node-case)
|  |  |     |  +--ro isis-node
|  |  |     |     +--ro iso-system-id    netc:iso-system-identifier
|  |  |     +--:(isis-pseudonode-case)
|  |  |     |  +--ro isis-pseudonode
|  |  |     |     +--ro is-is-router-identifier
|  |  |     |     |  +--ro iso-system-id    netc:iso-system-identifier
|  |  |     |     +--ro psn                        uint8
|  |  |     +--:(ospf-node-case)
|  |  |     |  +--ro ospf-node
|  |  |     |     +--ro ospf-router-id    uint32
|  |  |     +--:(ospf-pseudonode-case)
|  |  |        +--ro ospf-pseudonode
|  |  |           +--ro ospf-router-id    uint32
|  |  |           +--ro lan-interface     ospf-interface-identifier
|  |  +--ro prefix-descriptors
|  |     +--ro multi-topology-id?             topology-identifier
|  |     +--ro ospf-route-type?               ospf-route-type
|  |     +--ro ip-reachability-information?   inet:ip-prefix
|  +--:(te-lsp-case)
|     +--ro (address-family)?
|     |  +--:(ipv4-case)
|     |  |  +--ro ipv4-tunnel-sender-address      inet:ipv4-address
|     |  |  +--ro ipv4-tunnel-endpoint-address    inet:ipv4-address
|     |  +--:(ipv6-case)
|     |     +--ro ipv6-tunnel-sender-address      inet:ipv6-address
|     |     +--ro ipv6-tunnel-endpoint-address    inet:ipv6-address
|     +--ro tunnel-id?                      rsvp:tunnel-id
|     +--ro lsp-id?                         rsvp:lsp-id
+--ro attributes
   +--ro (link-state-attribute)?
      +--:(node-attributes-case)
      |  +--ro node-attributes
      |     +--ro topology-identifier*   topology-identifier
      |     +--ro node-flags?            node-flag-bits
```

```
|     +--ro isis-area-id*          isis-area-identifier
|     +--ro dynamic-hostname?      string
|     +--ro ipv4-router-id?        ipv4-router-identifier
|     +--ro ipv6-router-id?        ipv6-router-identifier
|     +--ro sr-capabilities
|     |  +--ro mpls-ipv4?      boolean
|     |  +--ro mpls-ipv6?      boolean
|     |  +--ro sr-ipv6?        boolean
|     |  +--ro range-size?     uint32
|     |  +--ro (sid-label-index)?
|     |     +--:(local-label-case)
|     |     |  +--ro local-label?    netc:mpls-label
|     |     +--:(ipv6-address-case)
|     |     |  +--ro ipv6-address?   inet:ipv6-address
|     |     +--:(sid-case)
|     |        +--ro sid?            uint32
|     +--ro sr-algorithm
|        +--ro algorithms*   algorithm
+--:(link-attributes-case)
|  +--ro link-attributes
|     +--ro local-ipv4-router-id?       ipv4-router-identifier
|     +--ro local-ipv6-router-id?       ipv6-router-identifier
|     +--ro remote-ipv4-router-id?      ipv4-router-identifier
|     +--ro remote-ipv6-router-id?      ipv6-router-identifier
|     +--ro mpls-protocol?              mpls-protocol-mask
|     +--ro te-metric?                  netc:te-metric
|     +--ro metric?                     netc:metric
|     +--ro shared-risk-link-groups*    rsvp:srlg-id
|     +--ro link-name?                  string
|     +--ro max-link-bandwidth?         netc:bandwidth
|     +--ro max-reservable-bandwidth?   netc:bandwidth
|     +--ro unreserved-bandwidth* [priority]
|     |  +--ro priority     uint8
|     |  +--ro bandwidth?   netc:bandwidth
|     +--ro link-protection?            link-protection-type
|     +--ro admin-group?                administrative-group
|     +--ro sr-adj-ids*
|     |  +--ro (flags)?
|     |  |  +--:(ospf-adj-flags-case)
|     |  |  |  +--ro backup?           boolean
|     |  |  |  +--ro set?              boolean
|     |  |  +--:(isis-adj-flags-case)
|     |  |     +--ro backup?           boolean
|     |  |     +--ro set?              boolean
|     |  |     +--ro address-family?   boolean
|     |  +--ro weight?           weight
|     |  +--ro (sid-label-index)?
|     |     +--:(local-label-case)
|     |     |  +--ro local-label?      netc:mpls-label
|     |     +--:(ipv6-address-case)
|     |     |  +--ro ipv6-address?     inet:ipv6-address
|     |     +--:(sid-case)
|     |        +--ro sid?              uint32
|     +--ro sr-lan-adj-ids*
|     |  +--ro (flags)?
|     |  |  +--:(ospf-adj-flags-case)
|     |  |  |  +--ro backup?           boolean
|     |  |  |  +--ro set?              boolean
```

```
|      |  |  +--:(isis-adj-flags-case)
|      |  |     +--ro backup?           boolean
|      |  |     +--ro set?              boolean
|      |  |     +--ro address-family?   boolean
|      |  +--ro weight?           weight
|      |  +--ro iso-system-id?    netc:iso-system-identifier
|      |  +--ro neighbor-id?      inet:ipv4-address
|      |  +--ro (sid-label-index)?
|      |     +--:(local-label-case)
|      |     |  +--ro local-label?      netc:mpls-label
|      |     +--:(ipv6-address-case)
|      |     |  +--ro ipv6-address?     inet:ipv6-address
|      |     +--:(sid-case)
|      |        +--ro sid?              uint32
|      +--ro peer-node-sid
|      |  +--ro weight?         weight
|      |  +--ro (sid-label-index)?
|      |     +--:(local-label-case)
|      |     |  +--ro local-label?    netc:mpls-label
|      |     +--:(ipv6-address-case)
|      |     |  +--ro ipv6-address?   inet:ipv6-address
|      |     +--:(sid-case)
|      |        +--ro sid?            uint32
|      +--ro peer-adj-sid
|      |  +--ro weight?         weight
|      |  +--ro (sid-label-index)?
|      |     +--:(local-label-case)
|      |     |  +--ro local-label?    netc:mpls-label
|      |     +--:(ipv6-address-case)
|      |     |  +--ro ipv6-address?   inet:ipv6-address
|      |     +--:(sid-case)
|      |        +--ro sid?            uint32
|      +--ro peer-set-sids*
|         +--ro weight?         weight
|         +--ro (sid-label-index)?
|            +--:(local-label-case)
|            |  +--ro local-label?    netc:mpls-label
|            +--:(ipv6-address-case)
|            |  +--ro ipv6-address?   inet:ipv6-address
|            +--:(sid-case)
|               +--ro sid?            uint32
+--:(prefix-attributes-case)
|  +--ro prefix-attributes
|     +--ro igp-bits
|     |  x--ro up-down?               bits
|     |  +--ro is-is-up-down?         boolean
|     |  +--ro ospf-no-unicast?       boolean
|     |  +--ro ospf-local-address?    boolean
|     |  +--ro ospf-propagate-nssa?   boolean
|     +--ro route-tags*                route-tag
|     +--ro extended-tags*             extended-route-tag
|     +--ro prefix-metric?             netc:igp-metric
|     +--ro ospf-forwarding-address?   inet:ip-address
|     +--ro sr-prefix
|     |  +--ro (flags)?
|     |  |  +--:(isis-prefix-flags-case)
|     |  |  |  +--ro no-php?            boolean
|     |  |  |  +--ro explicit-null?     boolean
```

```
                |      |  |  |  +--ro readvertisement?   boolean
                |      |  |  |  +--ro node-sid?          boolean
                |      |  |  +--:(ospf-prefix-flags-case)
                |      |  |     +--ro no-php?            boolean
                |      |  |     +--ro explicit-null?     boolean
                |      |  |     +--ro mapping-server?    boolean
                |      |  +--ro algorithm?         algorithm
                |      |  +--ro (sid-label-index)?
                |      |     +--:(local-label-case)
                |      |     |  +--ro local-label?       netc:mpls-label
                |      |     +--:(ipv6-address-case)
                |      |     |  +--ro ipv6-address?      inet:ipv6-address
                |      |     +--:(sid-case)
                |      |        +--ro sid?               uint32
                |      +--ro ipv6-sr-prefix
                |      |  +--ro algorithm?   algorithm
                |      +--ro sr-range
                |      |  +--ro inter-area?   boolean
                |      |  +--ro range-size?   uint16
                |      |  +--ro sub-tlvs*
                |      |     +--ro (range-sub-tlv)?
                |      |        +--:(binding-sid-tlv-case)
                |      |        |  +--ro weight?                weight
                |      |        |  +--ro (flags)?
                |      |        |  |  +--:(isis-binding-flags-case)
                |      |        |  |  |  +--ro address-family?        boolean
                |      |        |  |  |  +--ro mirror-context?        boolean
                |      |        |  |  |  +--ro spread-tlv?            boolean
                |      |        |  |  |  +--ro leaked-from-level-2?   boolean
                |      |        |  |  |  +--ro attached-flag?         boolean
                |      |        |  |  +--:(ospf-binding-flags-case)
                |      |        |  |     +--ro mirroring?             boolean
                |      |        |  +--ro binding-sub-tlvs*
                |      |        |     +--ro (binding-sub-tlv)?
                |      |        |        +--:(prefix-sid-case)
                |      |        |        |  +--ro (flags)?
                |      |        |        |  |  +--:(isis-prefix-flags-case)
                |      |        |        |  |  |  +--ro no-php?            boolean
                |      |        |        |  |  |  +--ro explicit-null?     boolean
                |      |        |        |  |  |  +--ro readvertisement?   boolean
                |      |        |        |  |  |  +--ro node-sid?          boolean
                |      |        |        |  |  +--:(ospf-prefix-flags-case)
                |      |        |        |  |     +--ro no-php?            boolean
                |      |        |        |  |     +--ro explicit-null?     boolean
                |      |        |        |  |     +--ro mapping-server?    boolean
                |      |        |        |  +--ro algorithm?         algorithm
                |      |        |        |  +--ro (sid-label-index)?
                |      |        |        |     +--:(local-label-case)
                |      |        |        |     |  +--ro local-label?       netc:mpls-
↪label
                |      |        |        |     +--:(ipv6-address-case)
                |      |        |        |     |  +--ro ipv6-address?      inet:ipv6-
↪address
                |      |        |        |     +--:(sid-case)
                |      |        |        |        +--ro sid?               uint32
                |      |        |        +--:(ipv6-prefix-sid-case)
                |      |        |        |  +--ro algorithm?         algorithm
                |      |        |        +--:(sid-label-case)
```

```
                |      |         |         |  +--ro (sid-label-index)?
                |      |         |         |     +--:(local-label-case)
                |      |         |         |     |  +--ro local-label?       netc:mpls-
↪label
                |      |         |         |     +--:(ipv6-address-case)
                |      |         |         |     |  +--ro ipv6-address?      inet:ipv6-
↪address
                |      |         |         |     +--:(sid-case)
                |      |         |         |        +--ro sid?               uint32
                |      |         |        +--:(ero-metric-case)
                |      |         |        |  +--ro ero-metric?        netc:te-metric
                |      |         |        +--:(ipv4-ero-case)
                |      |         |        |  +--ro loose?             boolean
                |      |         |        |  +--ro address            inet:ipv4-address
                |      |         |        +--:(ipv6-ero-case)
                |      |         |        |  +--ro loose?             boolean
                |      |         |        |  +--ro address            inet:ipv6-address
                |      |         |        +--:(unnumbered-interface-id-ero-case)
                |      |         |        |  +--ro loose?             boolean
                |      |         |        |  +--ro router-id?         uint32
                |      |         |        |  +--ro interface-id?      uint32
                |      |         |        +--:(ipv4-ero-backup-case)
                |      |         |        |  +--ro loose?             boolean
                |      |         |        |  +--ro address            inet:ipv4-address
                |      |         |        +--:(ipv6-ero-backup-case)
                |      |         |        |  +--ro loose?             boolean
                |      |         |        |  +--ro address            inet:ipv6-address
                |      |         |        +--:(unnumbered-interface-id-backup-ero-case)
                |      |         |           +--ro loose?             boolean
                |      |         |           +--ro router-id?         uint32
                |      |         |           +--ro interface-id?      uint32
                |      |        +--:(prefix-sid-tlv-case)
                |      |        |  +--ro (flags)?
                |      |        |  |  +--:(isis-prefix-flags-case)
                |      |        |  |  |  +--ro no-php?                boolean
                |      |        |  |  |  +--ro explicit-null?         boolean
                |      |        |  |  |  +--ro readvertisement?       boolean
                |      |        |  |  |  +--ro node-sid?              boolean
                |      |        |  |  +--:(ospf-prefix-flags-case)
                |      |        |  |     +--ro no-php?                boolean
                |      |        |  |     +--ro explicit-null?         boolean
                |      |        |  |     +--ro mapping-server?        boolean
                |      |        |  +--ro algorithm?             algorithm
                |      |        |  +--ro (sid-label-index)?
                |      |        |     +--:(local-label-case)
                |      |        |     |  +--ro local-label?           netc:mpls-label
                |      |        |     +--:(ipv6-address-case)
                |      |        |     |  +--ro ipv6-address?          inet:ipv6-address
                |      |        |     +--:(sid-case)
                |      |        |        +--ro sid?                   uint32
                |      |        +--:(ipv6-prefix-sid-tlv-case)
                |      |        |  +--ro algorithm?             algorithm
                |      |        +--:(sid-label-tlv-case)
                |      |           +--ro (sid-label-index)?
                |      |              +--:(local-label-case)
                |      |              |  +--ro local-label?           netc:mpls-label
                |      |              +--:(ipv6-address-case)
                |      |                 +--ro ipv6-address?          inet:ipv6-address
```

```
|     |              +--:(sid-case)
|     |                 +--ro sid?                    uint32
|     +--ro sr-binding-sid-labels*
|        +--ro weight?                weight
|        +--ro (flags)?
|        |  +--:(isis-binding-flags-case)
|        |  |  +--ro address-family?        boolean
|        |  |  +--ro mirror-context?        boolean
|        |  |  +--ro spread-tlv?            boolean
|        |  |  +--ro leaked-from-level-2?   boolean
|        |  |  +--ro attached-flag?         boolean
|        |  +--:(ospf-binding-flags-case)
|        |     +--ro mirroring?             boolean
|        +--ro binding-sub-tlvs*
|           +--ro (binding-sub-tlv)?
|              +--:(prefix-sid-case)
|              |  +--ro (flags)?
|              |  |  +--:(isis-prefix-flags-case)
|              |  |  |  +--ro no-php?            boolean
|              |  |  |  +--ro explicit-null?     boolean
|              |  |  |  +--ro readvertisement?   boolean
|              |  |  |  +--ro node-sid?          boolean
|              |  |  +--:(ospf-prefix-flags-case)
|              |  |     +--ro no-php?            boolean
|              |  |     +--ro explicit-null?     boolean
|              |  |     +--ro mapping-server?    boolean
|              |  +--ro algorithm?         algorithm
|              |  +--ro (sid-label-index)?
|              |     +--:(local-label-case)
|              |     |  +--ro local-label?       netc:mpls-label
|              |     +--:(ipv6-address-case)
|              |     |  +--ro ipv6-address?      inet:ipv6-address
|              |     +--:(sid-case)
|              |        +--ro sid?               uint32
|              +--:(ipv6-prefix-sid-case)
|              |  +--ro algorithm?         algorithm
|              +--:(sid-label-case)
|              |  +--ro (sid-label-index)?
|              |     +--:(local-label-case)
|              |     |  +--ro local-label?       netc:mpls-label
|              |     +--:(ipv6-address-case)
|              |     |  +--ro ipv6-address?      inet:ipv6-address
|              |     +--:(sid-case)
|              |        +--ro sid?               uint32
|              +--:(ero-metric-case)
|              |  +--ro ero-metric?        netc:te-metric
|              +--:(ipv4-ero-case)
|              |  +--ro loose?             boolean
|              |  +--ro address            inet:ipv4-address
|              +--:(ipv6-ero-case)
|              |  +--ro loose?             boolean
|              |  +--ro address            inet:ipv6-address
|              +--:(unnumbered-interface-id-ero-case)
|              |  +--ro loose?             boolean
|              |  +--ro router-id?         uint32
|              |  +--ro interface-id?      uint32
|              +--:(ipv4-ero-backup-case)
|              |  +--ro loose?             boolean
```

```
|              |  +--ro address            inet:ipv4-address
|              +--:(ipv6-ero-backup-case)
|              |  +--ro loose?             boolean
|              |  +--ro address            inet:ipv6-address
|              +--:(unnumbered-interface-id-backup-ero-case)
|                 +--ro loose?             boolean
|                 +--ro router-id?         uint32
|                 +--ro interface-id?      uint32
x--:(te-lsp-attributes-case)
   +--ro te-lsp-attributes
```

Usage

The Link-State table in a instance of the speaker's Loc-RIB can be verified via REST:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/loc-rib/tables/bgp-linkstate:linkstate-address-family/bgp-linkstate:linkstate-subsequent-address-family/linkstate-routes`

Method: `GET`

Response Body:

```
<linkstate-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-linkstate">
   ...
</linkstate-routes>
```

Note: Link-State routes mapping to topology links/nodes/prefixes is supported by BGP Topology Provider.

References

- North-Bound Distribution of Link-State and Traffic Engineering (TE) Information Using BGP
- BGP Link-State extensions for Segment Routing
- Segment Routing BGP Egress Peer Engineering BGP-LS Extensions
- BGP Link-State Information Distribution Implementation Report

Flow Specification Family

The BGP Flow Specification (BGP-FS) Multiprotocol extension can be used to distribute traffic flow specifications. For example, the BGP-FS can be used in a case of (distributed) denial-of-service (DDoS) attack mitigation procedures and traffic filtering (BGP/MPLS VPN service, DC).

Contents

Configuration

This section shows a way to enable BGP-FS family in BGP speaker and peer configuration.

BGP Speaker

To enable BGP-FS support in BGP plugin, first configure BGP speaker instance:

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<protocol xmlns="http://openconfig.net/yang/network-instance">
    <name>bgp-example</name>
    <identifier xmlns:x="http://openconfig.net/yang/policy-types">x:BGP</identifier>
    <bgp xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
        <global>
            <config>
                <router-id>192.0.2.2</router-id>
                <as>65000</as>
            </config>
            <afi-safis>
                <afi-safi>
                    <afi-safi-name>IPV4-FLOW</afi-safi-name>
                </afi-safi>
                <afi-safi>
                    <afi-safi-name>IPV6-FLOW</afi-safi-name>
                </afi-safi>
                <afi-safi>
                    <afi-safi-name>IPV4-L3VPN-FLOW</afi-safi-name>
                </afi-safi>
```

```
                <afi-safi>
                    <afi-safi-name>IPV6-L3VPN-FLOW</afi-safi-name>
                </afi-safi>
            </afi-safis>
        </global>
    </bgp>
</protocol>
```

BGP Peer

Here is an example for BGP peer configuration with enabled BGP-FS family.

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<neighbor xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <neighbor-address>192.0.2.1</neighbor-address>
    <afi-safis>
        <afi-safi>
            <afi-safi-name>IPV4-FLOW</afi-safi-name>
        </afi-safi>
        <afi-safi>
            <afi-safi-name>IPV6-FLOW</afi-safi-name>
        </afi-safi>
        <afi-safi>
            <afi-safi-name>IPV4-L3VPN-FLOW</afi-safi-name>
        </afi-safi>
        <afi-safi>
            <afi-safi-name>IPV6-L3VPN-FLOW</afi-safi-name>
        </afi-safi>
    </afi-safis>
</neighbor>
```

Flow Specification API

Following trees illustrate the BGP Flow Specification routes structure.

IPv4 Flow Specification Route

```
:(flowspec-routes-case)
  +--ro flowspec-routes
     +--ro flowspec-route* [route-key path-id]
        +--ro route-key     string
        +--ro flowspec*
        |  +--ro (flowspec-type)?
        |     +--:(port-case)
        |     |  +--ro ports*
```

```
|      |     +--ro op?      numeric-operand
|      |     +--ro value?   uint16
|      +--:(destination-port-case)
|      |  +--ro destination-ports*
|      |     +--ro op?      numeric-operand
|      |     +--ro value?   uint16
|      +--:(source-port-case)
|      |  +--ro source-ports*
|      |     +--ro op?      numeric-operand
|      |     +--ro value?   uint16
|      +--:(icmp-type-case)
|      |  +--ro types*
|      |     +--ro op?      numeric-operand
|      |     +--ro value?   uint8
|      +--:(icmp-code-case)
|      |  +--ro codes*
|      |     +--ro op?      numeric-operand
|      |     +--ro value?   uint8
|      +--:(tcp-flags-case)
|      |  +--ro tcp-flags*
|      |     +--ro op?      bitmask-operand
|      |     +--ro value?   uint16
|      +--:(packet-length-case)
|      |  +--ro packet-lengths*
|      |     +--ro op?      numeric-operand
|      |     +--ro value?   uint16
|      +--:(dscp-case)
|      |  +--ro dscps*
|      |     +--ro op?      numeric-operand
|      |     +--ro value?   dscp
|      +--:(fragment-case)
|      |  +--ro fragments*
|      |     +--ro op?      bitmask-operand
|      |     +--ro value?   fragment
|      +--:(destination-prefix-case)
|      |  +--ro destination-prefix?   inet:ipv4-prefix
|      +--:(source-prefix-case)
|      |  +--ro source-prefix?        inet:ipv4-prefix
|      +--:(protocol-ip-case)
|         +--ro protocol-ips*
|            +--ro op?      numeric-operand
|            +--ro value?   uint8
+--ro path-id       path-id
+--ro attributes
   +--ro extended-communities*
      +--ro transitive?                              boolean
      +--ro (extended-community)?
         +--:(traffic-rate-extended-community-case)
         |  +--ro traffic-rate-extended-community
         |     +--ro informative-as?        bgp-t:short-as-number
         |     +--ro local-administrator?   netc:bandwidth
         +--:(traffic-action-extended-community-case)
         |  +--ro traffic-action-extended-community
         |     +--ro sample?            boolean
         |     +--ro terminal-action?   boolean
         +--:(redirect-extended-community-case)
         |  +--ro redirect-extended-community
         |     +--ro global-administrator?   bgp-t:short-as-number
```

```
                |     +--ro local-administrator?    binary
                +--:(traffic-marking-extended-community-case)
                |  +--ro traffic-marking-extended-community
                |     +--ro global-administrator?   dscp
                +--:(redirect-ipv4-extended-community-case)
                |  +--ro redirect-ipv4
                |     +--ro global-administrator?   inet:ipv4-address
                |     +--ro local-administrator?    uint16
                +--:(redirect-as4-extended-community-case)
                |  +--ro redirect-as4
                |     +--ro global-administrator?   inet:as-number
                |     +--ro local-administrator?    uint16
                +--:(redirect-ip-nh-extended-community-case)
                  +--ro redirect-ip-nh-extended-community
                     +--ro next-hop-address?   inet:ip-address
                     +--ro copy?               boolean
```

IPv6 Flow Specification Route

```
:(flowspec-ipv6-routes-case)
  +--ro flowspec-ipv6-routes
     +--ro flowspec-route* [route-key path-id]
        +--ro flowspec*
        |  +--ro (flowspec-type)?
        |     +--:(port-case)
        |     |  +--ro ports*
        |     |     +--ro op?      numeric-operand
        |     |     +--ro value?   uint16
        |     +--:(destination-port-case)
        |     |  +--ro destination-ports*
        |     |     +--ro op?      numeric-operand
        |     |     +--ro value?   uint16
        |     +--:(source-port-case)
        |     |  +--ro source-ports*
        |     |     +--ro op?      numeric-operand
        |     |     +--ro value?   uint16
        |     +--:(icmp-type-case)
        |     |  +--ro types*
        |     |     +--ro op?      numeric-operand
        |     |     +--ro value?   uint8
        |     +--:(icmp-code-case)
        |     |  +--ro codes*
        |     |     +--ro op?      numeric-operand
        |     |     +--ro value?   uint8
        |     +--:(tcp-flags-case)
        |     |  +--ro tcp-flags*
        |     |     +--ro op?      bitmask-operand
        |     |     +--ro value?   uint16
        |     +--:(packet-length-case)
        |     |  +--ro packet-lengths*
        |     |     +--ro op?      numeric-operand
        |     |     +--ro value?   uint16
        |     +--:(dscp-case)
        |     |  +--ro dscps*
        |     |     +--ro op?      numeric-operand
        |     |     +--ro value?   dscp
```

```
|      +--:(fragment-case)
|      |  +--ro fragments*
|      |     +--ro op?      bitmask-operand
|      |     +--ro value?   fragment
|      +--:(destination-ipv6-prefix-case)
|      |  +--ro destination-prefix?   inet:ipv6-prefix
|      +--:(source-ipv6-prefix-case)
|      |  +--ro source-prefix?        inet:ipv6-prefix
|      +--:(next-header-case)
|      |  +--ro next-headers*
|      |     +--ro op?      numeric-operand
|      |     +--ro value?   uint8
|      +--:(flow-label-case)
|         +--ro flow-label*
|            +--ro op?      numeric-operand
|            +--ro value?   uint32
+--ro path-id       path-id
+--ro attributes
   +--ro extended-communities*
      +--ro transitive?                              boolean
      +--ro (extended-community)?
         +--:(traffic-rate-extended-community-case)
         |  +--ro traffic-rate-extended-community
         |     +--ro informative-as?        bgp-t:short-as-number
         |     +--ro local-administrator?   netc:bandwidth
         +--:(traffic-action-extended-community-case)
         |  +--ro traffic-action-extended-community
         |     +--ro sample?            boolean
         |     +--ro terminal-action?   boolean
         +--:(redirect-extended-community-case)
         |  +--ro redirect-extended-community
         |     +--ro global-administrator?   bgp-t:short-as-number
         |     +--ro local-administrator?    binary
         +--:(traffic-marking-extended-community-case)
         |  +--ro traffic-marking-extended-community
         |     +--ro global-administrator?   dscp
         +--:(redirect-ipv6-extended-community-case)
         |  +--ro redirect-ipv6
         |     +--ro global-administrator?   inet:ipv6-address
         |     +--ro local-administrator?    uint16
         +--:(redirect-as4-extended-community-case)
         |  +--ro redirect-as4
         |     +--ro global-administrator?   inet:as-number
         |     +--ro local-administrator?    uint16
         +--:(redirect-ip-nh-extended-community-case)
            +--ro redirect-ip-nh-extended-community
               +--ro next-hop-address?   inet:ip-address
               +--ro copy?               boolean
```

Usage

The flowspec route represents rules and an action, defined as an extended community.

IPv4 Flow Specification

The IPv4 Flowspec table in an instance of the speaker's Loc-RIB can be verified via REST:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/loc-rib/tables/bgp-types:ipv4-address-family/bgp-flowspec:flowspec-subsequent-address-family/bgp-flowspec:flowspec-routes`

Method: `GET`

Response Body:

```
<flowspec-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-flowspec">
    <flowspec-route>
        <path-id>0</path-id>
        <route-key>all packets to 192.168.0.1/32 AND from 10.0.0.2/32 AND where IP␣
↪protocol equals to 17 or equals to 6 AND where port equals to 80 or equals to 8080␣
↪AND where destination port is greater than 8080 and is less than 8088 or equals to␣
↪3128 AND where source port is greater than 1024 </route-key>
        <attributes>
            <local-pref>
                <pref>100</pref>
            </local-pref>
            <origin>
                <value>igp</value>
            </origin>
            <as-path></as-path>
            <extended-communities>
                <transitive>true</transitive>
                <redirect-extended-community>
                    <local-administrator>AgMWLg==</local-administrator>
                    <global-administrator>258</global-administrator>
                </redirect-extended-community>
            </extended-communities>
        </attributes>
        <flowspec>
            <destination-prefix>192.168.0.1/32</destination-prefix>
        </flowspec>
        <flowspec>
            <source-prefix>10.0.0.2/32</source-prefix>
        </flowspec>
        <flowspec>
            <protocol-ips>
                <op>equals</op>
                <value>17</value>
            </protocol-ips>
            <protocol-ips>
                <op>equals end-of-list</op>
                <value>6</value>
            </protocol-ips>
        </flowspec>
        <flowspec>
            <ports>
                <op>equals</op>
                <value>80</value>
            </ports>
            <ports>
                <op>equals end-of-list</op>
                <value>8080</value>
```

```
            </ports>
        </flowspec>
        <flowspec>
            <destination-ports>
                <op>greater-than</op>
                <value>8080</value>
            </destination-ports>
            <destination-ports>
                <op>less-than and-bit</op>
                <value>8088</value>
            </destination-ports>
            <destination-ports>
                <op>equals end-of-list</op>
                <value>3128</value>
            </destination-ports>
        </flowspec>
        <flowspec>
            <source-ports>
                <op>end-of-list greater-than</op>
                <value>1024</value>
            </source-ports>
        </flowspec>
    </flowspec-route>
</flowspec-routes>
```

IPv6 Flows Specification

The IPv6 Flowspec table in an instance of the speaker's Loc-RIB can be verified via REST:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/loc-rib/tables/bgp-types:ipv6-address-family/bgp-flowspec:flowspec-subsequent-address-family/bgp-flowspec:flowspec-ipv6-routes`

Method: `GET`

Response Body:

```
<flowspec-ipv6-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-flowspec">
    <flowspec-route>
        <path-id>0</path-id>
        <route-key>all packets to 2001:db8:31::/64 AND from 2001:db8:30::/64 AND␣
↪where next header equals to 17 AND where DSCP equals to 50 AND where flow label␣
↪equals to 2013 </route-key>
        <attributes>
            <local-pref>
                <pref>100</pref>
            </local-pref>
            <origin>
                <value>igp</value>
            </origin>
            <as-path></as-path>
            <extended-communities>
                <transitive>true</transitive>
                <traffic-rate-extended-community>
                    <informative-as>0</informative-as>
                    <local-administrator>AAAAAA==</local-administrator>
                </traffic-rate-extended-community>
```

```
            </extended-communities>
        </attributes>
        <flowspec>
            <destination-prefix>2001:db8:31::/64</destination-prefix>
        </flowspec>
        <flowspec>
            <source-prefix>2001:db8:30::/64</source-prefix>
        </flowspec>
        <flowspec>
            <next-headers>
                <op>equals end-of-list</op>
                <value>17</value>
            </next-headers>
        </flowspec>
        <flowspec>
            <dscps>
                <op>equals end-of-list</op>
                <value>50</value>
            </dscps>
        </flowspec>
        <flowspec>
            <flow-label>
                <op>equals end-of-list</op>
                <value>2013</value>
            </flow-label>
        </flowspec>
    </flowspec-route>
</flowspec-ipv6-routes>
```

IPv4 L3VPN Flows Specification

The IPv4 L3VPN Flowspec table in an instance of the speaker's Loc-RIB can be verified via REST:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/loc-rib/tables/bgp-types:ipv4-address-family/bgp-flowspec:flowspec-l3vpn-subsequent-address-family/bgp-flowspec:flowspec-l3vpn-ipv4-routes`

Method: `GET`

Response Body:

```
<flowspec-l3vpn-ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-flowspec">
    <flowspec-l3vpn-route>
        <path-id>0</path-id>
        <route-key>[l3vpn with route-distinguisher 172.16.0.44:101] all packets from␣
↪10.0.0.3/32</route-key>
        <attributes>
            <local-pref>
                <pref>100</pref>
            </local-pref>
            <ipv4-next-hop>
                <global>5.6.7.8</global>
            </ipv4-next-hop>
            <origin>
                <value>igp</value>
            </origin>
            <as-path></as-path>
```

```
            <extended-communities>
                <transitive>true</transitive>
                <redirect-ip-nh-extended-community>
                    <copy>false</copy>
                    <next-hop-address>0.0.0.0</next-hop-address>
                </redirect-ip-nh-extended-community>
            </extended-communities>
        </attributes>
        <route-distinguisher>172.16.0.44:101</route-distinguisher>
        <flowspec>
            <source-prefix>10.0.0.3/32</source-prefix>
        </flowspec>
    </flowspec-l3vpn-route>
</flowspec-l3vpn-ipv4-routes>
```

Programming

IPv4 Flow Specification

This examples show how to originate and remove IPv4 fowspec route via programmable RIB. Make sure the *Application Peer* is configured first.

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/bgp-flowspec:flowspec-subsequent-address-family/bgp-flowspec:flowspec-routes`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<flowspec-route xmlns="urn:opendaylight:params:xml:ns:yang:bgp-flowspec">
    <route-key>flow1</route-key>
    <path-id>0</path-id>
    <flowspec>
        <destination-prefix>192.168.0.1/32</destination-prefix>
    </flowspec>
    <flowspec>
        <source-prefix>10.0.0.1/32</source-prefix>
    </flowspec>
    <flowspec>
        <protocol-ips>
            <op>equals end-of-list</op>
            <value>6</value>
        </protocol-ips>
    </flowspec>
    <flowspec>
        <ports>
            <op>equals end-of-list</op>
            <value>80</value>
        </ports>
    </flowspec>
    <flowspec>
        <destination-ports>
            <op>greater-than</op>
            <value>8080</value>
```

```
        </destination-ports>
        <destination-ports>
            <op>and-bit less-than end-of-list</op>
            <value>8088</value>
        </destination-ports>
    </flowspec>
    <flowspec>
        <source-ports>
            <op>greater-than end-of-list</op>
            <value>1024</value>
        </source-ports>
    </flowspec>
    <flowspec>
        <types>
            <op>equals end-of-list</op>
            <value>0</value>
        </types>
    </flowspec>
    <flowspec>
        <codes>
            <op>equals end-of-list</op>
            <value>0</value>
        </codes>
    </flowspec>
    <flowspec>
        <tcp-flags>
            <op>match end-of-list</op>
            <value>32</value>
        </tcp-flags>
    </flowspec>
    <flowspec>
        <packet-lengths>
            <op>greater-than</op>
            <value>400</value>
        </packet-lengths>
        <packet-lengths>
            <op>and-bit less-than end-of-list</op>
            <value>500</value>
        </packet-lengths>
    </flowspec>
    <flowspec>
        <dscps>
            <op>equals end-of-list</op>
            <value>20</value>
        </dscps>
    </flowspec>
    <flowspec>
        <fragments>
            <op>match end-of-list</op>
            <value>first</value>
        </fragments>
    </flowspec>
    <attributes>
        <origin>
            <value>igp</value>
        </origin>
        <as-path/>
        <local-pref>
```

```
            <pref>100</pref>
        </local-pref>
        <extended-communities>
            ....
        </extended-communities>
    </attributes>
</flowspec-route>
```

Extended Communities

- **Traffic Rate**

```
1 <extended-communities>
2     <transitive>true</transitive>
3     <traffic-rate-extended-community>
4         <informative-as>123</informative-as>
5         <local-administrator>AAAAAA==</local-administrator>
6     </traffic-rate-extended-community>
7 </extended-communities>
```

@line 5: A rate in bytes per second, *AAAAAA*== (0) means traffic discard.

- **Traffic Action**

```
<extended-communities>
    <transitive>true</transitive>
    <traffic-action-extended-community>
        <sample>true</sample>
        <terminal-action>false</terminal-action>
    </traffic-action-extended-community>
</extended-communities>
```

- **Redirect to VRF AS 2byte format**

```
<extended-communities>
    <transitive>true</transitive>
    <redirect-extended-community>
        <global-administrator>123</global-administrator>
        <local-administrator>AAAAew==</local-administrator>
    </redirect-extended-community>
</extended-communities>
```

- **Redirect to VRF IPv4 format**

```
<extended-communities>
    <transitive>true</transitive>
    <redirect-ipv4>
        <global-administrator>192.168.0.1</global-administrator>
        <local-administrator>12345</local-administrator>
    </redirect-ipv4>
</extended-communities>
```

- **Redirect to VRF AS 4byte format**

```
<extended-communities>
    <transitive>true</transitive>
    <redirect-as4>
```

```
        <global-administrator>64495</global-administrator>
        <local-administrator>12345</local-administrator>
    </redirect-as4>
</extended-communities>
```

- **Redirect to IP**

```
<extended-communities>
    <transitive>true</transitive>
    <redirect-ip-nh-extended-community>
        <copy>false</false>
    </redirect-ip-nh-extended-community>
</extended-communities>
```

- **Traffic Marking**

```
<extended-communities>
    <transitive>true</transitive>
    <traffic-marking-extended-community>
        <global-administrator>20</global-administrator>
    </traffic-marking-extended-community>
</extended-communities>
```

To remove the route added above, following request can be used:

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/bgp-flowspec:flowspec-subsequent-address-family/bgp-flowspec:flowspec-routes/bgp-flowspec:flowspec-route/flow1/0`

Method: `DELETE`

IPv4 L3VPN Flow Specification

This examples show how to originate and remove IPv4 L3VPN fowspec route via programmable RIB.

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/bgp-flowspec:flowspec-l3vpn-subsequent-address-family/bgp-flowspec:flowspec-l3vpn-ipv4-routes`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<flowspec-l3vpn-route xmlns="urn:opendaylight:params:xml:ns:yang:bgp-flowspec">
    <path-id>0</path-id>
    <route-key>flow-l3vpn</route-key>
    <route-distinguisher>172.16.0.44:101</route-distinguisher>
    <flowspec>
        <source-prefix>10.0.0.3/32</source-prefix>
    </flowspec>
    <attributes>
        <local-pref>
            <pref>100</pref>
        </local-pref>
        <origin>
```

```
            <value>igp</value>
        </origin>
        <as-path></as-path>
           <extended-communities>
               <transitive>true</transitive>
               <redirect-ipv4>
                   <global-administrator>172.16.0.44</global-administrator>
                   <local-administrator>102</local-administrator>
               </redirect-ipv4>
           </extended-communities>
    </attributes>
</flowspec-l3vpn-route>
```

To remove the route added above, following request can be used:

URL: /restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/bgp-flowspec:flowspec-l3vpn-subsequent-address-family/bgp-flowspec:flowspec-l3vpn-ipv4-routes/flowspec-l3vpn-route/flow-l3vpn/0

Method: DELETE

IPv6 Flow Specification

This examples show how to originate and remove IPv6 fowspec route via programmable RIB.

URL: /restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv6-address-family/bgp-flowspec:flowspec-subsequent-address-family/bgp-flowspec:flowspec-ipv6-routes

Method: POST

Content-Type: application/xml

Request Body:

```
<flowspec-route xmlns="urn:opendaylight:params:xml:ns:yang:bgp-flowspec">
    <route-key>flow-v6</route-key>
    <path-id>0</path-id>
    <flowspec>
        <destination-prefix>2001:db8:30::3/128</destination-prefix>
    </flowspec>
    <flowspec>
        <source-prefix>2001:db8:31::3/128</source-prefix>
     </flowspec>
    <flowspec>
        <flow-label>
            <op>equals end-of-list</op>
            <value>1</value>
        </flow-label>
    </flowspec>
    <attributes>
        <extended-communities>
            <transitive>true</transitive>
            <redirect-ipv6>
                <global-administrator>2001:db8:1::6</global-administrator>
                <local-administrator>12345</local-administrator>
            </redirect-ipv6>
```

```
        </extended-communities>
        <origin>
            <value>igp</value>
        </origin>
        <as-path/>
        <local-pref>
            <pref>100</pref>
        </local-pref>
    </attributes>
</flowspec-route>
```

To remove the route added above, following request can be used:

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv6-address-family/bgp-flowspec:flowspec-subsequent-address-family/bgp-flowspec:flowspec-ipv6-routes/bgp-flowspec:flowspec-route/flow-v6/0`

Method: `DELETE`

References

- Dissemination of Flow Specification Rules
- Dissemination of Flow Specification Rules for IPv6
- BGP Flow-Spec Extended Community for Traffic Redirect to IP Next Hop
- Clarification of the Flowspec Redirect Extended Community
- Revised Validation Procedure for BGP Flow Specifications

EVPN Family

The BGP MPLS-Based Ethernet VPN (BGP EVPN) Multiprotocol extension can be used to distribute Ethernet L2VPN service related routes in order to support a concept of MAC routing. A major use-case for BGP EVPN is data-center interconnection (DCI), where advantage of BGP EVPN are MAC/IP address advertising across MPLS network, Multihoming functionality including Fast Convergence, Split Horizon and Aliasing support, VM (MAC) Mobility, support Multicast and Broadcast traffic. In addition to MPLS, IP tunnelling encapsulation techniques like VXLAN, NVGRE, MPLSoGRE and others can be used for packet transportation. Also, Provider Backbone Bridging (PBB) can be combined with EVPN in order to reduce a number of MAC Advertisement routes.

Contents

Configuration

This section shows a way to enable EVPN family in BGP speaker and peer configuration.

BGP Speaker

To enable EVPN support in BGP plugin, first configure BGP speaker instance:

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<protocol xmlns="http://openconfig.net/yang/network-instance">
    <name>bgp-example</name>
    <identifier xmlns:x="http://openconfig.net/yang/policy-types">x:BGP</identifier>
    <bgp xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
        <global>
            <config>
                <router-id>192.0.2.2</router-id>
                <as>65000</as>
            </config>
            <afi-safis>
                <afi-safi>
                    <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">
↪x:L2VPN-EVPN</afi-safi-name>
                </afi-safi>
            </afi-safis>
        </global>
    </bgp>
</protocol>
```

BGP Peer

Here is an example for BGP peer configuration with enabled EVPN family.

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<neighbor xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <neighbor-address>192.0.2.1</neighbor-address>
    <afi-safis>
        <afi-safi>
            <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">x:L2VPN-EVPN
↪</afi-safi-name>
        </afi-safi>
```

```
    </afi-safis>
</neighbor>
```

EVPN Route API

Following tree illustrate the BGP EVPN route structure.

```
:(evpn-routes-case)
   +--ro evpn-routes
      +--ro evpn-route* [route-key]
         +--ro route-key                      string
         +--ro (evpn-choice)
         |  +--:(ethernet-a-d-route-case)
         |  |  +--ro ethernet-a-d-route
         |  |     +--ro (esi)
         |  |     |  +--:(arbitrary-case)
         |  |     |  |  +--ro arbitrary
         |  |     |  |     +--ro arbitrary    binary
         |  |     |  +--:(lacp-auto-generated-case)
         |  |     |  |  +--ro lacp-auto-generated
         |  |     |  |     +--ro ce-lacp-mac-address    yang:mac-address
         |  |     |  |     +--ro ce-lacp-port-key       uint16
         |  |     |  +--:(lan-auto-generated-case)
         |  |     |  |  +--ro lan-auto-generated
         |  |     |  |     +--ro root-bridge-mac-address    yang:mac-address
         |  |     |  |     +--ro root-bridge-priority       uint16
         |  |     |  +--:(mac-auto-generated-case)
         |  |     |  |  +--ro mac-auto-generated
         |  |     |  |     +--ro system-mac-address     yang:mac-address
         |  |     |  |     +--ro local-discriminator    uint24
         |  |     |  +--:(router-id-generated-case)
         |  |     |  |  +--ro router-id-generated
         |  |     |  |     +--ro router-id              inet:ipv4-address
         |  |     |  |     +--ro local-discriminator    uint32
         |  |     |  +--:(as-generated-case)
         |  |     |     +--ro as-generated
         |  |     |        +--ro as                     inet:as-number
         |  |     |        +--ro local-discriminator    uint32
         |  |     +--ro ethernet-tag-id
         |  |     |  +--ro vlan-id    uint32
         |  |     +--ro mpls-label             netc:mpls-label
         |  +--:(mac-ip-adv-route-case)
         |  |  +--ro mac-ip-adv-route
         |  |     +--ro (esi)
         |  |     |  +--:(arbitrary-case)
         |  |     |  |  +--ro arbitrary
         |  |     |  |     +--ro arbitrary    binary
         |  |     |  +--:(lacp-auto-generated-case)
         |  |     |  |  +--ro lacp-auto-generated
         |  |     |  |     +--ro ce-lacp-mac-address    yang:mac-address
         |  |     |  |     +--ro ce-lacp-port-key       uint16
         |  |     |  +--:(lan-auto-generated-case)
         |  |     |  |  +--ro lan-auto-generated
         |  |     |  |     +--ro root-bridge-mac-address    yang:mac-address
         |  |     |  |     +--ro root-bridge-priority       uint16
         |  |     |  +--:(mac-auto-generated-case)
```

```
|  |     |  |  +--ro mac-auto-generated
|  |     |  |     +--ro system-mac-address     yang:mac-address
|  |     |  |     +--ro local-discriminator    uint24
|  |     |  +--:(router-id-generated-case)
|  |     |  |  +--ro router-id-generated
|  |     |  |     +--ro router-id              inet:ipv4-address
|  |     |  |     +--ro local-discriminator    uint32
|  |     |  +--:(as-generated-case)
|  |     |     +--ro as-generated
|  |     |        +--ro as                     inet:as-number
|  |     |        +--ro local-discriminator    uint32
|  |     +--ro ethernet-tag-id
|  |     |  +--ro vlan-id    uint32
|  |     +--ro mac-address            yang:mac-address
|  |     +--ro ip-address?            inet:ip-address
|  |     +--ro mpls-label1            netc:mpls-label
|  |     +--ro mpls-label2?           netc:mpls-label
|  +--:(inc-multi-ethernet-tag-res-case)
|  |  +--ro inc-multi-ethernet-tag-res
|  |     +--ro ethernet-tag-id
|  |     |  +--ro vlan-id    uint32
|  |     +--ro orig-route-ip?     inet:ip-address
|  +--:(es-route-case)
|     +--ro es-route
|        +--ro (esi)
|        |  +--:(arbitrary-case)
|        |  |  +--ro arbitrary
|        |  |     +--ro arbitrary    binary
|        |  +--:(lacp-auto-generated-case)
|        |  |  +--ro lacp-auto-generated
|        |  |     +--ro ce-lacp-mac-address    yang:mac-address
|        |  |     +--ro ce-lacp-port-key       uint16
|        |  +--:(lan-auto-generated-case)
|        |  |  +--ro lan-auto-generated
|        |  |     +--ro root-bridge-mac-address    yang:mac-address
|        |  |     +--ro root-bridge-priority       uint16
|        |  +--:(mac-auto-generated-case)
|        |  |  +--ro mac-auto-generated
|        |  |     +--ro system-mac-address     yang:mac-address
|        |  |     +--ro local-discriminator    uint24
|        |  +--:(router-id-generated-case)
|        |  |  +--ro router-id-generated
|        |  |     +--ro router-id              inet:ipv4-address
|        |  |     +--ro local-discriminator    uint32
|        |  +--:(as-generated-case)
|        |     +--ro as-generated
|        |        +--ro as                     inet:as-number
|        |        +--ro local-discriminator    uint32
|        +--ro orig-route-ip          inet:ip-address
+--ro route-distinguisher           bgp-t:route-distinguisher
+--ro attributes
   +--ro extended-communities*
   |  +--ro transitive?                              boolean
   |  +--ro (extended-community)?
   |     +--:(encapsulation-case)
   |     |  +--ro encapsulation-extended-community
   |     |     +--ro tunnel-type    encapsulation-tunnel-type
   |     +--:(esi-label-extended-community-case)
```

```
|     |  +--ro esi-label-extended-community
|     |     +--ro single-active-mode?   boolean
|     |     +--ro esi-label             netc:mpls-label
|     +--:(es-import-route-extended-community-case)
|     |  +--ro es-import-route-extended-community
|     |     +--ro es-import    yang:mac-address
|     +--:(mac-mobility-extended-community-case)
|     |  +--ro mac-mobility-extended-community
|     |     +--ro static?       boolean
|     |     +--ro seq-number    uint32
|     +--:(default-gateway-extended-community-case)
|     |  +--ro default-gateway-extended-community!
|     +--:(layer-2-attributes-extended-community-case)
|        +--ro layer-2-attributes-extended-community
|           +--ro primary-pe?     boolean
|           +--ro backup-pe?      boolean
|           +--ro control-word?   boolean
|           +--ro l2-mtu          uint16
+--ro pmsi-tunnel!
   +--ro leaf-information-required    boolean
   +--ro mpls-label?                  netc:mpls-label
   +--ro (tunnel-identifier)?
      +--:(rsvp-te-p2mp-lsp)
      |  +--ro rsvp-te-p2mp-lps
      |     +--ro p2mp-id               uint32
      |     +--ro tunnel-id             uint16
      |     +--ro extended-tunnel-id    inet:ip-address
      +--:(mldp-p2mp-lsp)
      |  +--ro mldp-p2mp-lsp
      |     +--ro address-family       identityref
      |     +--ro root-node-address    inet:ip-address
      |     +--ro opaque-value*
      |        +--ro opaque-type             uint8
      |        +--ro opaque-extended-type?   uint16
      |        +--ro opaque                  yang:hex-string
      +--:(pim-ssm-tree)
      |  +--ro pim-ssm-tree
      |     +--ro p-address            inet:ip-address
      |     +--ro p-multicast-group    inet:ip-address
      +--:(pim-sm-tree)
      |  +--ro pim-sm-tree
      |     +--ro p-address            inet:ip-address
      |     +--ro p-multicast-group    inet:ip-address
      +--:(bidir-pim-tree)
      |  +--ro bidir-pim-tree
      |     +--ro p-address            inet:ip-address
      |     +--ro p-multicast-group    inet:ip-address
      +--:(ingress-replication)
      |  +--ro ingress-replication
      |     +--ro receiving-endpoint-address?   inet:ip-address
      +--:(mldp-mp2mp-lsp)
         +--ro mldp-mp2mp-lsp
            +--ro opaque-type             uint8
            +--ro opaque-extended-type?   uint16
            +--ro opaque
      ...
```

Usage

The L2VPN EVPN table in an instance of the speaker's Loc-RIB can be verified via REST:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/loc-rib/tables/odl-bgp-evpn:l2vpn-address-family/odl-bgp-evpn:evpn-subsequent-address-family/evpn-routes`

Method: `GET`

Response Body:

```
<evpn-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-evpn">
   <evpn-route>
      <route-key>AxEAAcCoZAED6AAAAQAgwKhkAQ==</route-key>
      <route-distinguisher>192.168.100.1:1000</route-distinguisher>
      <inc-multi-ethernet-tag-res>
         <ethernet-tag-id>
            <vlan-id>256</vlan-id>
         </ethernet-tag-id>
         <orig-route-ip>192.168.100.1</orig-route-ip>
      </inc-multi-ethernet-tag-res>
      <attributes>
         <ipv4-next-hop>
            <global>172.23.29.104</global>
         </ipv4-next-hop>
         <as-path/>
         <origin>
            <value>igp</value>
         </origin>
         <extended-communities>
            <extended-communities>
                <transitive>true</transitive>
                <route-target-extended-community>
                    <global-administrator>65504</global-administrator>
                    <local-administrator>AAAD6A==</local-administrator>
                </route-target-extended-community>
            </extended-communities>
         </extended-communities>
         <pmsi-tunnel>
             <leaf-information-required>true</leaf-information-required>
             <mpls-label>20024</mpls-label>
             <ingress-replication>
                 <receiving-endpoint-address>192.168.100.1</receiving-endpoint-
↪address>
             </ingress-replication>
         </pmsi-tunnel>
      </attributes>
   </evpn-route>
</evpn-routes>
```

Programming

This examples show how to originate and remove EVPN routes via programmable RIB. There are four different types of EVPN routes, and several extended communities. Routes can be used for variety of use-cases supported by BGP/MPLS EVPN, PBB EVPN and NVO EVPN. Make sure the *Application Peer* is configured first.

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/odl-bgp-evpn:l2vpn-address-family/odl-bgp-evpn:evpn-subsequent-address-family/odl-bgp-evpn:evpn-routes`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
1  <evpn-route xmlns="urn:opendaylight:params:xml:ns:yang:bgp-evpn">
2      <route-key>evpn</route-key>
3      <route-distinguisher>172.12.123.3:200</route-distinguisher>
4      ....
5      <attributes>
6          <ipv4-next-hop>
7              <global>199.20.166.41</global>
8          </ipv4-next-hop>
9          <as-path/>
10         <origin>
11             <value>igp</value>
12         </origin>
13         <extended-communities>
14         ....
15         </extended-communities>
16     </attributes>
17 </evpn-route>
```

@line 3: Route Distinguisher (RD) - set to RD of the MAC-VRF advertising the NLRI, recommended format *<IP>:<VLAN_ID>*

@line 4: One of the EVPN route must be set here.

@line 14: In some cases, specific extended community presence is required. The route may carry one or more Route Target attributes.

EVPN Routes:

- **Ethernet AD per ESI**

```
<ethernet-a-d-route>
    <mpls-label>0</mpls-label>
    <ethernet-tag-id>
        <vlan-id>4294967295</vlan-id>
    </ethernet-tag-id>
    <arbitrary>
        <arbitrary>AAAAAAAAAAAA</arbitrary>
    </arbitrary>
</ethernet-a-d-route>
```

- **Ethernet AD per EVI**

```
<ethernet-a-d-route>
    <mpls-label>24001</mpls-label>
    <ethernet-tag-id>
        <vlan-id>2200</vlan-id>
    </ethernet-tag-id>
    <arbitrary>
        <arbitrary>AAAAAAAAAAAA</arbitrary>
```

```
    </arbitrary>
</ethernet-a-d-route>
```

- **MAC/IP Advertisement**

```
<mac-ip-adv-route>
    <arbitrary>
        <arbitrary>AAAAAAAAAAAA</arbitrary>
    </arbitrary>
    <ethernet-tag-id>
        <vlan-id>2100</vlan-id>
    </ethernet-tag-id>
    <mac-address>f2:0c:dd:80:9f:f7</mac-address>
    <ip-address>10.0.1.12</ip-address>
    <mpls-label1>299776</mpls-label1>
</mac-ip-adv-route>
```

- **Inclusive Multicast Ethernet Tag**

```
<inc-multi-ethernet-tag-res>
    <ethernet-tag-id>
        <vlan-id>2100</vlan-id>
    </ethernet-tag-id>
    <orig-route-ip>43.43.43.43</orig-route-ip>
</inc-multi-ethernet-tag-res>
```

- **Ethernet Segment**

```
<es-route>
    <orig-route-ip>43.43.43.43</orig-route-ip>
    <arbitrary>
        <arbitrary>AAAAAAAAAAAA</arbitrary>
    </arbitrary>
</es-route>
```

EVPN Ethernet Segment Identifier (ESI):

- **Type 0** Indicates an arbitrary 9-octet ESI.

```
<arbitrary>
    <arbitrary>AAAAAAAAAAAA</arbitrary>
</arbitrary>
```

- **Type 1** IEEE 802.1AX LACP is used.

```
<lacp-auto-generated>
    <ce-lacp-mac-address>f2:0c:dd:80:9f:f7</ce-lacp-mac-address>
    <ce-lacp-port-key>22</ce-lacp-port-key>
</lacp-auto-generated>
```

- **Type 2** Indirectly connected hosts via a bridged LAN.

```
<lan-auto-generated>
    <root-bridge-mac-address>f2:0c:dd:80:9f:f7</root-bridge-mac-address>
    <root-bridge-priority>20</root-bridge-priority>
</lan-auto-generated>
```

- **Type 3** MAC-based ESI.

```
<mac-auto-generated>
    <system-mac-address>f2:0c:dd:80:9f:f7</system-mac-address>
    <local-discriminator>2000</local-discriminator>
</mac-auto-generated>
```

- **Type 4** Router-ID ESI

```
<router-id-generated>
    <router-id>43.43.43.43</router-id>
    <local-discriminator>2000</local-discriminator>
</router-id-generated>
```

- **Type 5** AS-based ESI

```
<as-generated>
    <as>16843009</as>
    <local-discriminator>2000</local-discriminator>
</as-generated>
```

Extended Communities:

- **ESI Label Extended Community**

```
<extended-communities>
    <transitive>true</transitive>
    <esi-label-extended-community>
        <single-active-mode>false</single-active-mode>
        <esi-label>24001</esi-label>
    </esi-label-extended-community >
</extended-communities>
```

- **ES-Import Route Target**

```
<extended-communities>
    <transitive>true</transitive>
    <es-import-route-extended-community>
        <es-import>f2:0c:dd:80:9f:f7</es-import>
    </es-import-route-extended-community>
</extended-communities>
```

- **MAC Mobility Extended Community**

```
<extended-communities>
    <transitive>true</transitive>
    <mac-mobility-extended-community>
        <static>true</static>
        <seq-number>200</seq-number>
    </mac-mobility-extended-community>
</extended-communities>
```

- **Default Gateway Extended Community**

```
<extended-communities>
    <transitive>true</transitive>
    <default-gateway-extended-community>
    </default-gateway-extended-community>
</extended-communities>
```

- **EVPN Layer 2 attributes extended community**

```
<extended-communities>
    <transitive>false</transitive>
    <layer-2-attributes-extended-community>
        <primary-pe>true</primary-pe>
        <backup-pe>true</backup-pe>
        <control-word >true</control-word>
        <l2-mtu>200</l2-mtu>
    </layer-2-attributes-extended-community>
</extended-communities>
```

- **BGP Encapsulation extended community**

```
1 <extended-communities>
2     <transitive>false</transitive>
3     <encapsulation-extended-community>
4         <tunnel-type>vxlan</tunnel-type>
5     </encapsulation-extended-community>
6 </extended-communities>
```

@line 4: full list of tunnel types

- **P-Multicast Service Interface Tunnel (PMSI) attribute**

```
<pmsi-tunnel>
    <leaf-information-required>true</leaf-information-required>
    <mpls-label>20024</mpls-label>
    <ingress-replication>
        <receiving-endpoint-address>172.12.123.3</receiving-endpoint-address>
    </ingress-replication>
</pmsi-tunnel>
```

To remove the route added above, following request can be used:

URL: `/restconf/config/bgp-rib:application-rib/10.25.1.9/tables/bgp-types:ipv4-address-family/odl-bgp-evpn:l2vpn-address-family/odl-bgp-evpn:evpn-subsequent-address-family/odl-bgp-evpn:evpn-routes/evpn-route/evpn`

Method: `DELETE`

Table 1.4: EVPN Routes Usage.

EVN Route Type	Extended Communities	Usage
Ethernet Auto-discovery	ESI Label, BGP EncapsulationEVPN Layer 2 attributes	Fast Convergence, Split Horizon, Aliasing
MAC/IP Advertisement	BGP Encapsulation, MAC Mobility, Default Gateway	MAC address reachability
Inclusive Multicast Ethernet Tag	PMSI Tunnel, BGP Encapsulation	Handling of Multi-destination traffic
Ethernet Segment	BGP Encapsulation, ES-Import Route Target	Designated Forwarder Election

References

- BGP MPLS-Based Ethernet VPN
- Provider Backbone Bridging Combined with Ethernet VPN
- VPWS support in EVPN
- A Network Virtualization Overlay Solution using EVPN
- Interconnect Solution for EVPN Overlay networks
- Usage and applicability of BGP MPLS based Ethernet VPN

Additional Path

The ADD-PATH capability allows to advertise multiple paths for the same address prefix. It can help with optimal routing and routing convergence in a network by providing potential alternate or backup paths.

Contents

Configuration

This section shows a way to enable ADD-PATH capability in BGP speaker and peer configuration.

Note: The capability is applicable for IP Unicast, IP Labeled Unicast and Flow Specification address families.

BGP Speaker

To enable ADD-PATH capability in BGP plugin, first configure BGP speaker instance:

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<protocol xmlns="http://openconfig.net/yang/network-instance">
    <name>bgp-example</name>
    <identifier xmlns:x="http://openconfig.net/yang/policy-types">x:BGP</identifier>
    <bgp xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
```

```
        <global>
            <config>
                <router-id>192.0.2.2</router-id>
                <as>65000</as>
            </config>
            <afi-safis>
                <afi-safi>
                    <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">
↪x:IPV4-UNICAST</afi-safi-name>
                    <receive>true</receive>
                    <send-max>2</send-max>
                </afi-safi>
            </afi-safis>
        </global>
    </bgp>
</protocol>
```

@line 14: Defines path selection strategy: *send-max* > 1 -> Advertise N Paths or *send-max* = 0 -> Advertise All Paths

Here is an example for update a specific family with enable ADD-PATH capability

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/global/afi-safis/afi-safi/openconfig-bgp-types:IPV4%2DUNICAST`

Method: `PUT`

Content-Type: `application/xml`

Request Body:

```
<afi-safi xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
   <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">x:IPV4-UNICAST</afi-
↪safi-name>
   <receive>true</receive>
   <send-max>0</send-max>
</afi-safi>
```

BGP Peer

Here is an example for BGP peer configuration with enabled ADD-PATH capability.

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<neighbor xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <neighbor-address>192.0.2.1</neighbor-address>
    <afi-safis>
        <afi-safi>
            <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">x:IPV4-
↪LABELLED-UNICAST</afi-safi-name>
```

```
        </afi-safi>
        <afi-safi>
            <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">x:IPV4-
↪UNICAST</afi-safi-name>
            <receive>true</receive>
            <send-max>0</send-max>
        </afi-safi>
    </afi-safis>
</neighbor>
```

Note: The path selection strategy is not configurable on per peer basis. The send-max presence indicates a willingness to send ADD-PATH NLRIs to the neighbor.

Here is an example for update specific family BGP peer configuration with enabled ADD-PATH capability.

URL: `/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors/neighbor/192.0.2.1/afi-safis/afi-safi/openconfig-bgp-types:IPV4%2DUNICAST`

Method: `PUT`

Content-Type: `application/xml`

Request Body:

```
<afi-safi xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
   <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">x:IPV4-UNICAST</afi-
↪safi-name>
   <receive>true</receive>
   <send-max>0</send-max>
</afi-safi>
```

Usage

The IPv4 Unicast table with enabled ADD-PATH capability in an instance of the speaker's Loc-RIB can be verified via REST:

URL: `/restconf/operational/bgp-rib:bgp-rib/rib/bgp-example/loc-rib/tables/bgp-types:ipv4-address-family/bgp-types:unicast-subsequent-address-family/ipv4-routes`

Method: `GET`

Response Body:

```
<ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet">
    <ipv4-route>
        <path-id>1</path-id>
        <prefix>193.0.2.1/32</prefix>
        <attributes>
            <as-path></as-path>
            <origin>
                <value>igp</value>
            </origin>
            <local-pref>
```

```
                <pref>100</pref>
            </local-pref>
            <ipv4-next-hop>
                <global>10.0.0.1</global>
            </ipv4-next-hop>
        </attributes>
    </ipv4-route>
    <ipv4-route>
        <path-id>2</path-id>
        <prefix>193.0.2.1/32</prefix>
        <attributes>
            <as-path></as-path>
            <origin>
                <value>igp</value>
            </origin>
            <local-pref>
                <pref>100</pref>
            </local-pref>
            <ipv4-next-hop>
                <global>10.0.0.2</global>
            </ipv4-next-hop>
        </attributes>
    </ipv4-route>
</ipv4-routes>
```

@line 3: The routes with the same destination are distinguished by *path-id* attribute.

References

- Advertisement of Multiple Paths in BGP
- Best Practices for Advertisement of Multiple Paths in IBGP

Route Refresh

The Route Refresh Capability allows to dynamically request a re-advertisement of the Adj-RIB-Out from a BGP peer. This is useful when the inbound routing policy for a peer changes and all prefixes from a peer must be reexamined against a new policy.

Contents

Configuration

The capability is enabled by default, no additional configuration is required.

Usage

To send a Route Refresh request from OpenDaylight BGP speaker instance to its neighbor, invoke RPC:

URL: `/restconf/operations/bgp-peer-rpc:route-refresh-request`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<input xmlns="urn:opendaylight:params:xml:ns:yang:bgp-peer-rpc">
    <afi xmlns:types="urn:opendaylight:params:xml:ns:yang:bgp-types">types:ipv4-
↪address-family</afi>
    <safi xmlns:types="urn:opendaylight:params:xml:ns:yang:bgp-types">types:unicast-
↪subsequent-address-family</safi>
    <peer-ref xmlns:rib="urn:opendaylight:params:xml:ns:yang:bgp-rib">/rib:bgp-rib/
↪rib:rib[rib:id="bgp-example"]/rib:peer[rib:peer-id="bgp://10.25.1.9"]</peer-ref>
</input>
```

References

- Route Refresh Capability for BGP-4

Operational State

The OpenDaylight BGP implementation provides a set of APIs (described below), that give its operational state refreshed periodically, by default every 5 seconds. The following APIs describe what is available starting with how to change the default refresh rate.

Contents

Operational State Configuration

URL: `/restconf/config/bgp-state-config:bgp-state-config`

Method: `PUT`

Content-Type: `application/xml`

Request Body:

```
<bgp-state-config xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
    <config-name xmlns="urn:opendaylight:params:xml:ns:yang:bgp-state-config">
↪operationalState</config-name>
    <timer xmlns="urn:opendaylight:params:xml:ns:yang:bgp-state-config">1</timer>
</bgp-state-config>
```

@line 3: Time in seconds between operational state update.

BGP RIB Operational State

URL: `/restconf/operational/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/global/state`

Method: `GET`

Content-Type: `application/xml`

Response Body:

```
<state xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <as>65000</as>
    <router-id>192.0.2.2</router-id>
    <total-paths>0</total-paths>
    <total-prefixes>0</total-prefixes>
</state>
```

@line 2: AS number of the remote peer.

@line 3: The unique protocol instance identifier.

@line 4: Total number of Paths installed on RIB (Loc-RIB)

@line 5: Total number of Prefixes installed on RIB (Loc-RIB)

BGP RIB Families Operational State

URL: `/restconf/operational/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/global/afi-safis`

Method: `GET`

Content-Type: `application/xml`

Response Body:

```
<afi-safis xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <afi-safi>
        <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">x:IPV4-UNICAST</
↪afi-safi-name>
        <state>
            <total-paths>0</total-paths>
            <total-prefixes>0</total-prefixes>
        </state>
    </afi-safi>
    <afi-safi>
        <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">x:IPV6-UNICAST</
↪afi-safi-name>
        <state>
            <total-paths>0</total-paths>
            <total-prefixes>0</total-prefixes>
        </state>
    </afi-safi>
    ....
</afi-safis>
```

@line 3: Family Identifier.

@line 5: Total number of Paths installed on RIB (Loc-RIB) per specific family.

@line 6: Total number of Prefixes installed on RIB (Loc-RIB) per specific family.

BGP Neighbors Operational State

URL: `/restconf/operational/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors`

Method: `GET`

Content-Type: `application/xml`

Response Body:

```
<neighbors xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <neighbor>
        <neighbor-address>192.0.2.1</neighbor-address>
        .....
    </neighbor>
    <neighbor>
        <neighbor-address>192.0.2.2</neighbor-address>
        .....
    </neighbor>
</neighbors>
```

@line 3: IP address of the remote BGP peer. Also serves as an unique identifier of a neighbor in a list of neighbors.

BGP Neighbor Operational State

Note: Supported Capabilities only provided when session has been established.

URL: `/restconf/operational/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbor/127.0.0.2/state`

Method: `GET`

Content-Type: `application/xml`

Response Body:

```
<state xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <session-state>ESTABLISHED</session-state>
    <supported-capabilities xmlns:x="http://openconfig.net/yang/bgp-types">x:ASN32</supported-capabilities>
    <supported-capabilities xmlns:x="http://openconfig.net/yang/bgp-types">x:MPBGP</supported-capabilities>
    <messages>
        <sent>
            <UPDATE>0</UPDATE>
            <NOTIFICATION>0</NOTIFICATION>
        </sent>
        <received>
            <UPDATE>4</UPDATE>
            <NOTIFICATION>0</NOTIFICATION>
        </received>
    </messages>
</state>
```

@line 2: Session status

@line 3-4: BGP capabilities supported (ASN32 / MPBGP / ROUTE_REFRESH / GRACEFUL_RESTART / ADD_PATHS)

@line 7: Total count of Update Messages sent

@line 8: Total count of Notification Messages sent

@line 11: Total count of Update Messages received

@line 12: Total count of Notification Messages received

BGP Neighbor Families Operational State

Note: Graceful Restart not supported yet. Planned for Carbon.

URL: `/restconf/operational/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors/neighbor/192.0.2.1/afi-safis`

Method: `GET`

Content-Type: `application/xml`

Response Body:

```
<afi-safis xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
     <afi-safi>
         <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">x:IPV4-UNICAST</afi-safi-name>
```

```
4           <state>
5               <active>false</active>
6           </state>
7           <graceful-restart>
8               <state>
9                   <received>false</received>
10                  <advertised>false</advertised>
11              </state>
12          </graceful-restart>
13      </afi-safi>
14      <afi-safi>
15          <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">x:IPV6-UNICAST
↪</afi-safi-name>
16          <state>
17              <active>false</active>
18          </state>
19          <graceful-restart>
20              <state>
21                  <received>false</received>
22                  <advertised>false</advertised>
23              </state>
24          </graceful-restart>
25      </afi-safi>
26  </afi-safis>
```

@line 3: Family Identifier.

@line 5: True if family is advertized by peer.

@line 7: Graceful Restart Operational State per specific family.

@line 9: True if the peer supports graceful restart.

@line 10: True if we support graceful restart.

BGP Neighbor Family Operational State

Note: Prefixes state is only provided once session is established.

Note: Graceful Restart not supported yet. Planned to be implemented in Carbon.

URL: `/restconf/operational/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors/neighbor/192.0.2.1/afi-safis/afi-safi/openconfig-bgp-types:IPV4%2DUNICAST`

Method: `GET`

Content-Type: `application/xml`

Response Body:

```
1  <afi-safi xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
2      <afi-safi-name xmlns:x="http://openconfig.net/yang/bgp-types">x:IPV4-UNICAST</afi-
↪safi-name>
```

```
    <state>
        <active>true</active>
        <prefixes>
            <installed>3</installed>
            <sent>0</sent>
            <received>3</received>
        </prefixes>
    </state>
    <graceful-restart>
        <state>
            <received>false</received>
            <advertised>false</advertised>
        </state>
    </graceful-restart>
</afi-safi>
```

@line 2: Family Identifier.

@line 4: True if family is advertized to and by peer.

@line 6: Total count of prefixes advertized by peer and installed (effective-rib-in).

@line 7: Total count of prefixes advertized to peer (adj-rib-out).

@line 8: Total count of prefixes advertized by peer (adj-rib-in).

BGP Neighbor Timers Operational State

Note: State is only provided once session is established.

URL: `/restconf/operational/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors/neighbor/192.0.2.1/timers`

Method: `GET`

Content-Type: `application/xml`

Response Body:

```
<timers xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <state>
        <negotiated-hold-time>180</negotiated-hold-time>
        <uptime>1580676</uptime>
    </state>
</timers>
```

@line 3: The negotiated hold-time for the BGP session in seconds.

@line 4: Session duration since establishment in milliseconds.

BGP Neighbor Transport Operational State

Note: State is only provided once session is established.

URL: `/restconf/operational/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors/neighbor/192.0.2.1/transport`

Method: `GET`

Content-Type: `application/xml`

Response Body:

```
<transport xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <state>
        <remote-address>127.0.0.2</remote-address>
        <remote-port>44718</remote-port>
        <local-port>1790</local-port>
    </state>
</transport>
```

@line 3: IP address of the remote BGP peer.

@line 4: Port of the remote BGP peer.

@line 5: Local port.

BGP Neighbor Error Handling Operational State

Note: State is only provided once session is established.

Note: Error handling not supported yet. Planned for Carbon.

URL: `/restconf/operational/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors/neighbor/192.0.2.1/error-handling`

Method: `GET`

Content-Type: `application/xml`

Response Body:

```
<error-handling xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions">
    <state>
        <erroneous-update-messages>0</erroneous-update-messages>
    </state>
</error-handling>
```

@line 3: The number of BGP UPDATE messages for which the treat-as-withdraw mechanism has been applied based on erroneous message contents

BGP Neighbor Graceful Restart Operational State

Note: Graceful Restart not supported yet. Planned for Carbon.

URL: `/restconf/operational/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/bgp/neighbors/neighbor/192.0.2.1/graceful-restart`

Method: `GET`

Content-Type: `application/xml`

Response Body:

```
<graceful-restart xmlns="urn:opendaylight:params:xml:ns:yang:bgp:openconfig-extensions
↪">
    <state>
        <peer-restart-time>0</peer-restart-time>
        <peer-restarting>false</peer-restarting>
        <local-restarting>false</local-restarting>
    </state>
</graceful-restart>
```

@line 3: The period of time (advertised by the peer) that the peer expects a restart of a BGP session to take.

@line 4: This flag indicates whether the remote neighbor is currently in the process of restarting, and hence received routes are currently stale.

@line 5: This flag indicates whether the local neighbor is currently restarting. The flag is unset after all NLRI have been advertised to the peer, and the End-of-RIB (EOR) marker has been unset.

BGP Peer Groups Operational State

URL: `/restconf/operational/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-example/peer-groups`

Method: `GET`

Content-Type: `application/xml`

Response Body:

```
<peer-groups>
    <peer-group>
        <peer-group-name>application-peers</peer-group-name>
        <state>
            <total-paths>0</total-paths>
            <total-prefixes>0</total-prefixes>
        </state>
</peer-group>
```

@line 3: Peer Group Identifier.

@line 5: At this moment the cost for count path under effect-rib-in is to high. Therefore the value is the same as total prefixes.

@line 6: Total Prefixes installed under by peers pertaining to this peer group (effective-rib-in). This count doesn't differentiate repeated prefixes.

High Availability

Running OpenDaylight BGP in clustered environment brings an advantage of the plugin's high availability (HA). This section illustrates a basic scenario for HA, also presents a configuration for clustered OpenDaylight BGP.

Contents

Configuration

Following example shows a configuration for running BGP in clustered environment.

1. As the first step, configure (replicated *deafult* shard and *topology* shard if needed) and run OpenDaylight in clustered environment, install BGP and RESTCONF.
2. On one node (OpenDaylight instance), configure BGP speaker instance and neighbor. In addition, configure BGP topology exporter if required. The configuration is shared across all interconnected cluster nodes, however BGP become active only on one node. Other nodes with configured BGP serves as stand-by backups.
3. Remote peer should be configured to accept/initiate connection from/to all OpenDaylight cluster nodes with configured BGP plugin.
4. Connect remote peer, let it advertise some routes. Verify routes presence in Loc-RIB and/or BGP topology exporter instance on all nodes of the OpenDaylight cluster.

Warning: Replicating RIBs across the cluster nodes is causing severe scalability issue and overall performance degradation. To avoid this problems, configure BGP RIB module as a separate shard without enabled replication. Change configuration on all nodes as following (*configuration/initial*):

- **In `modules.conf` add a new module:**

```
{
    name = "bgp_rib"
    namespace = "urn:opendaylight:params:xml:ns:yang:bgp-rib"
    shard-strategy = "module"
}
```

- **In `module-shards.conf` define a new module shard:**

```
{
    name = "bgp_rib"
    shards = [
        {
            name="bgp_rib"
            replicas = [
                "member-1"
            ]
        }
    ]
}
```

Note: Use correct member name in module shard configuration.

Failover scenario

Following section presents a basic BGP speaker failover scenario on 3-node OpenDaylight cluster setup.

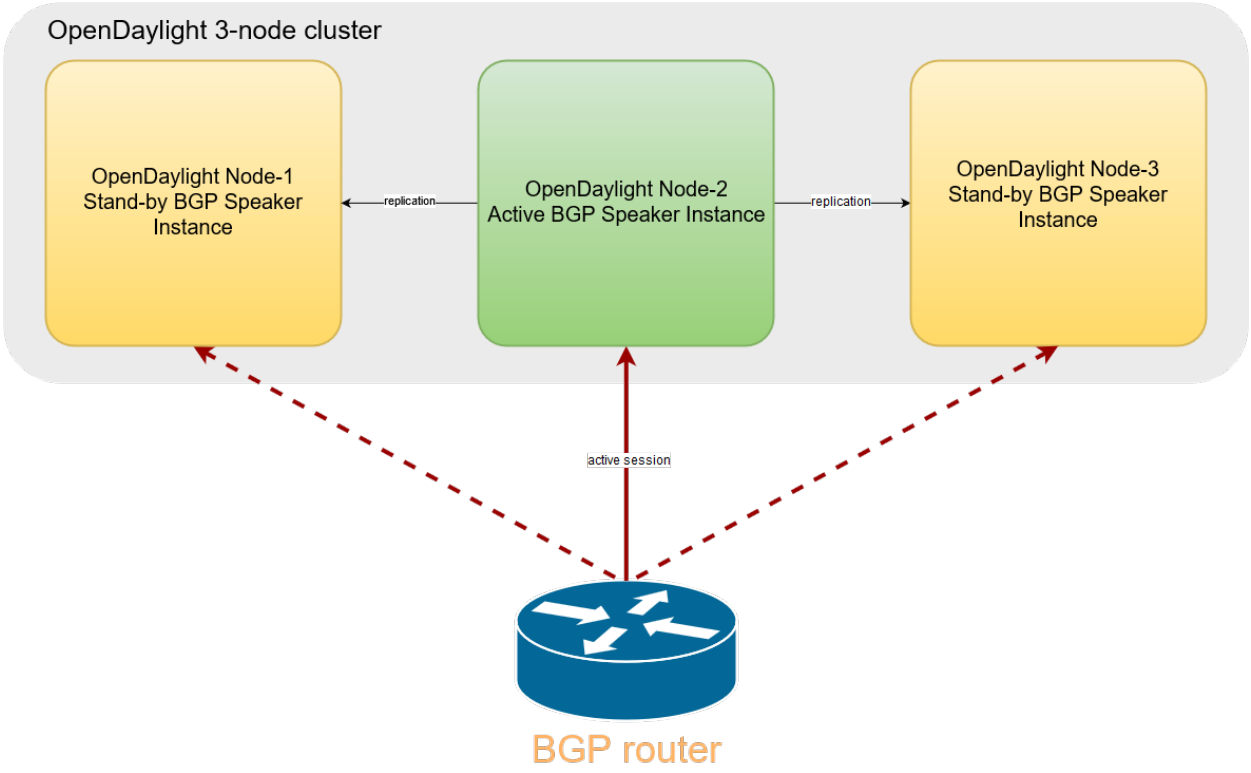


Fig. 1.23: Once the OpenDaylight BGP is configured, the speaker become active on one of the cluster nodes. Remote peer can establish connection with this BGP instance. Routes advertised by remote peer are replicated, hence RIBs state on all nodes is the same.

Topology Provider

This section provides an overview of the BGP topology provider service. It shows how to configure and use all available BGP topology providers. Providers are building topology view of BGP routes stored in local BGP speaker's Loc-RIB. Output topologies are rendered in a form of standardised IETF network topology model.

Contents

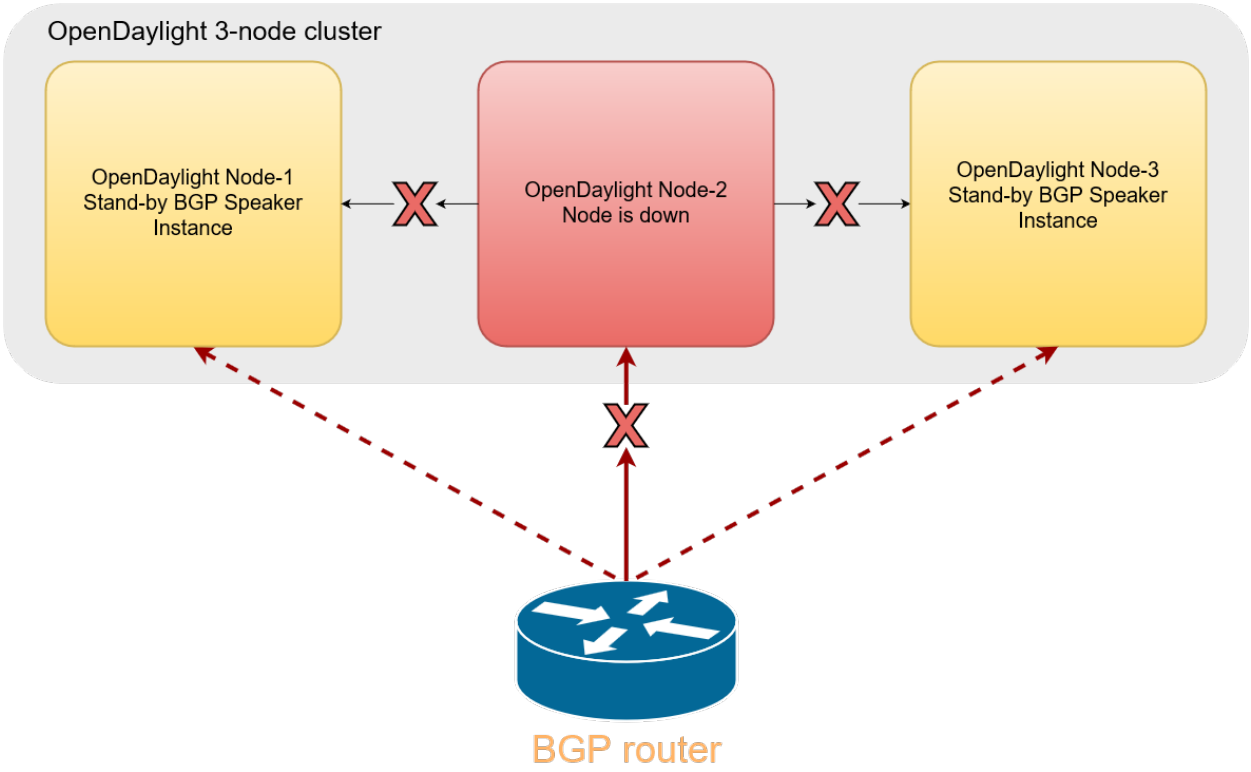


Fig. 1.24: In a case a cluster node, where BGP instance is running, goes down (unexpected failure, restart), active BGP session is dropped.

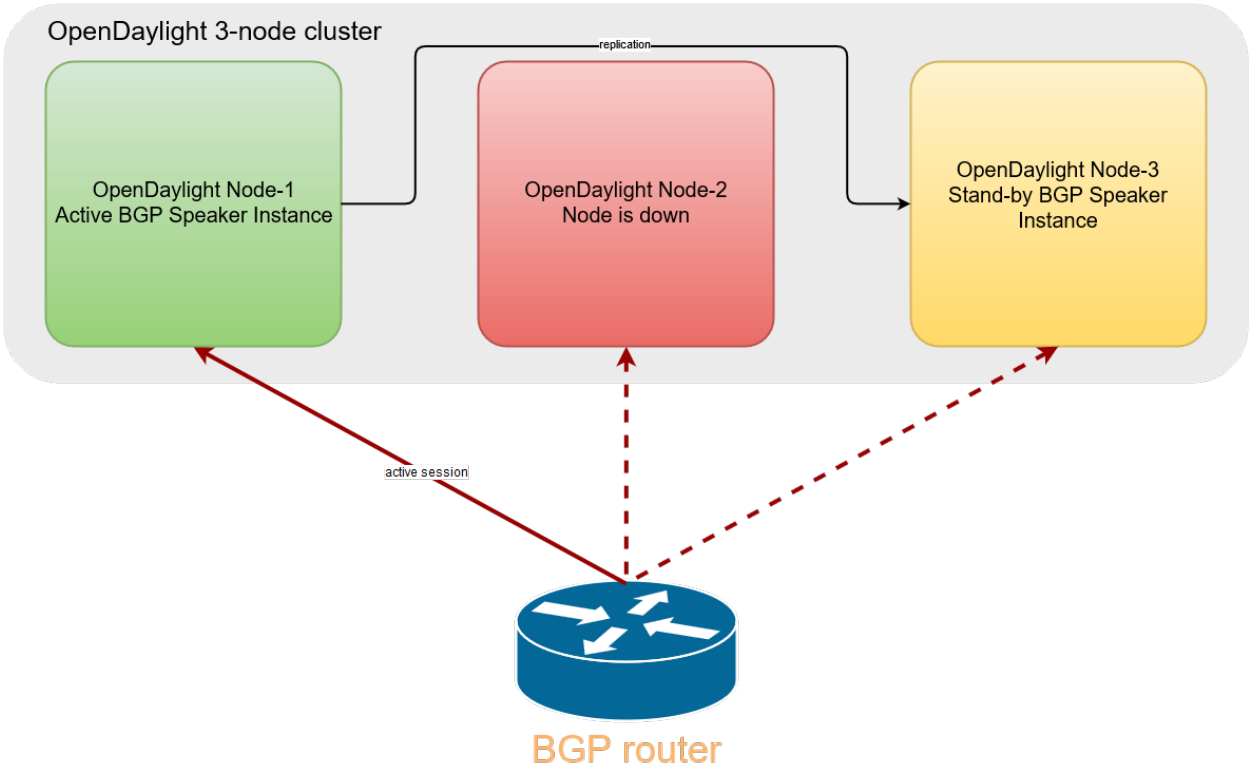


Fig. 1.25: Now, one of the stand-by BGP speaker instances become active. Remote peer establishes new connection and advertises routes again.

Inet Reachability Topology

Inet reachability topology exporter offers a mapping service from IPv4/6 routes to network topology nodes.

Configuration

Following example shows how to create a new instance of IPv4 BGP topology exporter:

URL: `/restconf/config/network-topology:network-topology`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
1 <topology xmlns="urn:TBD:params:xml:ns:yang:network-topology">
2     <topology-id>bgp-example-ipv4-topology</topology-id>
3     <topology-types>
4         <bgp-ipv4-reachability-topology xmlns=
  ↪"urn:opendaylight:params:xml:ns:yang:odl-bgp-topology-types"></bgp-ipv4-
  ↪reachability-topology>
5     </topology-types>
6     <rib-id xmlns="urn:opendaylight:params:xml:ns:yang:odl-bgp-topology-config">bgp-
  ↪example</rib-id>
7 </topology>
```

@line 2: An identifier for a topology.

@line 4: Used to identify type of the topology. In this case BGP IPv4 reachability topology.

@line 6: A name of the local BGP speaker instance.

The topology exporter instance can be removed in a following way:

URL: `/restconf/config/network-topology:network-topology/topology/bgp-example-ipv4-topology`

Method: `DELETE`

Following example shows how to create a new instance of IPv6 BGP topology exporter:

URL: `/restconf/config/network-topology:network-topology`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<topology xmlns="urn:TBD:params:xml:ns:yang:network-topology">
    <topology-id>bgp-example-ipv6-topology</topology-id>
    <topology-types>
        <bgp-ipv6-reachability-topology xmlns=
↪"urn:opendaylight:params:xml:ns:yang:odl-bgp-topology-types"></bgp-ipv6-
↪reachability-topology>
    </topology-types>
    <rib-id xmlns="urn:opendaylight:params:xml:ns:yang:odl-bgp-topology-config">bgp-
↪example</rib-id>
</topology>
```

Usage

Operational state of the topology can be verified via REST:

URL: `/restconf/operational/network-topology:network-topology/topology/bgp-example-ipv4-topology`

Method: `GET`

Response Body:

```
1  <topology xmlns="urn:TBD:params:xml:ns:yang:network-topology">
2      <topology-id>bgp-example-ipv4-topology</topology-id>
3      <server-provided>true</server-provided>
4      <topology-types>
5          <bgp-ipv4-reachability-topology xmlns=
   ↪"urn:opendaylight:params:xml:ns:yang:odl-bgp-topology-types"></bgp-ipv4-
   ↪reachability-topology>
6      </topology-types>
7      <node>
8          <node-id>10.10.1.1</node-id>
9          <igp-node-attributes xmlns="urn:TBD:params:xml:ns:yang:nt:l3-unicast-igp-
   ↪topology">
10             <prefix>
11                 <prefix>10.0.0.10/32</prefix>
12             </prefix>
13         </igp-node-attributes>
14     </node>
15 </topology>
```

@line 8: The identifier of a node in a topology. Its value is mapped from route's NEXT_HOP attribute.

@line 11: The IP prefix attribute of the node. Its value is mapped from routes's destination IP prefix.

BGP Linkstate Topology

BGP linkstate topology exporter offers a mapping service from BGP-LS routes to network topology nodes and links.

Configuration

Following example shows how to create a new instance of linkstate BGP topology exporter:

URL: `/restconf/config/network-topology:network-topology`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<topology  xmlns="urn:TBD:params:xml:ns:yang:network-topology">
    <topology-id>bgp-example-linkstate-topology</topology-id>
    <topology-types>
        <bgp-linkstate-topology xmlns="urn:opendaylight:params:xml:ns:yang:odl-bgp-
↪topology-types"></bgp-linkstate-topology>
    </topology-types>
    <rib-id xmlns="urn:opendaylight:params:xml:ns:yang:odl-bgp-topology-config">bgp-
↪example</rib-id>
</topology>
```

Usage

Operational state of the topology can be verified via REST. A sample output below represents a two node topology with two unidirectional links interconnecting those nodes.

URL: `/restconf/operational/network-topology:network-topology/topology/bgp-example-linkstate-topology`

Method: `GET`

Response Body:

```
<topology xmlns="urn:TBD:params:xml:ns:yang:network-topology">
    <topology-id>bgp-example-linkstate-topology</topology-id>
    <server-provided>true</server-provided>
    <topology-types>
        <bgp-linkstate-topology xmlns="urn:opendaylight:params:xml:ns:yang:odl-bgp-
↪topology-types"></bgp-linkstate-topology>
    </topology-types>
    <node>
        <node-id>bgpls://IsisLevel2:1/type=node&amp;as=65000&amp;domain=673720360&amp;
↪router=0000.0000.0040</node-id>
        <termination-point>
            <tp-id>bgpls://IsisLevel2:1/type=tp&amp;ipv4=203.20.160.40</tp-id>
            <igp-termination-point-attributes xmlns="urn:TBD:params:xml:ns:yang:nt:l3-
↪unicast-igp-topology"/>
        </termination-point>
        <igp-node-attributes xmlns="urn:TBD:params:xml:ns:yang:nt:l3-unicast-igp-
↪topology">
            <prefix>
                <prefix>40.40.40.40/32</prefix>
                <metric>10</metric>
            </prefix>
            <prefix>
                <prefix>203.20.160.0/24</prefix>
                <metric>10</metric>
            </prefix>
            <name>node1</name>
            <router-id>40.40.40.40</router-id>
            <isis-node-attributes xmlns="urn:TBD:params:xml:ns:yang:network:isis-
↪topology">
```

```
                <ted>
                    <te-router-id-ipv4>40.40.40.40</te-router-id-ipv4>
                </ted>
                <iso>
                    <iso-system-id>MDAwMDAwMDAwMDY0</iso-system-id>
                </iso>
            </isis-node-attributes>
        </igp-node-attributes>
    </node>
    <node>
        <node-id>bgpls://IsisLevel2:1/type=node&amp;as=65000&amp;domain=673720360&amp;
↪router=0000.0000.0039</node-id>
        <termination-point>
            <tp-id>bgpls://IsisLevel2:1/type=tp&amp;ipv4=203.20.160.39</tp-id>
            <igp-termination-point-attributes xmlns="urn:TBD:params:xml:ns:yang:nt:l3-
↪unicast-igp-topology"/>
        </termination-point>
        <igp-node-attributes xmlns="urn:TBD:params:xml:ns:yang:nt:l3-unicast-igp-
↪topology">
            <prefix>
                <prefix>39.39.39.39/32</prefix>
                <metric>10</metric>
            </prefix>
            <prefix>
                <prefix>203.20.160.0/24</prefix>
                <metric>10</metric>
            </prefix>
            <name>node2</name>
            <router-id>39.39.39.39</router-id>
            <isis-node-attributes xmlns="urn:TBD:params:xml:ns:yang:network:isis-
↪topology">
                <ted>
                    <te-router-id-ipv4>39.39.39.39</te-router-id-ipv4>
                </ted>
                <iso>
                    <iso-system-id>MDAwMDAwMDAwMDg3</iso-system-id>
                </iso>
            </isis-node-attributes>
        </igp-node-attributes>
    </node>
    <link>
        <destination>
            <dest-node>bgpls://IsisLevel2:1/type=node&amp;as=65000&amp;
↪domain=673720360&amp;router=0000.0000.0039</dest-node>
            <dest-tp>bgpls://IsisLevel2:1/type=tp&amp;ipv4=203.20.160.39</dest-tp>
        </destination>
        <link-id>bgpls://IsisLevel2:1/type=link&amp;local-as=65000&amp;local-
↪domain=673720360&amp;local-router=0000.0000.0040&amp;remote-as=65000&amp;remote-
↪domain=673720360&amp;remote-router=0000.0000.0039&amp;ipv4-iface=203.20.160.40&amp;
↪ipv4-neigh=203.20.160.39</link-id>
        <source>
            <source-node>bgpls://IsisLevel2:1/type=node&amp;as=65000&amp;
↪domain=673720360&amp;router=0000.0000.0040</source-node>
            <source-tp>bgpls://IsisLevel2:1/type=tp&amp;ipv4=203.20.160.40</source-tp>
        </source>
        <igp-link-attributes xmlns="urn:TBD:params:xml:ns:yang:nt:l3-unicast-igp-
↪topology">
            <metric>10</metric>
```

```
            <isis-link-attributes xmlns="urn:TBD:params:xml:ns:yang:network:isis-
↪topology">
                <ted>
                    <color>0</color>
                    <max-link-bandwidth>1250000.0</max-link-bandwidth>
                    <max-resv-link-bandwidth>12500.0</max-resv-link-bandwidth>
                    <te-default-metric>0</te-default-metric>
                    <unreserved-bandwidth>
                        <bandwidth>12500.0</bandwidth>
                        <priority>0</priority>
                    </unreserved-bandwidth>
                    <unreserved-bandwidth>
                        <bandwidth>12500.0</bandwidth>
                        <priority>1</priority>
                    </unreserved-bandwidth>
                    <unreserved-bandwidth>
                        <bandwidth>12500.0</bandwidth>
                        <priority>2</priority>
                    </unreserved-bandwidth>
                    <unreserved-bandwidth>
                        <bandwidth>12500.0</bandwidth>
                        <priority>3</priority>
                    </unreserved-bandwidth>
                    <unreserved-bandwidth>
                        <bandwidth>12500.0</bandwidth>
                        <priority>4</priority>
                    </unreserved-bandwidth>
                    <unreserved-bandwidth>
                        <bandwidth>12500.0</bandwidth>
                        <priority>5</priority>
                    </unreserved-bandwidth>
                    <unreserved-bandwidth>
                        <bandwidth>12500.0</bandwidth>
                        <priority>6</priority>
                    </unreserved-bandwidth>
                    <unreserved-bandwidth>
                        <bandwidth>12500.0</bandwidth>
                        <priority>7</priority>
                    </unreserved-bandwidth>
                </ted>
            </isis-link-attributes>
        </igp-link-attributes>
    </link>
    <link>
        <destination>
            <dest-node>bgpls://IsisLevel2:1/type=node&amp;as=65000&amp;
↪domain=673720360&amp;router=0000.0000.0040</dest-node>
            <dest-tp>bgpls://IsisLevel2:1/type=tp&amp;ipv4=203.20.160.40</dest-tp>
        </destination>
        <link-id>bgpls://IsisLevel2:1/type=link&amp;local-as=65000&amp;local-
↪domain=673720360&amp;local-router=0000.0000.0039&amp;remote-as=65000&amp;remote-
↪domain=673720360&amp;remote-router=0000.0000.0040&amp;ipv4-iface=203.20.160.39&amp;
↪ipv4-neigh=203.20.160.40</link-id>
        <source>
            <source-node>bgpls://IsisLevel2:1/type=node&amp;as=65000&amp;
↪domain=673720360&amp;router=0000.0000.0039</source-node>
            <source-tp>bgpls://IsisLevel2:1/type=tp&amp;ipv4=203.20.160.39</source-tp>
        </source>
```

```
        <igp-link-attributes xmlns="urn:TBD:params:xml:ns:yang:nt:l3-unicast-igp-
↪topology">
            <metric>10</metric>
            <isis-link-attributes xmlns="urn:TBD:params:xml:ns:yang:network:isis-
↪topology">
                <ted>
                    <color>0</color>
                    <max-link-bandwidth>1250000.0</max-link-bandwidth>
                    <max-resv-link-bandwidth>12500.0</max-resv-link-bandwidth>
                    <te-default-metric>0</te-default-metric>
                    <unreserved-bandwidth>
                        <bandwidth>12500.0</bandwidth>
                        <priority>0</priority>
                    </unreserved-bandwidth>
                    <unreserved-bandwidth>
                        <bandwidth>12500.0</bandwidth>
                        <priority>1</priority>
                    </unreserved-bandwidth>
                    <unreserved-bandwidth>
                        <bandwidth>12500.0</bandwidth>
                        <priority>2</priority>
                    </unreserved-bandwidth>
                    <unreserved-bandwidth>
                        <bandwidth>12500.0</bandwidth>
                        <priority>3</priority>
                    </unreserved-bandwidth>
                    <unreserved-bandwidth>
                        <bandwidth>12500.0</bandwidth>
                        <priority>4</priority>
                    </unreserved-bandwidth>
                    <unreserved-bandwidth>
                        <bandwidth>12500.0</bandwidth>
                        <priority>5</priority>
                    </unreserved-bandwidth>
                    <unreserved-bandwidth>
                        <bandwidth>12500.0</bandwidth>
                        <priority>6</priority>
                    </unreserved-bandwidth>
                    <unreserved-bandwidth>
                        <bandwidth>12500.0</bandwidth>
                        <priority>7</priority>
                    </unreserved-bandwidth>
                </ted>
            </isis-link-attributes>
        </igp-link-attributes>
    </link>
</topology>
```

BGP Network Topology Configuration Loader

BGP Network Topology Configuration Loader allows user to define static initial configuration for a BGP protocol instance. This service will detect the creation of new configuration files following the pattern "network-topology-*.xml" under the path "etc/opendaylight/bgp". Once the file is processed, the defined configuration will be available from the configuration Data Store.

Note: If the BGP topology instance is already present, no update or configuration will be applied.

When installing BGP an example will be provided and a default configuration loaded.

PATH: `etc/opendaylight/bgp/network-topology-config.xml`

```
<network-topology xmlns="urn:TBD:params:xml:ns:yang:network-topology">
    <topology>
        <topology-id>example-ipv4-topology</topology-id>
        <topology-types>
            <bgp-ipv4-reachability-topology xmlns=
↪"urn:opendaylight:params:xml:ns:yang:odl-bgp-topology-types"/>
        </topology-types>
        <rib-id xmlns="urn:opendaylight:params:xml:ns:yang:odl-bgp-topology-config">
↪example-bgp-rib</rib-id>
    </topology>
    <topology>
        <topology-id>example-ipv6-topology</topology-id>
        <topology-types>
            <bgp-ipv6-reachability-topology xmlns=
↪"urn:opendaylight:params:xml:ns:yang:odl-bgp-topology-types"/>
        </topology-types>
        <rib-id xmlns="urn:opendaylight:params:xml:ns:yang:odl-bgp-topology-config">
↪example-bgp-rib</rib-id>
    </topology>
    <topology>
        <topology-id>example-linkstate-topology</topology-id>
        <topology-types>
            <bgp-linkstate-topology xmlns="urn:opendaylight:params:xml:ns:yang:odl-
↪bgp-topology-types"/>
        </topology-types>
        <rib-id xmlns="urn:opendaylight:params:xml:ns:yang:odl-bgp-topology-config">
↪example-bgp-rib</rib-id>
    </topology>
</network-topology>
```

Test Tools

BGP test tools serves to test basic BGP functionality, scalability and performance.

Contents

BGP Test Tool

The BGP Test Tool is a stand-alone Java application purposed to simulate remote BGP peers, that are capable to advertise sample routes. This application is not part of the OpenDaylight Karaf distribution, however it can be downloaded from OpenDaylight's Nexus (use latest release version):

`https://nexus.opendaylight.org/content/repositories/opendaylight.release/org/opendaylight/bgpcep/bgp-testtool`

Usage

The application can be run from command line:

```
java -jar bgp-testtool-*-executable.jar
```

with optional input parameters:

```
-i <BOOLEAN>, --active <BOOLEAN>
   Active initialisation of the connection, by default false.

-ho <N>, --holdtimer <N>
   In seconds, value of the desired holdtimer, by default 90.

-sc <N>, --speakersCount <N>
   Number of simulated BGP speakers, when creating each speaker, uses incremented␣
↪local-address for binding, by default 0.

-ra <IP_ADDRESS:PORT,...>, --remoteAddress <IP_ADDRESS:PORT,...>
   A list of IP addresses of remote BGP peers, that the tool can accept or initiate␣
↪connect to that address (based on the mode), by default 192.0.2.2:1790.

-la <IP_ADDRESS:PORT>, --localAddress <IP_ADDRESS:PORT>
   IP address of BGP speakers which the tools simulates, by default 192.0.2.2:0.

-pr <N>, --prefixes <N>
   Number of prefixes to be advertised by each simulated speaker

-mp <BOOLEAN>, --multiPathSupport <BOOLEAN>
   Active ADD-PATH support, by default false.

-as <N>, --as <N>
   Local AS Number, by default 64496.

-ec <EXTENDED_COMMUNITIES>, --extended_communities <EXTENDED_COMMUNITIES>
   Extended communities to be send. Format: x,x,x  where  x  is  each  extended ␣
↪community from bgp-types.yang, by default empty.

-ll <LOG_LEVEL>, --log_level <LOG_LEVEL>
   Log level for console output, by default INFO.
```

BGP Application Peer Benchmark

It is a simple OpenDaylight application which is capable to inject and remove specific amount of IPv4 routes. This application is part of the OpenDaylight Karaf distribution.

Configuration

As a first step install BGP and RESTCONF, then configure *Application Peer*. Install `odl-bgpcep-bgp-benchmark` feature and reconfigure BGP Application Peer Benchmark application as per following:

URL: `/restconf/config/odl-bgp-app-peer-benchmark-config:config`

Method: `PUT`

Content-Type: `application/xml`

Request Body:

```
<odl-bgp-app-peer-benchmark-config xmlns="urn:opendaylight:params:xml:ns:yang:odl-bgp-app-peer-benchmark-config">
   <app-peer-id xmlns="urn:opendaylight:params:xml:ns:yang:odl-bgp-app-peer-benchmark-config">10.25.1.9</app-peer-id>
</odl-bgp-app-peer-benchmark-config>
```

@line 2: The *Application Peer* identifier.

Inject routes

Routes injection can be invoked via RPC:

URL: `/restconf/operations/odl-bgp-app-peer-benchmark:add-prefix`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<input xmlns="urn:opendaylight:params:xml:ns:yang:odl-bgp-app-peer-benchmark">
    <prefix>1.1.1.1/32</prefix>
    <count>100000</count>
    <batchsize>2000</batchsize>
    <nexthop>192.0.2.2</nexthop>
</input>
```

@line 2: A initial IPv4 prefix carried in route. Value is incremented for following routes.

@line 3: An amount of routes to be added to *Application Peer's* programmable RIB.

@line 4: A size of the transaction batch.

@line 5: A NEXT_HOP attribute value used in all injected routes.

Response Body:

```
<output xmlns="urn:opendaylight:params:xml:ns:yang:odl-bgp-app-peer-benchmark">
    <result>
        <duration>4301</duration>
        <rate>25000</rate>
        <count>100000</count>
    </result>
</output>
```

@line 3: Request duration in milliseconds.

@line 4: Writes per second rate.

@line 5: An amount of routes added to *Application Peer's* programmable RIB.

Remove routes

Routes deletion can be invoked via RPC:

URL: `/restconf/operations/odl-bgp-app-peer-benchmark:delete-prefix`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<input xmlns="urn:opendaylight:params:xml:ns:yang:odl-bgp-app-peer-benchmark">
    <prefix>1.1.1.1/32</prefix>
    <count>100000</count>
    <batchsize>2000</batchsize>
</input>
```

@line 2: A initial IPv4 prefix carried in route to be removed. Value is incremented for following routes.

@line 3: An amount of routes to be removed from *Application Peer's* programmable RIB.

@line 4: A size of the transaction batch.

Response Body:

```
<output xmlns="urn:opendaylight:params:xml:ns:yang:odl-bgp-app-peer-benchmark">
    <result>
        <duration>1837</duration>
        <rate>54500</rate>
        <count>100000</count>
    </result>
</output>
```

Troubleshooting

This section offers advices in a case OpenDaylight BGP plugin is not working as expected.

Contents

BGP is not working...

- First of all, ensure that all required features are installed, local and remote BGP configuration is correct.
- Check OpenDaylight Karaf logs:

From Karaf console:

```
log:tail
```

or open log file: `data/log/karaf.log`

Possibly, a reason/hint for a cause of the problem can be found there.

- Try to minimise effect of other OpenDaylight features, when searching for a reason of the problem.
- Try to set DEBUG severity level for BGP logger via Karaf console commands, in order to collect more information:

```
log:set DEBUG org.opendaylight.protocol.bgp
```

```
log:set DEBUG org.opendaylight.bgpcep.bgp
```

Bug reporting

Before you report a bug, check BGPCEP Jira to ensure same/similar bug is not already filed there.

Write an e-mail to bgpcep-users@lists.opendaylight.org and provide following information:

1. State OpenDaylight version
2. Describe your use-case and provide as much details related to BGP as possible
3. Steps to reproduce
4. Attach Karaf log files, optionally packet captures, REST input/output

BGP Monitoring Protocol User Guide

This guide contains information on how to use the OpenDaylight BGP Monitoring Protocol (BMP) plugin. It covers BMP basic concepts, supported capabilities, configuration and operations.

Contents

Overview

This section provides high-level overview of the BMP plugin, OpenDaylight implementation and BMP usage for SDN.

Contents

BGP Monitoring Protocol

The BGP Monitoring Protocol (BMP) serves to monitor BGP sessions. The BMP can be used to obtain route view instead of screen scraping. The BMP provides access to unprocessed routing information (Adj-RIB-In) and processed routes (applied inbound policy) of monitored router's peer. In addition, monitored router can provide periodic dump of statistics.

The BMP runs over TCP. Both monitored router and monitoring station can be configured as active or passive party of the connection. The passive party listens at particular port. The router can be monitored by multiple monitoring stations. BMP messages are sent by monitored router only, monitoring station supposed to collect and process data received over BMP.

BMP in SDN

The main concept of BMP is to monitor BGP sessions - monitoring station is aware of monitored peer's status, collects statistics and analyzes them in order to provide valuable information for network operators.

Moreover, BMP provides provides peer RIBs visibility, without need to establish BGP sessions. Unprocessed routes may serve as a source of information for software-driven routing optimization. In this case, SDN controller, a BMP monitoring station, collects routing information from monitored routers. The routes are used in subsequent optimization procedures.

OpenDaylight BMP plugin

The OpenDaylight BMP plugin provides monitoring station implementation. The plugin can establish BMP session with one or more monitored routers in order to collect routing and statistical information.

- Runtime configurable monitoring station
- Read-only routes and statistics view
- Supports various routing information types

Important: The BMP plugin is not storing historical data, it provides current snapshot only.

List of supported capabilities

The BMP plugin implementation is based on Internet standards:

- RFC7854 - BGP Monitoring Protocol (BMP)

Note: The BMP plugin is capable to process various types of routing information (IP Unicast, EVPN, L3VPN, Link-State,...). Please, see complete list in BGP user guide.

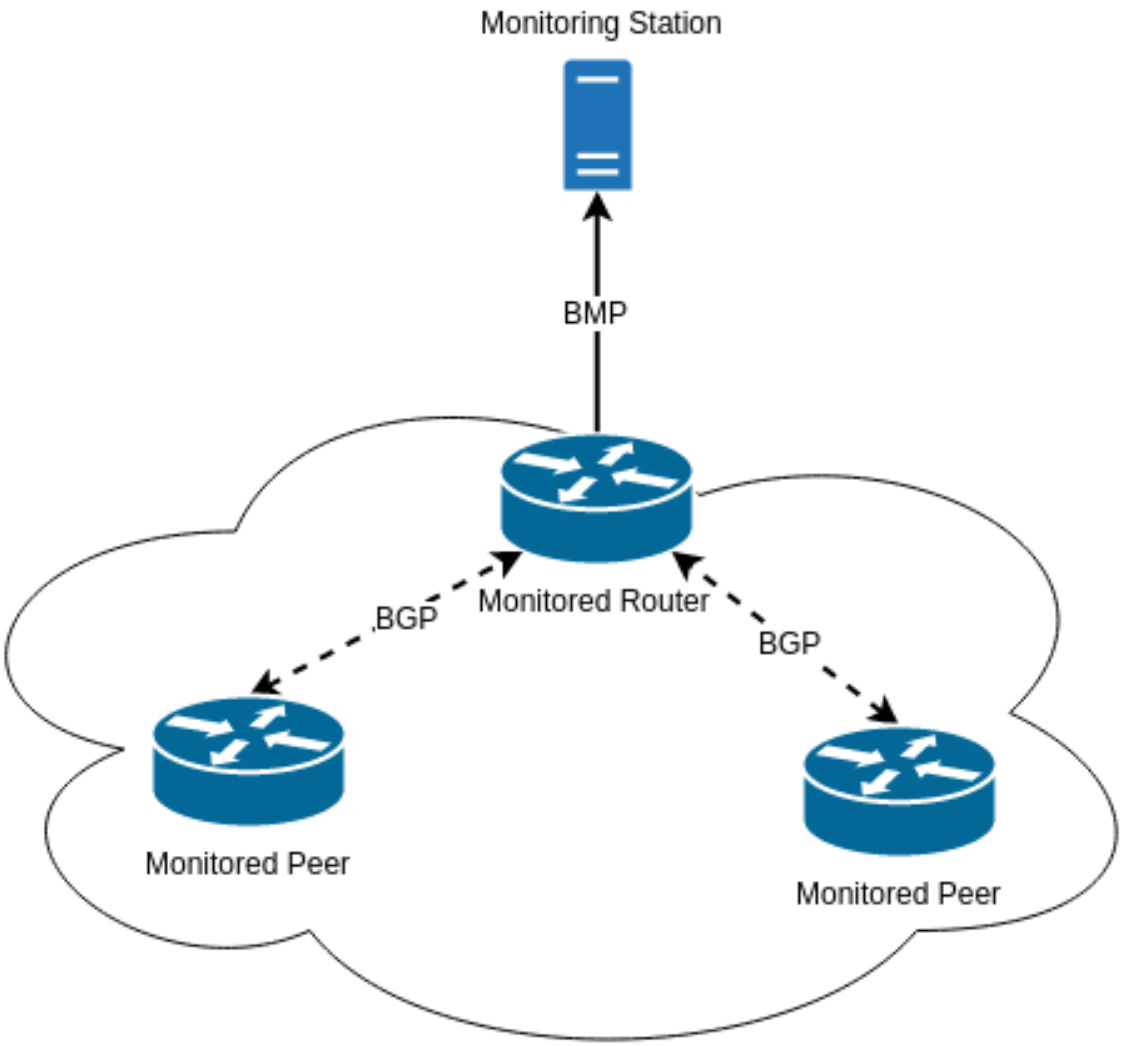


Fig. 1.26: The BMP overview - Monitoring Station, Monitored Router and Monitored Peers.

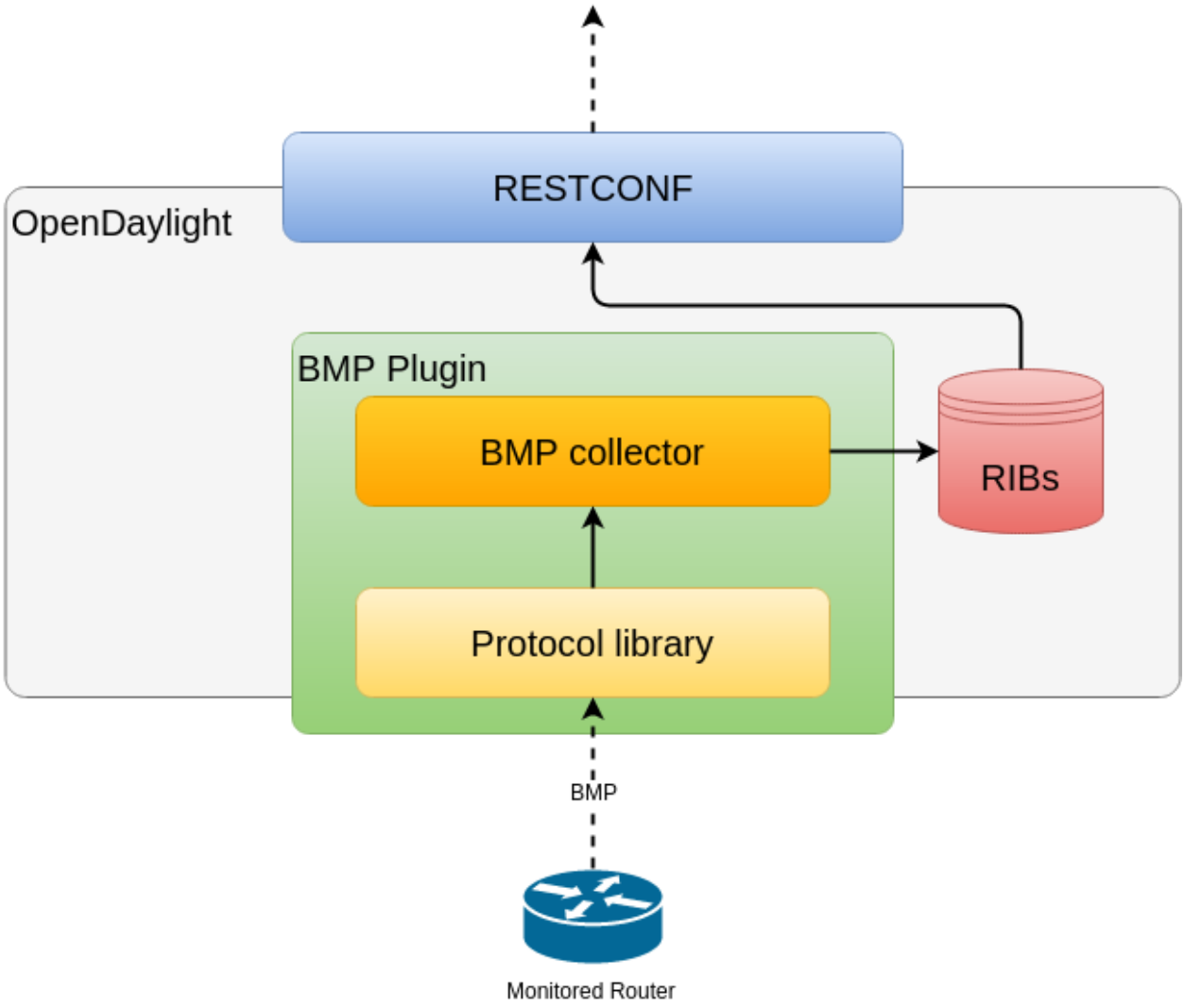


Fig. 1.27: OpenDaylight BMP plugin overview.

Running BMP

This section explains how to install BMP plugin.

1. Install BMP feature - `odl-bgpcep-bmp`. Also, for sake of this sample, it is required to install RESTCONF. In the Karaf console, type command:

```
feature:install odl-restconf odl-bgpcep-bmp
```

2. The BMP plugin contains a default configuration, which is applied after the feature starts up. One instance of BMP monitoring station is created (named *example-bmp-monitor*), and its presence can be verified via REST:

   URL: `/restconf/config/odl-bmp-monitor-config:odl-bmp-monitors/bmp-monitor-config/example-bmp-monitor`

   Method: `GET`

   Response Body:

```
<bmp-monitor-config xmlns="urn:opendaylight:params:xml:ns:yang:bmp-monitor-config
↪">
   <monitor-id>example-bmp-monitor</monitor-id>
   <server>
      <binding-port>12345</binding-port>
      <binding-address>0.0.0.0</binding-address>
   </server>
</bmp-monitor-config>
```

BMP Monitoring Station

The following section shows how to configure BMP basics, how to verify functionality and presents essential components of the plugin. Next samples demonstrate the plugin's runtime configuration capability.

The monitoring station is responsible for received BMP PDUs processing and storage. The default BMP server is listening at port *12345*.

Contents

Configuration

This section shows the way to configure the BMP monitoring station via REST API.

Monitoring station configuration

In order to change default's BMP monitoring station configuration, use following request.

URL: `/restconf/config/odl-bmp-monitor-config:odl-bmp-monitors/bmp-monitor-config/example-bmp-monitor`

Method: `PUT`

Content-Type: `application/xml`

Request Body:

```
1  <bmp-monitor-config xmlns="urn:opendaylight:params:xml:ns:yang:bmp-monitor-config">
2     <monitor-id>example-bmp-monitor</monitor-id>
3     <server>
4        <binding-port>12345</binding-port>
5        <binding-address>0.0.0.0</binding-address>
6     </server>
7  </bmp-monitor-config>
```

@line 4: **binding-port** - The BMP server listening port.

@line 5: **binding-address** - The BMP server biding address.

Note: User may create multiple BMP monitoring station instances at runtime.

Active mode configuration

In order to enable active connection, use following request.

URL: `/restconf/config/odl-bmp-monitor-config:odl-bmp-monitors/bmp-monitor-config/example-bmp-monitor`

Method: `PUT`

Content-Type: `application/xml`

Request Body:

```
1  <bmp-monitor-config xmlns="urn:opendaylight:params:xml:ns:yang:bmp-monitor-config">
2     <monitor-id>example-bmp-monitor</monitor-id>
3     <server>
4        <binding-port>12345</binding-port>
5        <binding-address>0.0.0.0</binding-address>
6     </server>
7     <monitored-router>
8        <address>192.0.2.2</address>
9        <port>1234</port>
10       <active>true</active>
11    </monitored-router>
12 </bmp-monitor-config>
```

@line 8: **address** - The monitored router's IP address.

@line 9: **port** - The monitored router's port.

@line 10: **active** - Active mode set.

Note: User may configure active session establishment for multiple monitored routers.

MD5 authentication configuration

In order to enable active connection, use following request.

URL: `/restconf/config/odl-bmp-monitor-config:odl-bmp-monitors/bmp-monitor-config/example-bmp-monitor`

Method: `PUT`

Content-Type: `application/xml`

Request Body:

```
1  <bmp-monitor-config xmlns="urn:opendaylight:params:xml:ns:yang:bmp-monitor-config">
2     <monitor-id>example-bmp-monitor</monitor-id>
3     <server>
4        <binding-port>12345</binding-port>
5        <binding-address>0.0.0.0</binding-address>
6     </server>
7     <monitored-router>
8        <address>192.0.2.2</address>
9        <password>changeme</password>
10    </monitored-router>
11 </bmp-monitor-config>
```

@line 8: **address** - The monitored router's IP address.

@line 9: **password** - The TCP MD5 signature.

Collector DB Tree

```
module: bmp-monitor
   +--rw bmp-monitor
      +--ro monitor* [monitor-id]
         +--ro monitor-id    monitor-id
         +--ro router* [router-id]
            +--ro name?          string
            +--ro description?   string
            +--ro info?          string
            +--ro router-id      router-id
            +--ro status?        status
            +--ro peer* [peer-id]
               +--ro peer-id                rib:peer-id
               +--ro type                   peer-type
               x--ro distinguisher
               |  +--ro distinguisher-type?   distinguisher-type
               |  +--ro distinguisher?        string
               +--ro peer-distinguisher?    union
               +--ro address                inet:ip-address
               +--ro as                     inet:as-number
               +--ro bgp-id                 inet:ipv4-address
               +--ro router-distinguisher?  string
               +--ro peer-session
```

```
|  +--ro local-address      inet:ip-address
|  +--ro local-port         inet:port-number
|  +--ro remote-port        inet:port-number
|  +--ro sent-open
|  |  +--ro version?          protocol-version
|  |  +--ro my-as-number?     uint16
|  |  +--ro hold-timer        uint16
|  |  +--ro bgp-identifier    inet:ipv4-address
|  |  +--ro bgp-parameters*
|  |     +--ro optional-capabilities*
|  |        +--ro c-parameters
|  |           +--ro as4-bytes-capability
|  |           |  +--ro as-number?   inet:as-number
|  |           +--ro bgp-extended-message-capability!
|  |           +--ro multiprotocol-capability
|  |           |  +--ro afi?    identityref
|  |           |  +--ro safi?   identityref
|  |           +--ro graceful-restart-capability
|  |           |  +--ro restart-flags    bits
|  |           |  +--ro restart-time     uint16
|  |           |  +--ro tables* [afi safi]
|  |           |     +--ro afi          identityref
|  |           |     +--ro safi         identityref
|  |           |     +--ro afi-flags    bits
|  |           +--ro add-path-capability
|  |           |  +--ro address-families*
|  |           |     +--ro afi?            identityref
|  |           |     +--ro safi?           identityref
|  |           |     +--ro send-receive?   send-receive
|  |           +--ro route-refresh-capability!
|  +--ro received-open
|  |  +--ro version?          protocol-version
|  |  +--ro my-as-number?     uint16
|  |  +--ro hold-timer        uint16
|  |  +--ro bgp-identifier    inet:ipv4-address
|  |  +--ro bgp-parameters*
|  |     +--ro optional-capabilities*
|  |        +--ro c-parameters
|  |           +--ro as4-bytes-capability
|  |           |  +--ro as-number?   inet:as-number
|  |           +--ro bgp-extended-message-capability!
|  |           +--ro multiprotocol-capability
|  |           |  +--ro afi?    identityref
|  |           |  +--ro safi?   identityref
|  |           +--ro graceful-restart-capability
|  |           |  +--ro restart-flags    bits
|  |           |  +--ro restart-time     uint16
|  |           |  +--ro tables* [afi safi]
|  |           |     +--ro afi          identityref
|  |           |     +--ro safi         identityref
|  |           |     +--ro afi-flags    bits
|  |           +--ro add-path-capability
|  |           |  +--ro address-families*
|  |           |     +--ro afi?            identityref
|  |           |     +--ro safi?           identityref
|  |           |     +--ro send-receive?   send-receive
|  |           +--ro route-refresh-capability!
|  +--ro information
```

```
|  |  +--ro string-information*
|  |     +--ro string-tlv
|  |        +--ro string-info?   string
|  +--ro status?            status
|  +--ro timestamp-sec?     yang:timestamp
|  +--ro timestamp-micro?   yang:timestamp
+--ro stats
|  +--ro rejected-prefixes?                 yang:counter32
|  +--ro duplicate-prefix-advertisements?   yang:counter32
|  +--ro duplicate-withdraws?               yang:counter32
|  +--ro invalidated-cluster-list-loop?     yang:counter32
|  +--ro invalidated-as-path-loop?          yang:counter32
|  +--ro invalidated-originator-id?         yang:counter32
|  +--ro invalidated-as-confed-loop?        yang:counter32
|  +--ro adj-ribs-in-routes?                yang:gauge64
|  +--ro loc-rib-routes?                    yang:gauge64
|  +--ro per-afi-safi-adj-rib-in-routes
|  |  +--ro afi-safi* [afi safi]
|  |     +--ro afi      identityref
|  |     +--ro safi     identityref
|  |     +--ro count?   yang:gauge64
|  +--ro per-afi-safi-loc-rib-routes
|  |  +--ro afi-safi* [afi safi]
|  |     +--ro afi      identityref
|  |     +--ro safi     identityref
|  |     +--ro count?   yang:gauge64
|  +--ro updates-treated-as-withdraw?       yang:counter32
|  +--ro prefixes-treated-as-withdraw?      yang:counter32
|  +--ro duplicate-updates?                 yang:counter32
|  +--ro timestamp-sec?                     yang:timestamp
|  +--ro timestamp-micro?                   yang:timestamp
+--ro pre-policy-rib
|  +--ro tables* [afi safi]
|     +--ro afi           identityref
|     +--ro safi          identityref
|     +--ro attributes
|     |  +--ro uptodate?   boolean
|     +--ro (routes)?
+--ro post-policy-rib
|  +--ro tables* [afi safi]
|     +--ro afi           identityref
|     +--ro safi          identityref
|     +--ro attributes
|     |  +--ro uptodate?   boolean
|     +--ro (routes)?
+--ro mirrors
   +--ro information?       bmp-msg:mirror-information-code
   +--ro timestamp-sec?     yang:timestamp
   +--ro timestamp-micro?   yang:timestamp
```

Operations

The BMP plugin offers view of collected routes and statistical information from monitored peers. To get top-level view of monitoring station:

URL: `/restconf/operational/bmp-monitor:bmp-monitor/monitor/example-bmp-monitor`

Method: `GET`

Response Body:

```
<bmp-monitor xmlns="urn:opendaylight:params:xml:ns:yang:bmp-monitor">
   <monitor>
      <monitor-id>example-bmp-monitor</monitor-id>
         <router>
            <router-id>10.10.10.10</router-id>
            <name>name</name>
            <description>monitored-router</description>
            <info>monitored router;</info>
            <status>up</status>
            <peer>
                <peer-id>20.20.20.20</peer-id>
                <address>20.20.20.20</address>
                <bgp-id>20.20.20.20</bgp-id>
                <as>65000</as>
                <type>global</type>
                <peer-session>
                  <remote-port>1790</remote-port>
                  <timestamp-sec>0</timestamp-sec>
                  <status>up</status>
                  <local-address>10.10.10.10</local-address>
                  <local-port>2200</local-port>
                  <received-open>
                     <hold-timer>180</hold-timer>
                     <my-as-number>65000</my-as-number>
                     <bgp-identifier>20.20.20.20</bgp-identifier>
                  </received-open>
                  <sent-open>
                     <hold-timer>180</hold-timer>
                     <my-as-number>65000</my-as-number>
                     <bgp-identifier>65000</bgp-identifier>
                  </sent-open>
                </peer-session>
                <pre-policy-rib>
                  <tables>
                     <afi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">
↪x:ipv4-address-family</afi>
                     <safi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">
↪x:unicast-subsequent-address-family</safi>
                     <ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet
↪">
                        <ipv4-route>
                           <prefix>10.10.10.0/24</prefix>
                           <attributes>
                           ...
                           </attributes>
                        </ipv4-route>
                     </ipv4-routes>
                     <attributes>
                        <uptodate>true</uptodate>
                     </attributes>
                  </tables>
                </pre-policy-rib>
                <post-policy-rib>
                  ...
                </post-policy-rib>
```

```
53                  <stats>
54                    <timestamp-sec>0</timestamp-sec>
55                    <invalidated-cluster-list-loop>0</invalidated-cluster-list-loop>
56                    <duplicate-prefix-advertisements>0</duplicate-prefix-advertisements>
57                    <loc-rib-routes>100</loc-rib-routes>
58                    <duplicate-withdraws>0</duplicate-withdraws>
59                    <invalidated-as-confed-loop>0</invalidated-as-confed-loop>
60                    <adj-ribs-in-routes>10</adj-ribs-in-routes>
61                    <invalidated-as-path-loop>0</invalidated-as-path-loop>
62                    <invalidated-originator-id>0</invalidated-originator-id>
63                    <rejected-prefixes>8</rejected-prefixes>
64                  </stats>
65                </peer>
66            </router>
67        </monitor>
68    </bmp-monitor>
```

@line 3: **monitor-id** - The BMP monitoring station instance identifier.

@line 5: **router-id** - The monitored router IP address, serves as an identifier.

@line 11: **peer-id** - The monitored peer's BGP identifier, serves a an identifier.

@line 12: **address** - The IP address of the peer, associated with the TCP session.

@line 13: **bgp-id** - The BGP Identifier of the peer.

@line 14: **as** - The Autonomous System number of the peer.

@line 15: **type** - Identifies type of the peer - *Global Instance*, *RD Instance* or *Local Instance*

@line 17: **remote-port** - The peer's port number associated with TCP session.

@line 20: **local-address** - The IP address of the monitored router associated with the peering TCP session.

@line 21: **local-port** - The port number of the monitored router associated with the peering TCP session.

@line 22: **received-open** - The full OPEN message received by monitored router from the peer.

@line 27: **sent-open** - The full OPEN message send by monitored router to the peer.

@line 33: **pre-policy-rib** - The Adj-RIB-In that contains unprocessed routing information.

@line 50: **post-policy-rib** - The Post-Policy Ad-RIB-In that contains routes filtered by inbound policy.

@line 53: **stats** - Contains various statistics, periodically updated by the router.

- **To view collected information from particular monitored router: URL:** `/restconf/operational/bmp-monitor:bmp-monitor/monitor/example-bmp-monitor/router/10.10.10.10`
- **To view collected information from particular monitored peer: URL:** `/restconf/operational/bmp-monitor:bmp-monitor/monitor/example-bmp-monitor/router/10.10.10.10/peer/20.20.20.20`

Test tools

BMP test tool serves to test basic BMP functionality, scalability and performance.

BMP mock

The BMP mock is a stand-alone Java application purposed to simulate a BMP-enabled router(s) and peers. The simulator is capable to report dummy routes and statistics. This application is not part of the OpenDaylight Karaf distribution, however it can be downloaded from OpenDaylight's Nexus (use latest release version):

`https://nexus.opendaylight.org/content/repositories/opendaylight.release/org/opendaylight/bgpcep/bgp-bmp-mock`

Usage

The application can be run from command line:

```
java -jar bgp-bmp-mock-*-executable.jar
```

with optional input parameters:

```
--local_address <address> (optional, default 127.0.0.1)
   The IPv4 address where BMP mock is bind to.

--remote_address <IP_ADDRESS:PORT,...>, -ra <IP_ADDRESS:PORT,...>
   A list of IP addresses of BMP monitoring station, by default 127.0.0.1:12345

--passive (optional, not present by default)
   This flags enables passive mode for simulated routers.

--routers_count <0..N> (optional, default 1)
    An amount of BMP routers to be connected to the BMP monitoring station.

--peers_count <0..N> (optional, default 0)
   An amount of peers reported by each BMP router.

--pre_policy_routes <0..N> (optional, default 0)
   An amount of "pre-policy" simple IPv4 routes reported by each peer.

--post_policy_routes <0..N> (optional, default 0)
   An amount of "post-policy" simple IPv4 routes reported by each peer.

--log_level <FATAL|ERROR|INFO|DEBUG|TRACE> (optional, default INFO)
   Set logging level for BMP mock.
```

Troubleshooting

This section offers advices in a case OpenDaylight BMP plugin is not working as expected.

Contents

BMP is not working...

- First of all, ensure that all required features are installed, local monitoring station and monitored router/peers configuration is correct.

  To list all installed features in OpenDaylight use the following command at the Karaf console:

```
feature:list -i
```

- Check OpenDaylight Karaf logs:

  From Karaf console:

```
log:tail
```

  or open log file: `data/log/karaf.log`

  Possibly, a reason/hint for a cause of the problem can be found there.

- Try to minimize effect of other OpenDaylight features, when searching for a reason of the problem.
- Try to set DEBUG severity level for BMP logger via Karaf console commands, in order to collect more information:

```
log:set DEBUG org.opendaylight.protocol.bmp
```

Bug reporting

Before you report a bug, check BGPCEP Jira to ensure same/similar bug is not already filed there.

Write an e-mail to bgpcep-users@lists.opendaylight.org and provide following information:

1. State OpenDaylight version
2. Describe your use-case and provide as much details related to BMP as possible
3. Steps to reproduce
4. Attach Karaf log files, optionally packet captures, REST input/output

BIER User Guide

Overview

The technology of Bit Index Explicit Replication (BIER) specifies a new architecture for the forwarding of multicast data packets. It provides optimal forwarding of multicast data packets through a "multicast domain". However, it does not require the use of a protocol for explicitly building multicast distribution trees, and it does not require intermediate nodes to maintain any per-flow state. See specific in draft-ietf-bier-architecture-05 and related documents.

The BIER project provides functionality about BIER/BIER-TE topo-mamagement and BIER/BIER-TE channel-mamagement, and invoking south-bound-interface for device driver.

BIER User-Facing Features

- **odl-bier-all**

  - This feature contains all other features/bundles of BIER project. If you install it, it provides all functions that the BIER project can support.

- **odl-bier-models**
  - This feature contains all models of BIER project, such as ietf-bier, ietf-multicast-information and so on.
- **odl-bier-bierman**
  - This feature generates BIER's topology from network topology, and configuration of BIER, BIER-TE, etc.
- **odl-bier-channel**
  - This feature provides function about multicast flow information configuration and deployment in BIER domain.
- **odl-bier-service**
  - This feature provides function which processing the result of BIER bierman and BIER channel, and invoking south-bound-interface for driver.
- **odl-bier-adapter**
  - This feature provides adapter for different BIER south-bound NETCONF interfaces, so all BFRs in BIER domain with different NETCONF configuration interfaces and they can operate normally together.
- **odl-bier-driver**
  - This feature is south-bound NETCONF interface for BIER, it has implemented standard interface (ietf-bier). If your BFR's NETCONF interface is Non-standard, you should add your own interface for driver.
- **odl-te-pce**
  - This feature provides path computation function for BIER-TE.
- **odl-bier-app**
  - This feature provides the interface of BIER management, which contain BIER/BIER-TE manager, channel manager, topology manager.

How To Start

Preparing for Installation

1. Forwarding devices must support the BGP-LS protocol, and already be configured so that OpenDaylight can discover those devices.
2. Forwarding devices must support BIER configuration via NETCONF, which has a standard IETF YANG model.
3. The feature *odl-bier-app* or third-party App provides the northbound interface of BIER management for BIER controller.

Installation Feature

Run OpenDaylight and install BIER Service *odl-bier-all* as below:

```
feature:install odl-bier-all
```

For a more detailed overview of the BIER, see the *BIER Developer Guide*.

CAPWAP User Guide

This document describes how to use the Control And Provisioning of Wireless Access Points (CAPWAP) feature in OpenDaylight. This document contains configuration, administration, and management sections for the feature.

Overview

CAPWAP feature fills the gap OpenDaylight Controller has with respect to managing CAPWAP compliant wireless termination point (WTP) network devices present in enterprise networks. Intelligent applications (e.g. centralized firmware management, radio planning) can be developed by tapping into the WTP network device's operational states via REST APIs.

CAPWAP Architecture

The CAPWAP feature is implemented as an MD-SAL based provider module, which helps discover WTP devices and update their states in MD-SAL operational datastore.

Scope of CAPWAP Project

In this release, CAPWAP project aims to only detect the WTPs and store their basic attributes in the operational data store, which is accessible via REST and JAVA APIs.

Installing CAPWAP

To install CAPWAP, download OpenDaylight and use the Karaf console to install the following feature:

odl-capwap-ac-rest

Configuring CAPWAP

As of this release, there are no configuration requirements.

Administering or Managing CAPWAP

After installing the odl-capwap-ac-rest feature from the Karaf console, users can administer and manage CAPWAP from the APIDOCS explorer.

Go to http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/apidoc/explorer/index.html, sign in, and expand the capwap-impl panel. From there, users can execute various API calls.

Tutorials

Viewing Discovered WTPs

Overview

This tutorial can be used as a walk through to understand the steps for starting the CAPWAP feature, detecting CAPWAP WTPs, accessing the operational states of WTPs.

Prerequisites

It is assumed that user has access to at least one hardware/software based CAPWAP compliant WTP. These devices should be configured with OpenDaylight controller IP address as a CAPWAP Access Controller (AC) address. It is also assumed that WTPs and OpenDaylight controller share the same ethernet broadcast domain.

Instructions

1. Run the OpenDaylight distribution and install odl-capwap-ac-rest from the Karaf console.
2. Go to http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/apidoc/explorer/index.html
3. Expand capwap-impl
4. Click /operational/capwap-impl:capwap-ac-root/
5. Click "Try it out"
6. The above step should display list of WTPs discovered using ODL CAPWAP feature.

Cardinal: OpenDaylight Monitoring as a Service

This section describes how to use the Cardinal feature in OpenDaylight and contains configuration, administration, and management sections for the feature.

Overview

Cardinal (OpenDaylight Monitoring as a Service) enables OpenDaylight and the underlying software defined network to be remotely monitored by deployed Network Management Systems (NMS) or Analytics suite. In the Boron release, Cardinal will add:

1. OpenDaylight MIB.
2. Enable ODL diagnostics/monitoring to be exposed across SNMP (v2c, v3) and REST north-bound.
3. Extend ODL System health, Karaf parameter and feature info, ODL plugin scalability and network parameters.
4. Support autonomous notifications (SNMP Traps).

Cardinal Architecture

The Cardinal architecture can be found at the below link:

https://wiki.opendaylight.org/images/8/89/Cardinal-ODL_Monitoring_as_a_Service_V2.pdf

Configuring Cardinal feature

To start Cardinal feature, start karaf and type the following command:

```
feature:install odl-cardinal
```

After this Cardinal should be up and working with SNMP daemon running on port 161.

Tutorials

Below are tutorials for Cardinal.

Using Cardinal

These tutorials are intended for any user who wants to monitor three basic component in OpenDaylight

1. System Info in which controller is running.
2. Karaf Info
3. Project Specific Information (Openflow and Netconf devices).

Prerequisites

There is no as such specific prerequisite. Cardinal can work without installing any third party software. However If one wants to see the output of a snmpget/snmpwalk on the CLI prompt, than one can install the SNMP using the below link:

https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-an-snmp-daemon-and-client-on-ubuntu-14-04

Using the above command line utility one can get the same result as the cardinal APIs will give for the snmpget/snmpwalk request.

Target Environment

This tutorial is developed considering the following environment:

controller-Linux(Ubuntu 14.02).

Instructions

Install Cardinal feature

Open karaf and install the cardinal feature using the following command:

```
feature:install odl-cardinal
```

Please verify that SNMP daemon is up on port 161 using the following command on the terminal window of Linux machine:

```
netstat -anp | grep "161"
netstat -anp | grep "2001"
netstat -anp | grep "2003"
```

If the grep on the ‘‘snmpd‘‘ port is successful than SNMP daemon is up and working.

APIs Reference

Please see Developer guide for usage of Cardinal APIs.

CLI commands to do snmpget/walk

One can do snmpget/walk on the ODL-CARDINAL-MIB. Open the linux terminal and type the below command:

```
snmpget -v2c -c public localhost Oid_Of_the_mib_variable
```

Or

```
snmpget -v2c -c public localhost ODL-CARDINAL-MIB::mib_variable_name
```

For snmpwalk use the below command:

```
snmpwalk -v2c -c public localhost SNMPv2-SMI::experimental
```

For tabular data (netconf devices), snmpwalk use the below command:

```
snmpwalk -v2c -c public localhost:2001 SNMPv2-SMI::experimental
```

For tabular data (openflow devices), snmpwalk use the below command:

```
snmpwalk -v2c -c public localhost:2003 SNMPv2-SMI::experimental
```

Centinel User Guide

The Centinel project aims at providing a distributed, reliable framework for efficiently collecting, aggregating and sinking streaming data across Persistence DB and stream analyzers (example: Graylog, Elastic search, Spark, Hive etc.). This document contains configuration, administration, management, using sections for the feature.

Overview

In this release of Centinel, this framework enables SDN applications/services to receive events from multiple streaming sources (e.g., Syslog, Thrift, Avro, AMQP, Log4j, HTTP/REST) and execute actions like network configuration/batch processing/real-time analytics. It also provides a Log Service to assist operators running SDN ecosystem by installing the feature odl-centinel-all.

With the configurations development of "Log Service" and plug-in for log analyzer (e.g., Graylog) will take place. Log service will do processing of real time events coming from log analyzer. Additionally, stream collector (Flume and Sqoop based) that will collect logs from OpenDaylight and sink it to persistence service (integrated with TSDR). Also includes RESTCONF interface to inject events to north bound applications for real-time analytic/network configuration. Centinel User Interface (web interface) will be available to operators to enable rules/alerts/dashboard.

Centinel core features

The core features of the Centinel framework are:

Stream collector Collecting, aggregating and sinking streaming data

Log Service Listen log stream events coming from log analyzer

Log Service Enables user to configure rules (e.g., alerts, diagnostic, health, dashboard)

Log Service Performs event processing/analytics

User Interface Enable set-rule, search, visualize, alert, diagnostic, dashboard etc.

Adaptor Log analyzer plug-in to Graylog and a generic data-model to extend to other stream analyzers (e.g., Logstash)

REST Service Northbound APIs for Log Service and Steam collector framework

Leverages TSDR persistence service, data query, purging and elastic search

Centinel Architecture

The following wiki pages capture the Centinel Model/Architecture

1. https://wiki.opendaylight.org/view/Centinel:Main
2. https://wiki.opendaylight.org/view/Project_Proposals:Centinel
3. https://wiki.opendaylight.org/images/0/09/Centinel-08132015.pdf

Administering or Managing Centinel with default configuration

Prerequisites

1. Check whether Graylog is up and running and plugins deployed as mentioned in installation guide.
2. Check whether HBase is up and respective tables and column families as mentioned in installation guide are created.
3. Check if apache flume is up and running.
4. Check if apache drill is up and running.

Running Centinel

The following steps should be followed to bring up the controller:

1. Download the Centinel OpenDaylight distribution release from below link: http://www.opendaylight.org/software/downloads
2. Run Karaf of the distribution from bin folder

```
./karaf
```

3. Install the centinel features using below command:

```
feature:install odl-centinel-all
```

4. Give some time for the centinel to come up.

User Actions

1. **Log In:** User logs into the Centinel with required credentials using following URL: http://localhost:8181/index.html
2. **Create Rule:**
   (a) Select Centinel sub-tree present in left side and go to Rule tab.
   (b) Create Rule with title and description.

(c) Configure flow rule on the stream to filter the logs accordingly for, e.g., `bundle_name=org.opendaylight.openflow-plugin`

3. **Set Alarm Condition:** Configure alarm condition, e.g., message-count-rule such that if 10 messages comes on a stream (e.g., The OpenFlow Plugin) in last 1 minute with an alert is generated.
4. **Subscription:** User can subscribe to the rule and alarm condition by entering the http details or email-id in subscription textfield by clicking on the subscribe button.
5. **Create Dashboard:** Configure dashboard for stream and alert widgets. Alarm and Stream count will be updated in corresponding widget in Dashboard.
6. **Event Tab:** Intercepted Logs, Alarms and Raw Logs in Event Tab will be displayed by selecting the appropriate radio button. User can also filter the searched data using SQL query in the search box.

Data Export/Import User Guide

Overview

The Data Export/Import is known as "daexim" (pronounced 'deck-sim') for short. The intended audience are administrators responsible for operations of OpenDaylight.

Data Export/Import provides an API (via RPCs) to request the bulk transfer of OpenDaylight system data between its internal data stores and the local file system. This can be used for scheduling data exports, checking the status of data being exported, canceling data export jobs, importing data from files in the file systems, and checking the import status.

Such export and import of data can be used during system upgrade, enabling the development of administrative procedures that make reconfigurations of the base system without concern of internal data loss.

Data Export produces a models declaration file and one or more data files. The models declaration file records exactly which YANG models were loaded (by module name, revision date and namespace). The data file(s) contain data store data as per the draft-ietf-netmod-yang-json RFC.

Data Import takes a models declaration file and zero or more data files. The models declaration file is used to check that the listed models are loaded before importing any data. Data is imported into each data store in turn with one transaction executed for each data store, irrespective of the number of files for that data store, as inter-module data dependencies may exist. Existing data store data may be cleared before importing.

Data Export Import Architecture

The daexim feature is a single feature. This feature leverages the existing infrastructure provided by MD-SAL and yangtools.

Installing the Feature

To install the feature perform the following steps.

```
karaf > feature:install odl-daexim-all
```

The interactions with this feature are via Restconf RPCs. The details are provided in the *Tutorials*.

Tutorials

The following tutorials provide examples of REST API that are supported by the Data Export/Import feature. As for all ODL RESTCONF calls, the following are the common setting for REST calls:

- Headers: * Content-Type: application/json * Accept: application/json * Authentication: admin:admin
- Method: HTTP POST
- <controller-ip> : Host (or IP) where OpenDaylight controller is running, e.g. localhost
- <restconf-port> : TCP port where RESTCONF has been configured to listen, e.g. 8181 by default

The files created by export are placed in a subdirectory called *daexim/* in the installation directory of OpenDaylight. Similarly files to import must be placed in this *daexim/* subdirectory.

Scheduling Export

The **schedule-export** RPC exports the data at a specific time in the future. The *run-at* time may be specified as an absolute UTC time or a relative offset from the server clock. Attempts to schedule an export in the past times are rejected. Each file has a JSON-encoded object that contains module data from the corresponding data store. Module data is not included in the object for any module identified in the exclusion list. Each file contains at least one empty JSON object.

URL: http://<controller-ip>:<restconf-port>/restconf/operations/data-export-import:schedule-export

Payload:

```
{
   "input": {
     "data-export-import:run-at": 500
   }
}
```

Checking Export Status

The **status-export** RPC checks the status of the exported data. If the status has the value of *initial*, an export has not been scheduled. If the status has the value of *scheduled*, *run-at* indicates the time at which the next export runs. If the status has the value of *in-progress*, *run-at* indicates the time at which the running export was scheduled to start. A status of *tasks* indicates activities that are scheduled and currently being performed. The *tasks* status serves as an indicator of progress and success of the activity. If the status has any other value, *run-at* indicates the time at which the last export was scheduled to start; and *tasks* indicates the activities that were undertaken. If the status for any node has failed, the corresponding reason for failure is listed.

URL: http://<controller-ip>:<restconf-port>/restconf/operations/data-export-import:status-export

Payload: No payload

Canceling Scheduled Export

The **cancel-export** RPC cancels an already scheduled data export job. To cancel the export, the server stops the tasks that are running (where possible, immediately), clears any scheduled export time value, and releases the associated resources. This RPC may be called at any time, whether an export is in progress, scheduled or not yet scheduled. The returned result is *True* when the server has successfully cleared tasks, the state, and resources. The status is *False* on

failure to do so. Note that if no export is scheduled or running, there is no tasks for the server to clear. Therefore, the return result is *True* because the server cannot fail.

URL: http://<controller-ip>:<restconf-port>/restconf/operations/data-export-import:cancel-export

Payload: No payload

Import from a file

The **immediate-import** RPC imports data from files already present in the file system.

URL: http://<controller-ip>:<restconf-port>/restconf/operations/data-export-import:immediate-import

Payload:

```
{
   "input" : {
     "check-models" : true,
     "clear-stores" : "all"
   }
}
```

Status of Import

The **status-import** RPC checks the last import status. If the status has the value of *initial*, an import has not taken place. For all other values of status, *imported-at* indicates the time at which the restoration has taken place. List nodes hold status about the restoration for each node.

URL: http://<controller-ip>:<restconf-port>/restconf/operations/data-export-import:status-import

Payload: No payload

DIDM User Guide

Overview

The Device Identification and Driver Management (DIDM) project addresses the need to provide device-specific functionality. Device-specific functionality is code that performs a feature, and the code is knowledgeable of the capability and limitations of the device. For example, configuring VLANs and adjusting FlowMods are features, and there may be different implementations for different device types. Device-specific functionality is implemented as Device Drivers. Device Drivers need to be associated with the devices they can be used with. To determine this association requires the ability to identify the device type.

DIDM Architecture

The DIDM project creates the infrastructure to support the following functions:

- **Discovery** - Determination that a device exists in the controller management domain and connectivity to the device can be established. For devices that support the OpenFlow protocol, the existing discovery mechanism in OpenDaylight suffices. Devices that do not support OpenFlow will be discovered through manual means such as the operator entering device information via GUI or REST API.
- **Identification** – Determination of the device type.
- **Driver Registration** – Registration of Device Drivers as routed RPCs.

- **Synchronization** – Collection of device information, device configuration, and link (connection) information.
- **Data Models for Common Features** – Data models will be defined to perform common features such as VLAN configuration. For example, applications can configure a VLAN by writing the VLAN data to the data store as specified by the common data model.
- **RPCs for Common Features** – Configuring VLANs and adjusting FlowMods are example of features. RPCs will be defined that specify the APIs for these features. Drivers implement features for specific devices and support the APIs defined by the RPCs. There may be different Driver implementations for different device types.

Atrium Support

The Atrium implements an open source router that speaks BGP to other routers, and forwards packets received on one port/vlan to another, based on the next-hop learnt via BGP peering. A BGP peering application for the Open Daylight Controller and a new model for flow objective drivers for switches integrated with the Open Daylight Atrium distribution was developed for this project. The implementation has the same level of feature partly that was introduced by the Atrium 2015/A distribution on the ONOS controller. An overview of the architecture is available at here (https://github.com/onfsdn/atrium-docs/wiki/ODL-Based-Atrium-Router-16A).

Atrium stack is implemented in OpenDaylight using Atrium and DIDM project. Atrium project provides the application implementation for BGP peering and the DIDM project provides implementation for FlowObjectives. FlowObjective provides an abstraction layer and present the pipeline agnostic api to application to consume.

FlowObjective

Flow Objectives describe an SDN application's objective (or intention) behind a flow it is sending to a device.

Application communicates the flow installation requirement using Flow Objectives. DIDM drivers translates the Flow Objectives to device specific flows as per the device pipeline.

There are three FlowObjectives (already implemented in ONOS controller) :

- Filtering Objective
- Next Objective
- Forwarding Objective

Installing DIDM

To install DIDM, download OpenDaylight and use the Karaf console to install the following features:

- odl-openflowplugin-all
- odl-didm-all

odl-didm-all installs the following required features:

- odl-didm-ovs-all
- odl-didm-ovs-impl
- odl-didm-util
- odl-didm-identification
- odl-didm-drivers

- odl-didm-hp-all

Configuring DIDM

This section shows an example configuration steps for installing a driver (HP 3800 OpenFlow switch driver).

Install DIDM features:

```
feature:install odl-didm-identification-api
feature:install odl-didm-drivers
```

In order to identify the device, device driver needs to be installed first. Identification Manager will be notified when a new device connects to the controller.

Install HP driver

feature:install odl-didm-hp-all installs the following features

- odl-didm-util
- odl-didm-identification
- odl-didm-drivers
- odl-didm-hp-all
- odl-didm-hp-impl

Now at this point, the driver has written all of the identification information in to the MD-SAL datastore. The identification manager should have that information so that it can try to identify the HP 3800 device when it connects to the controller.

Configure the switch and connect it to the controller from the switch CLI.

Run REST GET command to verify the device details:

http://<CONTROLLER-IP:8181>/restconf/operational/opendaylight-inventory:nodes

Run REST adjust-flow command to adjust flows and push to the device

Flow mod driver for HP 3800 device

This driver adjusts the flows and push the same to the device. This API takes the flow to be adjusted as input and displays the adjusted flow as output in the REST output container. Here is the REST API to adjust and push flows to HP 3800 device:

http://<CONTROLLER-IP:8181>/restconf/operations/openflow-feature:adjust-flow

FlowObjectives API

FlowObjective presents the OpenFlow pipeline agnostic API to Application to consume. Application communicate their intent behind installation of flow to Drivers using the FlowObjective. Driver translates the FlowObjective in device specific flows and uses the OpenFlowPlugin to install the flows to the device.

Filter Objective

http://<CONTROLLER-IP>:8181/restconf/operations/atrium-flow-objective:filter

Next Objective

http://<CONTROLLER-IP>:8181/restconf/operations/atrium-flow-objective:next

Forward Objective

http://<CONTROLLER-IP>:8181/restconf/operations/atrium-flow-objective:forward

Distribution Version reporting

Overview

This section provides an overview of **odl-distribution-version** feature.

A remote user of OpenDaylight usually has access to RESTCONF and NETCONF northbound interfaces, but does not have access to the system OpenDaylight is running on. OpenDaylight has released multiple versions including Service Releases, and there are incompatible changes between them. In order to know which YANG modules to use, which bugs to expect and which workarounds to apply, such user would need to know the exact version of at least one OpenDaylight component.

There are indirect ways to deduce such version, but the direct way is enabled by odl-distribution-version feature. Administrator can specify version strings, which would be available to users via NETCONF, or via RESTCONF if OpenDaylight is configured to initiate NETCONF connection to its config subsystem northbound interface.

By default, users have write access to config subsystem, so they can add, modify or delete any version strings present there. Admins can only influence whether the feature is installed, and initial values.

Config subsystem is local only, not cluster aware, so each member reports versions independently. This is suitable for heterogeneous clusters.

Default config file

Initial version values are set via config file `odl-version.xml` which is created in `$KARAF_HOME/etc/opendaylight/karaf/` upon installation of `odl-distribution-version` feature. If admin wants to use different content, the file with desired content has to be created there before feature installation happens.

By default, the config file defines two config modules, named `odl-distribution-version` and `odl-odlparent-version`.

RESTCONF usage

Opendaylight config subsystem NETCONF northbound is not made available just by installing `odl-distribution-version`, but most other feature installations would enable it. RESTCONF interfaces are enabled by installing `odl-restconf` feature, but that do not allow access to config subsystem by itself.

On single node deployments, installation of `odl-netconf-connector-ssh` is recommended, which would configure `controller-config` device and its MD-SAL mount point.

For cluster deployments, installing `odl-netconf-clustered-topology` is recommended. See documentation for clustering on how to create similar devices for each member, as `controller-config` name is not unique in that context.

Assuming single node deployment and user located on the same system, here is an example `curl` command accessing `odl-odlparent-version` config module:

```
curl 127.0.0.1:8181/restconf/config/network-topology:network-topology/topology/
↪topology-netconf/node/controller-config/yang-ext:mount/config:modules/module/odl-
↪distribution-version:odl-version/odl-odlparent-version
```

eman User Guide

Overview

The OpenDaylight Energy Management (eman) plugin implements an abstract Information Model that describes energy measurement and control features that may be supported by a variety of device types. The eman plugin may support a number of southbound interfaces to accommodate a set of protocols, including but not limited to SNMP, NETCONF, IPDR. The plugin presents a northbound REST API. This framework enables any number of applications to interoperate with any number of devices in order to measure and optimize energy usage. The Information Model will be inherited from the SCTE 216 standard – Adaptive Power Systems Interface Specification (APSIS), which in turn inherits definitions within the IETF eman document set.

This documentation is directed to those operating the features such as network administrator, cloud administrator, network engineer, or system administrators.

eman is composed of 3 Karaf features:

- `eman` incudes the YANG model and its implementation
- `eman-api` adds support for REST
- `eman-ui` adds support for DLUX.

Developers will typically interface with `eman-api`.

eman Architecture

`eman` defines a YANG model that represents the IETF energy management Information Model, and includes RPCs. The implementation of the model currently supports an SNMP 'binding' via interfacing with the OpenDaylight SNMP module. In the future, other Southbound protocols may be supported.

Developers my use the `eman-api` feature to read and write energy related data and commands to devices that support the IETF eman MIBS.

Besides a set of common controller features eman depends upon the OpenDaylight SNMP features to be installed.

Configuring eman

eman relies upon the presence of SNMP agents.

The following describes a way to install and configure an SNMP simulator on localhost.

on macOS, open terminal

1. Install snmpsim.:

```
$ sudo easy_install -n snmpsim
```

2. configure filesystem:

```
mkdir ~/.snmpsim, then mkdir ~/.snmpsim/data/
```

3. Install moak data. This file is used by pysnmp to provide mock data for an APSIS agent:

```
copy eman/sample_code/data/energy-object.snmprec to ~/.snmpsim/data/.
```

4. launch snmp simulator:

```
$ sudo snmpsimd.py --agent-udpv4-endpoint=127.0.0.1:161
    --process-group=<your group> --process-user=<your user>
```

5. VerifyOpen another terminal window and execute:

```
$ snmpget -v2c -c energy-object localhost:161 1.3.6.1.2.1.229.0.1.0.
```

The result should be '1', as defined in your snmprec file

Note: group and user are settings within our local OS. For Mac users, look at settings/users and groups. If port 161 is not available, use another unprivileged port such as 1161.

Note: snmpget queries snmpsimd to return a value for the OID 1.3.6.1.2.1.229.0.1.0. According to the energy-object.snmprec file, the value for that OID is '1'. Try other OIDs, or edit the snmprec file to see your results

Future release may include more flexible and robust means to simulate a network of energy aware SNMP agents.

Typically, a process may periodically poll a device to acquire power measurements and repose them into MD-SAL. Subsequently, process may read a history of power measurements from MD-SAL via the eman operational API.

Fabric As A Service

This document describes, from a user's or application's perspective, how to use the Fabric As A Service (FaaS) feature in OpenDaylight. This document contains configuration, administration, and management sections for the FaaS feature.

Overview

Currently network applications and network administrators mostly rely on lower level interfaces such as CLI, SNMP, OVSDB, NETCONF or OpenFlow to directly configure individual device for network service provisioning. In general, those interfaces are:

- Technology oriented, not application oriented.
- Vendor specific
- Individual device oriented, not network oriented.
- Not declarative, complicated and procedure oriented.

To address the gap between application needs and network interface, there are a few application centric language proposed in OpenDaylight including Group Based Policy (GBP), Network Intent Composition (NIC), and NEtwork MOdeling (NEMO) trying to replace traditional southbound interface to application. Those languages are top-down abstractions and model application requirements in a more application-oriented way.

After being involved with GBP development for a while, we feel the top down model still has a quite gap between the model and the underneath network since the existing interfaces to network devices lack of abstraction makes it very hard to map high level abstractions to the physical network. Often the applications built with these low level interfaces are coupled tightly with underneath technology and make the application's architecture monolithic, error prone and hard to maintain.

We think a bottom-up abstraction of network can simplify, reduce the gap, and make it easy to implement the application centric model. Moreover in some uses cases, an interface with network service oriented are still desired for example from network monitoring/troubleshooting perspective. That's where the Fabric as a Service comes in.

FaaS Architecture

Fabric and its controller (Fabric Controller) The Fabric object provides an abstraction of a homogeneous network or portion of the network and also has a built in Fabric controller which provides management plane and control plane for the fabric. The fabric controller implements the services required in Fabric Service and monitor and control the fabric operation.

Fabric Manager Fabric Manager manages all the fabric objects. also Fabric manager acts as a Unified Fabric Controller which provides inter-connect fabric control and configuration Also Fabric Manager is FaaS API service via Which FaaS user level logical network API (the top level API as mentioned previously) exposed and implemented.

FaaS render for GBP (Group Based Policy) FaaS render for GBP is an application of FaaS and provides the rendering service between GBP model and logical network model provided by Fabric Manager.

FaaS RESTCONF API

FaaS Provides two layers API:

- The top layer is a **user level logical network** API which defines CRUD services operating on the following abstracted constructs:
  - vcontainer - virtual container
  - logical Port - a input/out/access point of a logical device
  - logical link - connects ports
  - logical switch - a layer 2 forwarding device
  - logical router - a layer 3 forwarding device
  - Subnet
  - Rule/ACL
  - End Points Registration
  - End Points Attachment

Through these constructs, a logical network can be described without users knowing too much details about the physical network device and technology, i.e. users' network services is decoupled from underneath physical infrastructure. This decoupling brought the benefit that the users defined service is not locked in with any specific technology or

physical devices. FaaS maps the logical network to the physical network configuration automatically which in maximum eliminates the manual provisioning work. As a result, human error is avoided and OPEX for network operators is massively reduced. Moreover migration from one technology to another is much easier to do and transparent to users' services.

- The second layer defines an abstraction layer called **Fabric** API. The idea is to abstract network into a topology formed by a collections of fabric objects other than varies of physical devices.Each Fabric object provides a collection of unified services. The top level API enables application developers or users to write applications to map high level model such as GBP, Intent etc... into a logical network model, while the lower level gives the application more control to individual fabric object level. More importantly the Fabric API is more like SPI (Service Provider API) a fabric provider or vendor can implement the SPI based on its own Fabric technique such as TRILL, SPB etc ...

This document is focused on the top layer API. For how to use second level API operation, please refer to FaaS developer guide for more details.

Note: that for any JSON data or link not described here, please go to http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/apidoc/explorer/index.htm for details or clarification.

Resource Management API

The FaaS Resource Management API provides services to allocate and reclaim the network resources provided by Fabric object. Those APIs can be accessed via RESTCONF rendered from YANG in MD-SAL.

- Name: Create virtual container
  - virtual container is an collections of resources allocated to a tenant, for example, a list of physical ports, bandwidth or L2 network or logical routers quantity. etc...
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/vcontainer-topology:create-vcontainer
  - Description: create a given virtual container.
- Name: assign or remove fabric resource to a virtual container
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/vc-ld-node:add-vfabric-to-ld-node
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/vc-ld-node:rm-vfabric-to-ld-node
- Name: assign or remove appliance to a virtual container
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/vc-ld-node:add-appliance-to-ld-node
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/vc-ld-node:rm-appliance-to-ld-node
- Name: create or remove a child container
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/vc-ld-node:create-child-ld-node
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/vc-ld-node:rm-child-ld-node
- RESTCONF path for Virtual Container Datastore query (note: vcontainer-id equals tenantID for now):

  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/config/network-topology/topology/\protect\T1\textbraceleftvcontainer-id\protect\T1\textbraceright/vc-topology-attributes/
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/config/network-topology/topology/\protect\T1\textbraceleftvcontainer-id\protect\T1\textbraceright/node/\protect\T1\textbraceleftnet-node-id\protect\T1\textbraceright/vc-net-node-attributes
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/config/network-topology/topology/\protect\T1\textbraceleftvcontainer-id\protect\T1\textbraceright/node/\protect\T1\textbraceleftld-node-id\protect\T1\textbraceright/vc-ld-node-attributes

Installing Fabric As A Service

To install FaaS, download OpenDaylight and use the Karaf console to install the following feature: **odl-restconf odl-faas-all odl-groupbasedpolicy-faas** (if needs to use FaaS to render GBP)

Configuring FaaS

This section gives details about the configuration settings for various components in FaaS.

The FaaS configuration files for the Karaf distribution are located in distribution/karaf/target/assembly/etc/faas

- akka.conf
  - This file contains configuration related to clustering. Potential configuration properties can be found on the akka website at http://doc.akka.io
- fabric-factory.xml
- vxlan-fabric.xml
- vxlan-fabric-ovs-adapter.xml
  - Those 3 files are used to initialize fabric module and located under distribution/karaf/target/assembly/etc/opendaylight/karaf

Managing FaaS

Start opendaylight karaf distribution

- *>bin/karaf* Then From karaf console,Install features in the following order:
- *>feature:Install odl-restconf*
- *>feature:install odl-faas-all*
- *>feature install odl-groupbasedpolicy-faas*

After installing features above, users can manage Fabric resource and FaaS logical network channels from the API-DOCS explorer via RESTCONF

Go to http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/apidoc/explorer/index.html, sign in, and expand the FaaS panel. From there, users can execute various API calls to test their FaaS deployment such as create virtual container, create fabric, delete fabric, create/edit logical network elements.

Tutorials

Below are tutorials for 4 major use cases.

1. to create and provision a fabric
2. to allocate resource from the fabric to a tenant
3. to define a logical network for a tenant. Currently there are two ways to create a logical network
   (a) Create a GBP (Group Based Policy) profile for a tenant and then convert it to a logical network via GBP FaaS render Or
   (b) Manually create a logical network via RESTCONF APIs.
4. to attach or detach an Endpoint to a logical switch or logical router

Create a fabric

Overview

This tutorial walks users through the process of create a Fabric object

Prerequisites

A set of virtual switches (OVS) have to be registered or discovered by ODL. Mininet is recommended to create a OVS network. After an OVS network is created, set up the controller IP pointing to ODL IP address in each of the OVS. From ODL, a physical topology can be viewed via ODL DLUX UI or retrieved via RESTCONF API.

Instructions

- Run the OpenDaylight distribution and install odl-faas-all from the Karaf console.
- Go to http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/apidoc/explorer/index.html
- Get the network topology after OVS switches are registered in the controller
- Determine the nodes and links to be included in the to-be-defined Fabric object.
- Execute create-fabric RESTCONF API with the corresponding JSON data as required.

Create virtual container for a tenant

The purpose of this tutorial is to allocate network resources to a tenant

Overview

This tutorial walks users through the process of create a Fabric

Prerequisites

1 or more fabric objects have been created.

Instructions

- Run the OpenDaylight karaf distribution and install odl-faas-all feature from the Karaf console. >feature:install odl-rest-conf odl-faas-all odl-mdsal-apidoc
- Go to http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/apidoc/explorer/index.html
- Execute create-vcontainer with the following restconf API with corresponding JSON data > http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/vcontainer-topology:create-vcontainer

After a virtual container is created, fabric resource and appliance resource can be assigned to the container object via the following RESTConf API.

- http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/vc-ld-node:add-vfabric-to-ld-node
- http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/vc-ld-node:add-appliance-to-ld-node

Create a logical network

Overview

This tutorial walks users through the process of create a logical network for a tenant

Prerequisites

a virtual container has been created and assigned to the tenant

Instructions

Currently there are two ways to create a logical network.

- Option 1 is to use logical network RESTConf REST API and directly create individual network elements and connect them into a network
- Option 2 is to define a GBP model and FaaS can map GBP model automatically into a logical network. Notes that for option 2, if the generated network requires some modification, we recommend modify the GBP model rather than change the network directly due to there is no synchronization from network back to GBP model in current release.

Manual Provisioning

To create a logical switch

- http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/configuration/faas-logical-networks:tenant-logical-networks:logical-switches:logical-switches To create a logical router
- http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/configuration/faas-logical-networks:tenant-logical-networks:logical-routers:logical-routers To attach a logical switch to a router

  - Step 1: updating/adding a port A on the logical switch http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/configuration/faas-logical-networks:tenant-logical-networks:logical-switches:logical-switches
  - Step 2: updating/adding a port B on the logical router http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/configuration/faas-logical-networks:tenant-logical-networks:logical-routers:logical-routers
  - Step 3; create a link between the port A and B http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/configuration/faas-logical-networks:tenant-logical-networks:logical-edges:logical-edges

- To add security policies (ACL or SFC) on a port http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/configuration/faas-logical-networks:tenant-logical-networks:faas-security-rules
- To query the logical network just created http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/configuration/faas-logical-networks:tenant-logical-networks

Provision via GBP FaaS Render

- Run the OpenDaylight distribution and install odl-faas-all and GBP faas render feature from the Karaf console.
- Go to http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/apidoc/explorer/index.html
- Execute "create GBP model" via GBP REST API. The GBP model then can be automatically mapped into a logical network.

Attach/detach an end point to a logical device

Overview

This tutorial walks users through the process of registering an End Point to a logical device either logical switch or router. The purpose of this API is to inform the FaaS where an endpoint physically attach. The location information consists of the binding information between physical port identifier and logical port information. The logical port is indicated by the endpoint either Layer 2 attribute(MAC address) or Layer 3 attribute (IP address) and logical network ID (VLAN ID). The logical network ID is indirectly indicated the tenant ID since it is mutual exclusive resource allocated to a tenant.

Prerequisites

The logical switch to which those end points are attached has to be created beforehand. and the identifier of the logical switch is required for the following RESTCONF calls.

Instructions

- Run the OpenDaylight distribution and install odl-faas-all from the Karaf console.
- Go to http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/apidoc/explorer/index.html

- Execute "attach end point " with the following RESTCONF API and corresponding JSON data: http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/configuration/faas-logical-networks:tenant-logical-networks:faas-endpoints-locations

Genius User Guide

Overview

The Genius project provides generic network interfaces, utilities and services. Any OpenDaylight application can use these to achieve interference-free co-existence with other applications using Genius.

Modules and Interfaces

In the first phase delivered in OpenDaylight Boron release, Genius provides following modules—

- Modules providing a common view of network interfaces for different services
  - **Interface (logical port) Manager**
    - *Allows bindings/registration of multiple services to logical ports/interfaces*
    - *Ability to plug in different types of southbound protocol renderers*
  - **Overlay Tunnel Manager**
    - *Creates and maintains overlay tunnels between configured Tunnel Endpoints (TEPs)*
- Modules providing commonly used functions as shared services to avoid duplication of code and waste of resources
  - **Liveness Monitor**
    - *Provides tunnel/nexthop liveness monitoring services*
  - **ID Manager**
    - *Generates persistent unique integer IDs*
  - **MD-SAL Utils**
    - *Provides common generic APIs for interaction with MD-SAL*

Interface Manager Operations

Creating interfaces

The YANG file Data Model odl-interface.yang contains the interface configuration data-model.

You can create interfaces at the MD-SAL Data Node Path **/config/if:interfaces/interface**, with the following attributes—

Common attributes

- **name**—unique interface name, can be any unique string (e.g., UUID string)
- **type**—interface type, currently supported *iana-if-type:l2vlan and iana-if-type:tunnel*
- **enabled**—admin status, possible values *true* or *false*
- **parent-refs** : used to specify references to parent interface/port feeding to this interface

- datapath-node-identifier—identifier for a fixed/physical dataplane node, can be physical switch identifier
- parent-interface—can be a physical switch port (in conjunction of above), virtual switch port (e.g., neutron port) or another interface
- list node-identifier—identifier of the dependant underlying configuration protocol
  - *topology-id*—can be ovsdb configuration protocol
  - *node-id*—can be hwvtep node-id

Type specific attributes

- when type = l2vlan
  - **vlan-id**—VLAN id for trunk-member l2vlan interfaces
  - **l2vlan-mode**—currently supported ones are *transparent*, *trunk* or *trunk-member*
- when type = stacked_vlan (Not supported yet)
  - **stacked-vlan-id**—VLAN-Id for additional/second VLAN tag
- when type = tunnel
  - **tunnel-interface-type**—tunnel type, currently supported ones are:
    - tunnel-type-vxlan
    - tunnel-type-gre
    - tunnel-type-mpls-over-gre
  - **tunnel-source**—tunnel source IP address
  - **tunnel-destination**—tunnel destination IP address
  - **tunnel-gateway**—gateway IP address
  - **monitor-enabled**—tunnel monitoring enable control
  - **monitor-interval**—tunnel monitoring interval in millisiconds
- when type = mpls (Not supported yet)
  - **list labelStack**—list of lables
  - **num-labels**—number of lables configured

Supported REST calls are **GET, PUT, DELETE, POST**

Creating L2 port interfaces

Interfaces on normal L2 ports (e.g. Neutron tap ports) are created with type *l2vlan* and *l2vlan-mode* as *transparent*. This type of interface classifies packets passing through a particular L2 (OpenFlow) port. In dataplane, packets belonging to this interface are classified by matching in-port against the of-port-id assigned to the base port as specified in parent-interface.

URL: /restconf/config/ietf-interfaces:interfaces

Sample JSON data

```
"interfaces": {
    "interface": [
        {
            "name": "4158408c-942b-487c-9a03-0b603c39d3dd",
```

```
            "type": "iana-if-type:l2vlan",                          <--- interface type
↪'l2vlan' for normal L2 port
            "odl-interface:l2vlan-mode": "transparent",             <--- 'transparent'␣
↪VLAN port mode allows any (tagged, untagged) ethernet packet
            "odl-interface:parent-interface": "tap4158408c-94",     <--- port-name as it␣
↪appears on southbound interface
            "enabled": true
        }
    ]
}
```

Creating VLAN interfaces

A VLAN interface is created as a *l2vlan* interface in *trunk-member* mode, by configuring a VLAN-Id and a particular L2 (vlan trunk) interface. Parent VLAN trunk interface is created in the same way as the *transparent* interface as specified above. A *trunk-member* interface defines a flow on a particular L2 port and having a particular VLAN tag. On ingress, after classification the VLAN tag is popped out and corresponding unique dataplane-id is associated with the packet, before delivering the packet to service processing. When a service module delivers the packet to this interface for egress, it pushes corresponding VLAN tag and sends the packet out of the parent L2 port.

URL: /restconf/config/ietf-interfaces:interfaces

Sample JSON data

```
"interfaces": {
    "interface": [
        {
            "name": "4158408c-942b-487c-9a03-0b603c39d3dd:100",
            "type": "iana-if-type:l2vlan",
            "odl-interface:l2vlan-mode": "trunk-member",        <--- for 'trunk-member
↪', flow is classified with particular vlan-id on an l2 port
            "odl-interface:parent-interface": "4158408c-942b-487c-9a03-0b603c39d3dd",␣
↪ <--- Parent 'trunk' iterface name
            "odl-interface:vlan-id": "100",
            "enabled": true
        }
    ]
}
```

Creating Overlay Tunnel Interfaces

An overlay tunnel interface is created with type *tunnel* and particular *tunnel-interface-type*. Tunnel interfaces are created on a particular data plane node (virtual switches) with a pair of (local, remote) IP addresses. Currently supported tunnel interface types are VxLAN, GRE and MPLSoverGRE.

URL: /restconf/config/ietf-interfaces:interfaces

Sample JSON data

```
"interfaces": {
    "interface": [
        {
            "name": "MGRE_TUNNEL:1",
            "type": "iana-if-type:tunnel",
            "odl-interface:tunnel-interface-type": "odl-interface:tunnel-type-mpls-
↪over-gre",
```

```
            "odl-interface:datapath-node-identifier": 156613701272907,
            "odl-interface:tunnel-source": "11.0.0.43",
            "odl-interface:tunnel-destination": "11.0.0.66",
            "odl-interface:monitor-enabled": false,
            "odl-interface:monitor-interval": 10000,
            "enabled": true
        }
    ]
}
```

Binding services on interface

The YANG file odl-interface-service-bindings.yang contains the service binding configuration data model.

An application can bind services to a particular interface by configuring MD-SAL data node at path /config/interface-service-binding. Binding services on interface allows particular service to pull traffic arriving on that interface depending upon the service priority. Service modules can specify openflow-rules to be applied on the packet belonging to the interface. Usually these rules include sending the packet to specific service table/pipeline. Service modules are responsible for sending the packet back (if not consumed) to service dispatcher table, for next service to process the packet.

URL:/restconf/config/interface-service-bindings:service-bindings/

Sample JSON data

```
"service-bindings": {
  "services-info": [
    {
      "interface-name": "4152de47-29eb-4e95-8727-2939ac03ef84",
      "bound-services": [
        {
          "service-name": "ELAN",
          "service-type": "interface-service-bindings:service-type-flow-based"
          "service-priority": 3,
          "flow-priority": 5,
          "flow-cookie": 134479872,
          "instruction": [
            {
              "order": 2,
              "go-to-table": {
                "table_id": 50
              }
            },
            {
              "order": 1,
              "write-metadata": {
                "metadata": 83953188864,
                "metadata-mask": 1099494850560
              }
            }
          ],
        },
        {
         "service-name": "L3VPN",
         "service-type": "interface-service-bindings:service-type-flow-based"
         "service-priority": 2,
         "flow-priority": 10,
```

```
          "flow-cookie": 134217729,
          "instruction": [
             {
               "order": 2,
               "go-to-table": {
                 "table_id": 21
               }
             },
             {
               "order": 1,
               "write-metadata": {
                 "metadata": 100,
                 "metadata-mask": 4294967295
               }
             }
           ],
         }
       ]
     }
  ]
}
```

Interface Manager RPCs

In addition to the above defined configuration interfaces, Interface Manager also provides several RPCs to access interface operational data and other helpful information. Interface Manger RPCs are defined in odl-interface-rpc.yang

The following RPCs are available—

get-dpid-from-interface

This RPC is used to retrieve dpid/switch hosting the root port from given interface name.

```
rpc get-dpid-from-interface {
    description "used to retrieve dpid from interface name";
    input {
        leaf intf-name {
            type string;
        }
    }
    output {
        leaf dpid {
            type uint64;
        }
    }
}
```

get-port-from-interface

This RPC is used to retrieve south bound port attributes from the interface name.

```
rpc get-port-from-interface {
    description "used to retrieve south bound port attributes from the interface name
↪";
```

```
    input {
        leaf intf-name {
            type string;
        }
    }
    output {
        leaf dpid {
            type uint64;
        }
        leaf portno {
            type uint32;
        }
        leaf portname {
            type string;
        }
    }
}
```

get-egress-actions-for-interface

This RPC is used to retrieve group actions to use from interface name.

```
rpc get-egress-actions-for-interface {
    description "used to retrieve group actions to use from interface name";
    input {
        leaf intf-name {
            type string;
            mandatory true;
        }
        leaf tunnel-key {
            description "It can be VNI for VxLAN tunnel ifaces, Gre Key for GRE␣
↪tunnels, etc.";
            type uint32;
            mandatory false;
        }
    }
    output {
        uses action:action-list;
    }
}
```

get-egress-instructions-for-interface

This RPC is used to retrieve flow instructions to use from interface name.

```
rpc get-egress-instructions-for-interface {
    description "used to retrieve flow instructions to use from interface name";
    input {
        leaf intf-name {
            type string;
            mandatory true;
        }
        leaf tunnel-key {
            description "It can be VNI for VxLAN tunnel ifaces, Gre Key for GRE␣
↪tunnels, etc.";
```

```
            type uint32;
            mandatory false;
        }
    }
    output {
        uses offlow:instruction-list;
    }
}
```

get-endpoint-ip-for-dpn

This RPC is used to get the local ip of the tunnel/trunk interface on a particular DPN (Data Plane Node).

```
rpc get-endpoint-ip-for-dpn {
    description "to get the local ip of the tunnel/trunk interface";
    input {
        leaf dpid {
            type uint64;
        }
    }
    output {
        leaf-list local-ips {
            type inet:ip-address;
        }
    }
}
```

get-interface-type

This RPC is used to get the type of the interface (vlan/vxlan or gre).

```
rpc get-interface-type {
description "to get the type of the interface (vlan/vxlan or gre)";
    input {
        leaf intf-name {
            type string;
        }
    }
    output {
        leaf interface-type {
            type identityref {
                base if:interface-type;
            }
        }
    }
}
```

get-tunnel-type

This RPC is used to get the type of the tunnel interface(vxlan or gre).

```
rpc get-tunnel-type {
description "to get the type of the tunnel interface (vxlan or gre)";
    input {
        leaf intf-name {
            type string;
        }
    }
    output {
        leaf tunnel-type {
            type identityref {
                base odlif:tunnel-type-base;
            }
        }
    }
}
```

get-nodeconnector-id-from-interface

This RPC is used to get node-connector-id associated with an interface.

```
rpc get-nodeconnector-id-from-interface {
description "to get nodeconnector id associated with an interface";
    input {
        leaf intf-name {
            type string;
        }
    }
    output {
        leaf nodeconnector-id {
            type inv:node-connector-id;
        }
    }
}
```

get-interface-from-if-index

This RPC is used to get interface associated with an if-index (dataplane interface id).

```
rpc get-interface-from-if-index {
    description "to get interface associated with an if-index";
        input {
            leaf if-index {
                type int32;
            }
        }
        output {
            leaf interface-name {
                type string;
            }
        }
    }
```

create-terminating-service-actions

This RPC is used to create the tunnel termination service table entries.

```
rpc create-terminating-service-actions {
description "create the ingress terminating service table entries";
    input {
         leaf dpid {
             type uint64;
         }
         leaf tunnel-key {
             type uint64;
         }
         leaf interface-name {
             type string;
         }
         uses offlow:instruction-list;
    }
}
```

remove-terminating-service-actions

This RPC is used to remove the tunnel termination service table entries.

```
rpc remove-terminating-service-actions {
description "remove the ingress terminating service table entries";
    input {
         leaf dpid {
             type uint64;
         }
         leaf interface-name {
             type string;
         }
         leaf tunnel-key {
             type uint64;
         }
    }
}
```

ID Manager

TBD.

Group Based Policy User Guide

Overview

OpenDaylight Group Based Policy allows users to express network configuration in a declarative versus imperative way.

This is often described as asking for **"what you want"**, rather than **"how to do it"**.

In order to achieve this Group Based Policy (herein referred to as **GBP**) is an implementation of an **Intent System**.

An **Intent System**:

- is a process around an intent driven data model
- contains no domain specifics
- is capable of addressing multiple semantic definitions of intent

To this end, **GBP** Policy views an **Intent System** visually as:

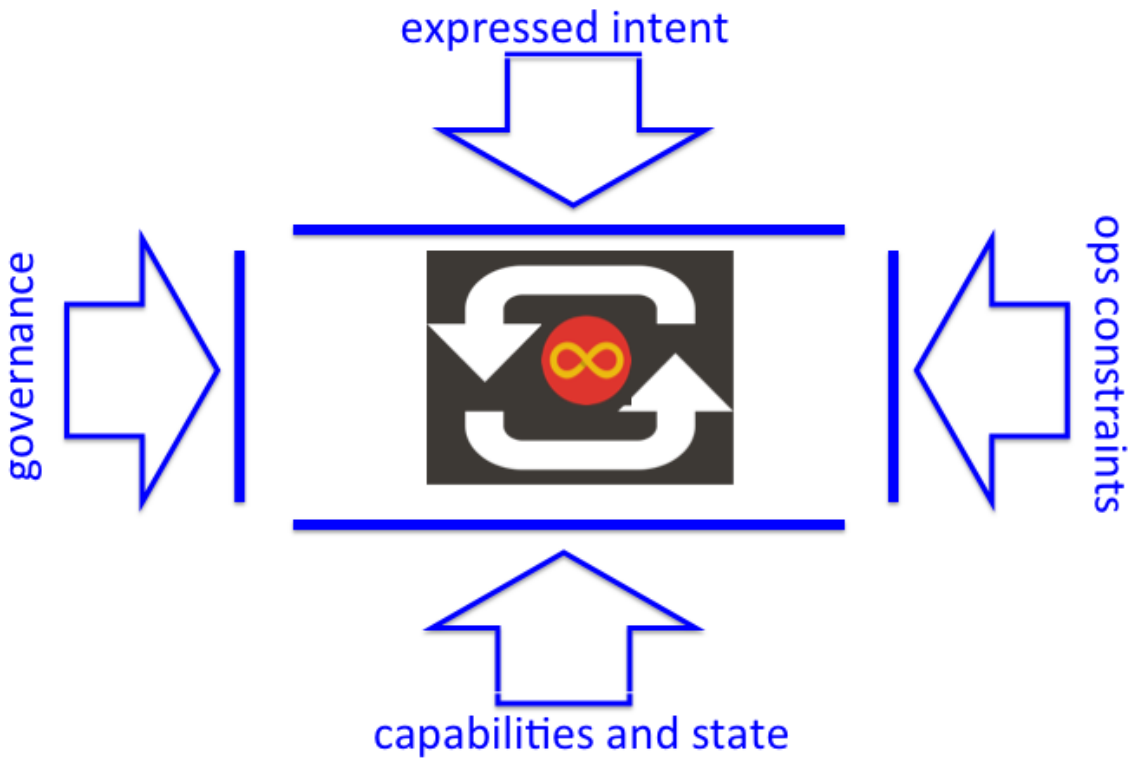


Fig. 1.28: Intent System Process and Policy Surfaces

- **expressed intent** is the entry point into the system.
- **operational constraints** provide policy for the usage of the system which modulates how the system is consumed. For instance *"All Financial applications must use a specific encryption standard"*.
- **capabilities and state** are provided by *renderers*. *Renderers* dynamically provide their capabilities to the core model, allowing the core model to remain non-domain specific.
- **governance** provides feedback on the delivery of the *expressed intent*. i.e. *"Did we do what you asked us?"*

In summary **GBP is about the Automation of Intent**.

By thinking of **Intent Systems** in this way, it enables:

- **automation of intent**

By focusing on **Model. Process. Automation**, a consistent policy resolution process enables for mapping between the **expressed intent** and renderers responsible for providing the capabilities of implementing that intent.

- recursive/intent level-independent behaviour.

  Where *one person's concrete is another's abstract*, intent can be fulfilled through a hierarchical implementation of non-domain specific policy resolution. Domain specifics are provided by the *renderers*, and exposed via the API, at each policy resolution instance. For example:

  - To DNS: The name "www.foo.com" is *abstract*, and it's IPv4 address 10.0.0.10 is *concrete*,
  - To an IP stack: 10.0.0.10 is *abstract* and the MAC 08:05:04:03:02:01 is *concrete*,
  - To an Ethernet switch: The MAC 08:05:04:03:02:01 is *abstract*, the resolution to a port in it's CAM table is *concrete*,
  - To an optical network: The port maybe *abstract*, yet the optical wavelength is *concrete*.

Note: *This is a very domain specific analogy, tied to something most readers will understand. It in no way implies the **GBP* should be implemented in an OSI type fashion. The premise is that by implementing a full **Intent System**, the user is freed from a lot of the constraints of how the expressed intent is realised.*

It is important to show the overall philosophy of **GBP** as it sets the project's direction.

In this release of OpenDaylight, **GBP** focused on **expressed intent**, **refactoring of how renderers consume and publish Subject Feature Definitions for multi-renderer support**.

GBP Base Architecture and Value Proposition

Terminology

In order to explain the fundamental value proposition of **GBP**, an illustrated example is given. In order to do that some terminology must be defined.

The Access Model is the core of the **GBP** Intent System policy resolution process.

- Endpoints:

  Define concrete uniquely identifiable entities. In this release, examples could be a Docker container, or a Neutron port

- EndpointGroups:

  EndpointGroups are sets of endpoints that share a common set of policies. EndpointGroups can participate in contracts that determine the kinds of communication that are allowed. EndpointGroups *consume* and *provide* contracts. They also expose both *requirements and capabilities*, which are labels that help to determine how contracts will be applied. An EndpointGroup can specify a parent EndpointGroup from which it inherits.

- Contracts:

  Contracts determine which endpoints can communicate and in what way. Contracts between pairs of EndpointGroups are selected by the contract selectors defined by the EndpointGroup. Contracts expose qualities, which are labels that can help EndpointGroups to select contracts. Once the contract is selected, contracts have clauses that can match against requirements and capabilities exposed by EndpointGroups, as well as any conditions that may be set on endpoints, in order to activate subjects that can allow specific kinds of communication. A contract is allowed to specify a parent contract from which it inherits.

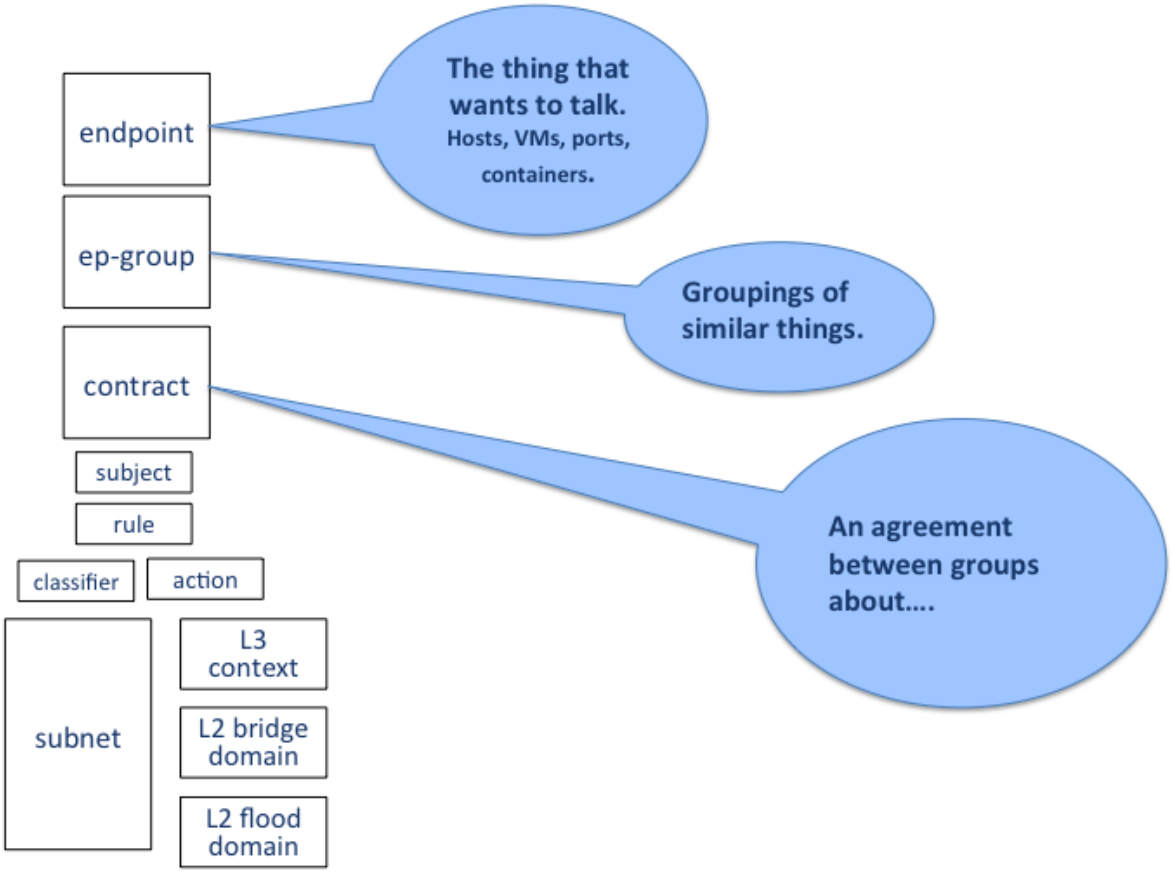


Fig. 1.29: GBP Access Model Terminology - Endpoints, EndpointGroups, Contract

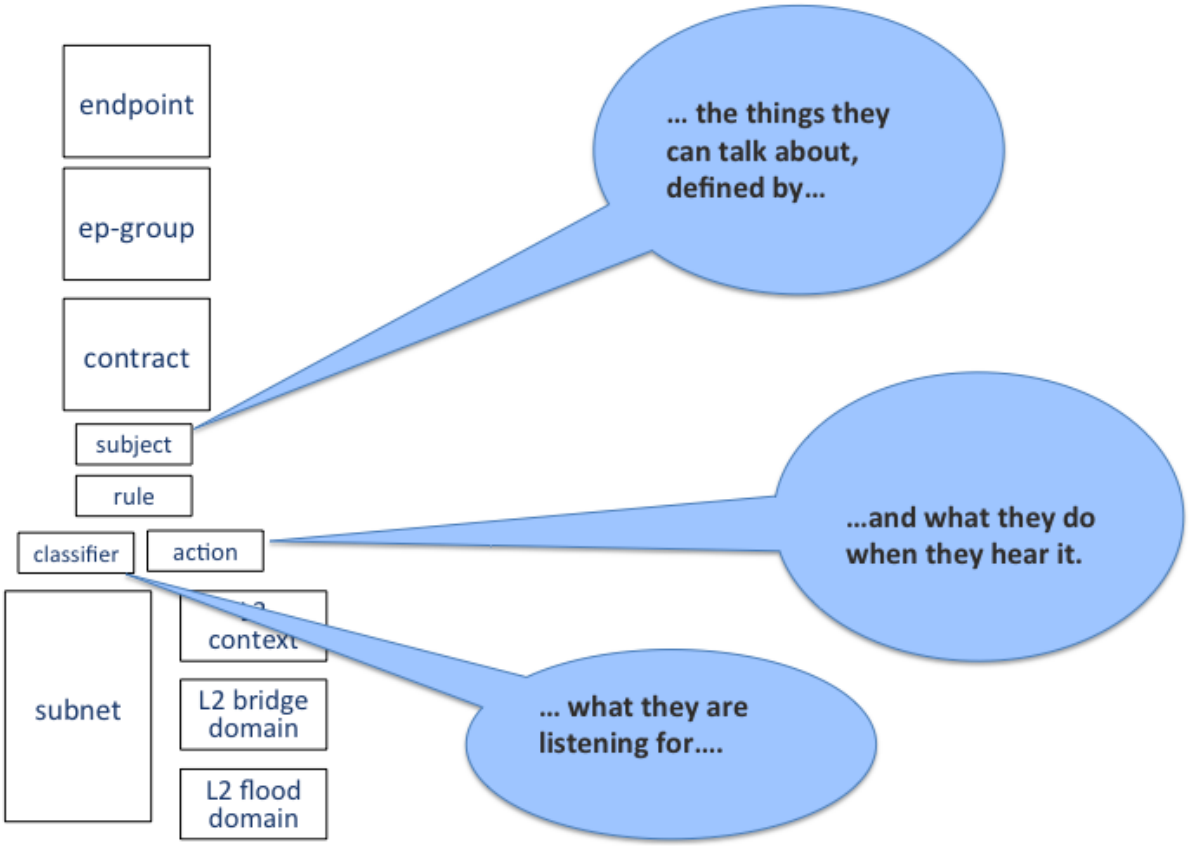


Fig. 1.30: GBP Access Model Terminology - Subject, Classifier, Action

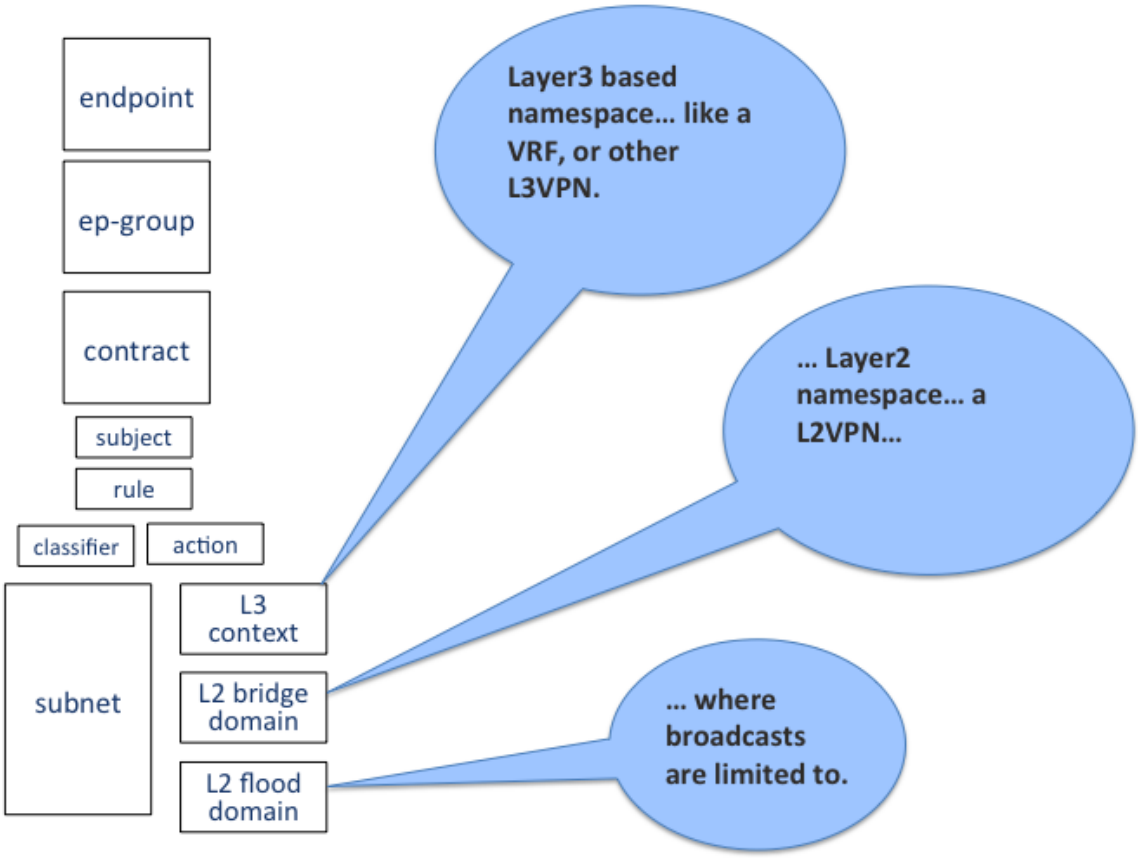


Fig. 1.31: GBP Forwarding Model Terminology - L3 Context, L2 Bridge Context, L2 Flood Context/Domain, Subnet

- Subject

  Subjects describe some aspect of how two endpoints are allowed to communicate. Subjects define an ordered list of rules that will match against the traffic and perform any necessary actions on that traffic. No communication is allowed unless a subject allows that communication.

- Clause

  Clauses are defined as part of a contract. Clauses determine how a contract should be applied to particular endpoints and EndpointGroups. Clauses can match against requirements and capabilities exposed by EndpointGroups, as well as any conditions that may be set on endpoints. Matching clauses define some set of subjects which can be applied to the communication between the pairs of endpoints.

Architecture and Value Proposition

GBP offers an intent based interface, accessed via the *UX*, via the *REST API* or directly from a domain-specific-language such as *Neutron* through a mapping interface.

There are two models in **GBP**:

- the access (or core) model
- the forwarding model

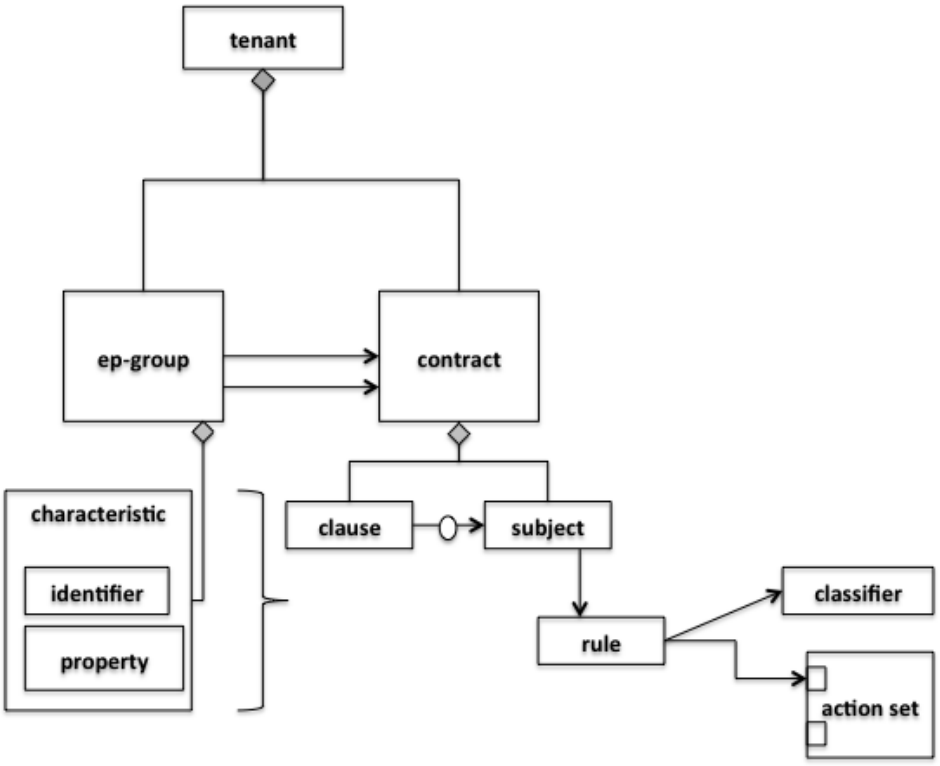


Fig. 1.32: GBP Access (or Core) Model

The *classifier* and *action* portions of the model can be thought of as hooks, with their definition provided by each *renderer* about its domain specific capabilities. In **GBP** for this release, there is one renderer, the *OpenFlow Overlay renderer (OfOverlay).*

These hooks are filled with *definitions* of the types of *features* the renderer can provide the *subject*, and are called **subject-feature-definitions**.

This means an *expressed intent* can be fulfilled by, and across, multiple renderers simultaneously, without any specific provisioning from the consumer of **GBP**.

Since **GBP** is implemented in OpenDaylight, which is an SDN controller, it also must address networking. This is done via the *forwarding model*, which is domain specific to networking, but could be applied to many different *types* of networking.

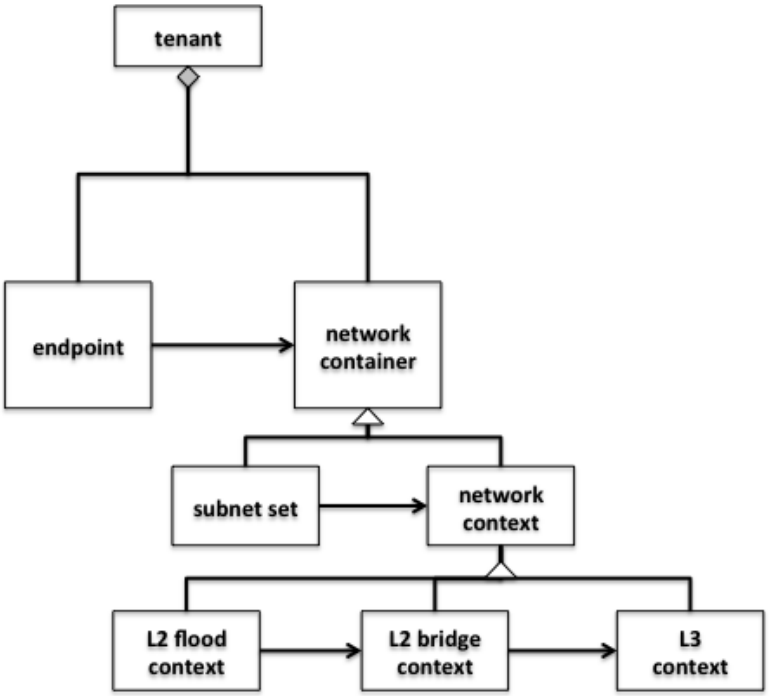


Fig. 1.33: GBP Forwarding Model

Each endpoint is provisioned with a *network-containment*. This can be a:

- subnet
  - normal IP stack behaviour, where ARP is performed in subnet, and for out of subnet, traffic is sent to default gateway.
  - a subnet can be a child of any of the below forwarding model contexts, but typically would be a child of a flood-domain
- L2 flood-domain

  - allows flooding behaviour.
  - is a n:1 child of a bridge-domain
  - can have multiple children
- L2 bridge-domain
  - is a layer2 namespace
  - is the realm where traffic can be sent at layer 2
  - is a n:1 child of a L3 context
  - can have multiple children
- L3 context
  - is a layer3 namespace
  - is the realm where traffic is passed at layer 3
  - is a n:1 child of a tenant
  - can have multiple children

A simple example of how the access and forwarding models work is as follows:

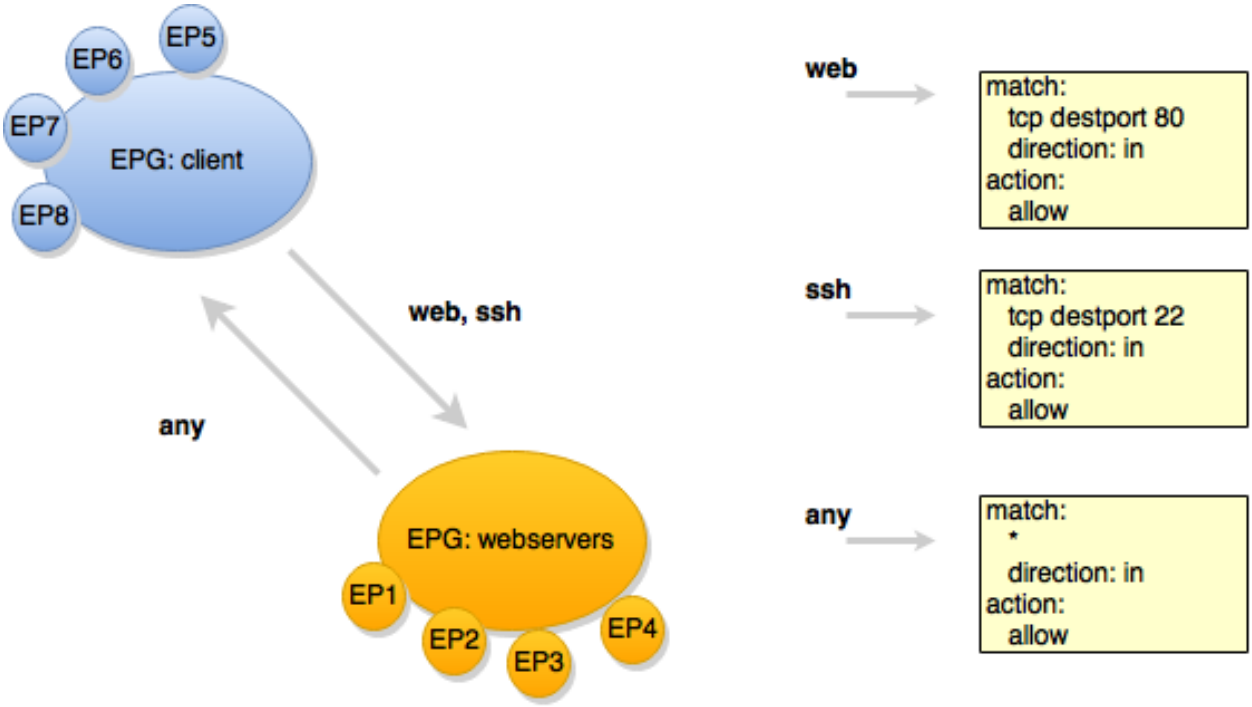


Fig. 1.34: GBP Endpoints, EndpointGroups and Contracts

In this example, the **EPG:webservers** is *providing* the *web* and *ssh* contracts. The **EPG:client** is consuming those contracts. **EPG:client** is providing the *any* contract, which is consumed by **EPG:webservers**.

The *direction* keyword is always from the perspective of the *provider* of the contract. In this case contract *web*, being *provided* by **EPG:webservers**, with the classifier to match TCP destination port 80, means:

- packets with a TCP destination port of 80

- sent to (*in*) endpoints in the **EPG:webservers**
- will be *allowed*.

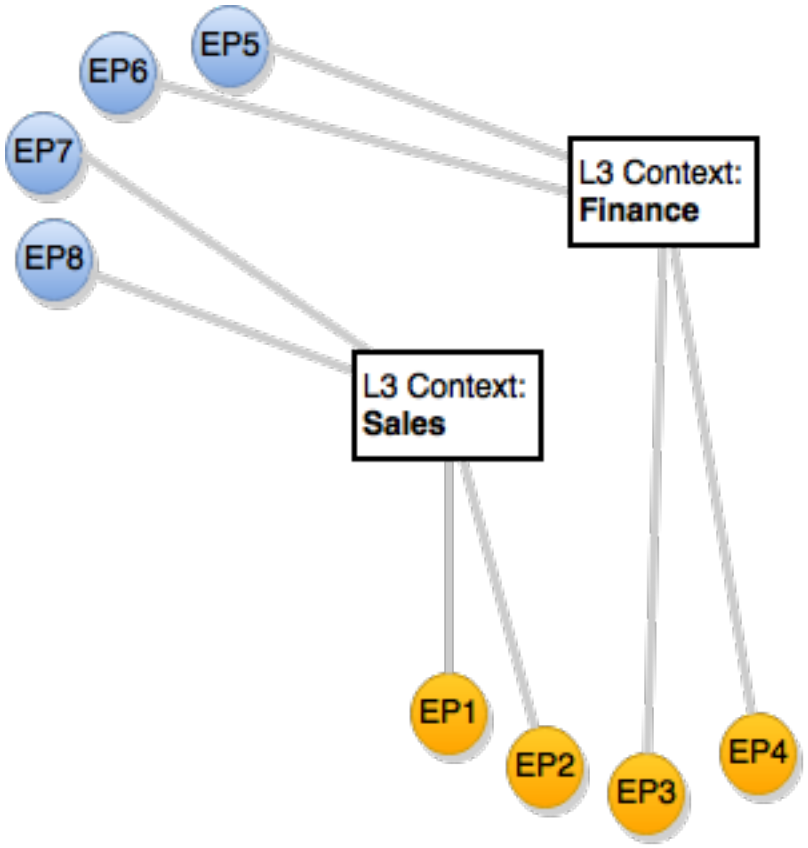


Fig. 1.35: GBP Endpoints and the Forwarding Model

When the forwarding model is considered in the figure above, it can be seen that even though all endpoints are communicating using a common set of contracts, their forwarding is *contained* by the forwarding model contexts or namespaces. In the example shown, the endpoints associated with a *network-containment* that has an ultimate parent of *L3Context:Sales* can only communicate with other endpoints within this L3Context. In this way L3VPN services can be implemented without any impact to the **Intent** of the contract.

High-level implementation Architecture

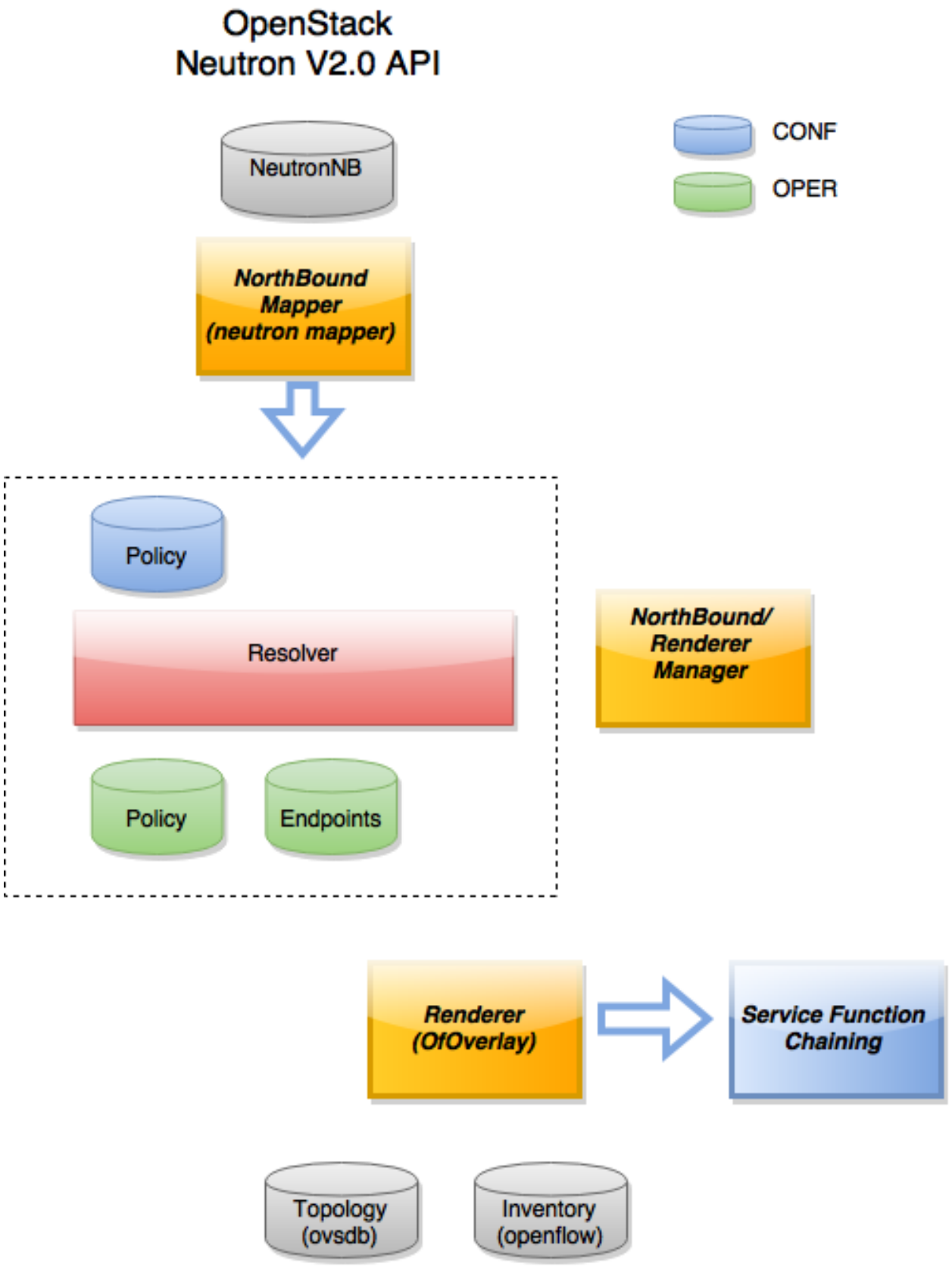
The overall architecture, including *Neutron* domain specific mapping, and the *OpenFlow Overlay renderer* looks as so:

The major benefit of this architecture is that the mapping of the domain-specific-language is completely separate and independent of the underlying renderer implementation.

For instance, using the *Neutron Mapper*, which maps the Neutron API to the **GBP** core model, any contract automatically generated from this mapping can be augmented via the *UX* to use *Service Function Chaining*, a capability not currently available in OpenStack Neutron.

When another renderer is added, for instance, NetConf, the same policy can now be leveraged across NetConf devices simultaneously:

As other domain-specific mappings occur, they too can leverage the same renderers, as the renderers only need to


OpenStack
Neutron V2.0 API
NeutronNB
CONF
OPER
NorthBound
Mapper
(neutron mapper)
Policy
Resolver
Policy
Endpoints
NorthBound/
Renderer
Manager
Renderer
(OfOverlay)
Service Function
Chaining
Topology
(ovsdb)
Inventory
(openflow)

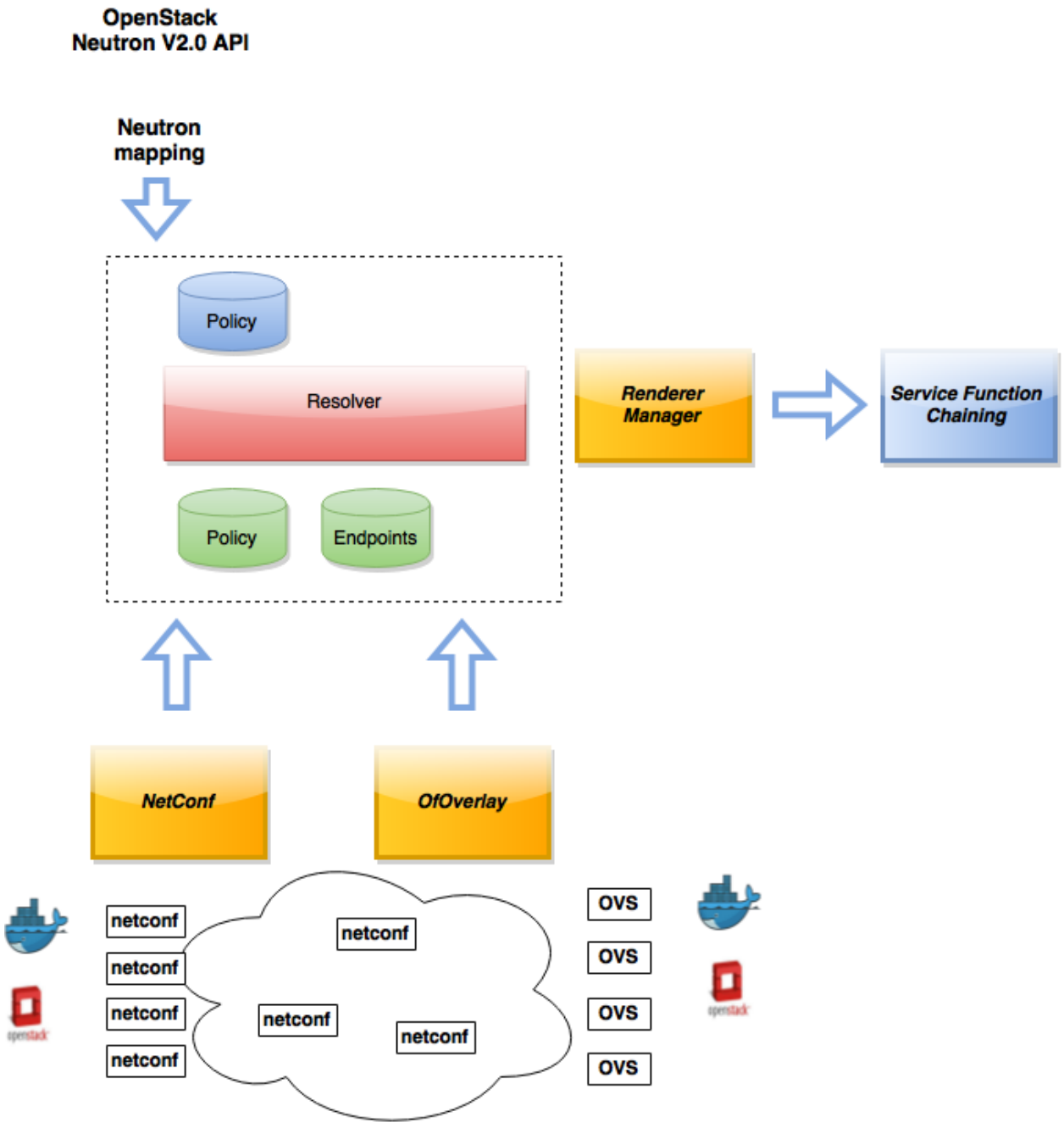


Fig. 1.37: GBP High Level Architecture - adding a renderer

implement the **GBP** access and forwarding models, and the domain-specific mapping need only manage mapping to the access and forwarding models. For instance:

In summary, the **GBP** architecture:

- separates concerns: the Expressed Intent is kept completely separated from the underlying renderers.
- is cohesive: each part does it's part and it's part only
- is scalable: code can be optimised around model mapping/implementation, and functionality re-used

Policy Resolution

Contract Selection

The first step in policy resolution is to select the contracts that are in scope.

EndpointGroups participate in contracts either as a *provider* or as a *consumer* of a contract. Each EndpointGroup can participate in many contracts at the same time, but for each contract it can be in only one role at a time. In addition, there are two ways for an EndpointGroup to select a contract: either with either a:

- *named selector*

  Named selectors simply select a specific contract by its contract ID.

- target selector.

  Target selectors allow for additional flexibility by matching against *qualities* of the contract's *target.*

Thus, there are a total of 4 kinds of contract selector:

- provider named selector

  Select a contract by contract ID, and participate as a provider.

- provider target selector

  Match against a contract's target with a quality matcher, and participate as a provider.

- consumer named selector

  Select a contract by contract ID, and participate as a consumer.

- consumer target selector

  Match against a contract's target with a quality matcher, and participate as a consumer.

To determine which contracts are in scope, contracts are found where either the source EndpointGroup selects a contract as either a provider or consumer, while the destination EndpointGroup matches against the same contract in the corresponding role. So if endpoint *x* in EndpointGroup *X* is communicating with endpoint *y* in EndpointGroup *Y*, a contract *C* is in scope if either *X* selects *C* as a provider and *Y* selects *C* as a consumer, or vice versa.

The details of how quality matchers work are described further in *Matchers*. Quality matchers provide a flexible mechanism for contract selection based on labels.

The end result of the contract selection phase can be thought of as a set of tuples representing selected contract scopes. The fields of the tuple are:

- Contract ID
- The provider EndpointGroup ID
- The name of the selector in the provider EndpointGroup that was used to select the contract, called the *matching provider selector.*

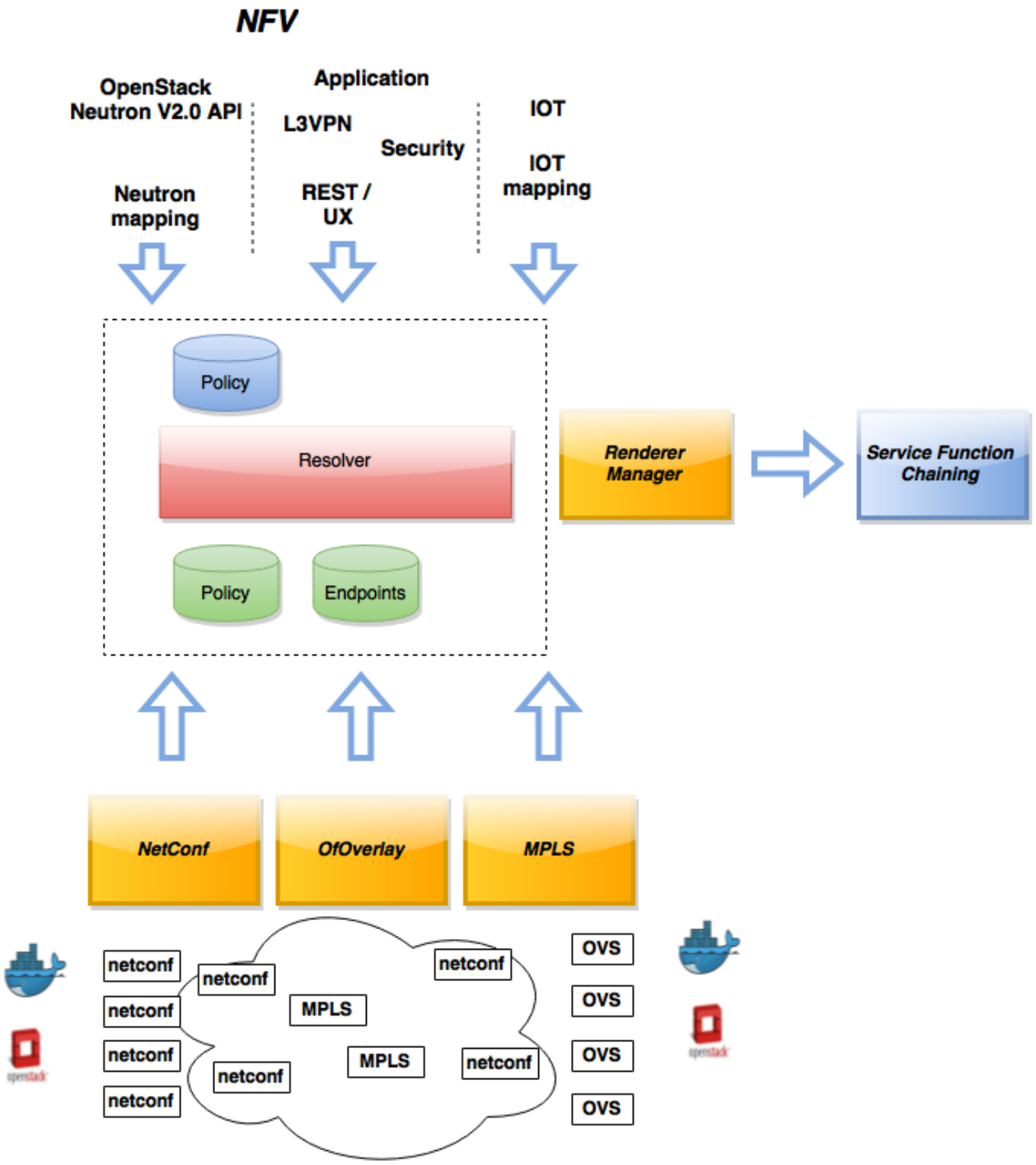


Fig. 1.38: GBP High Level Architecture - adding a renderer

- The consumer EndpointGroup ID
- The name of the selector in the consumer EndpointGroup that was used to select the contract, called the *matching consumer selector.*

The result is then stored in the datastore under **Resolved Policy**.

Subject Selection

The second phase in policy resolution is to determine which subjects are in scope. The subjects define what kinds of communication are allowed between endpoints in the EndpointGroups. For each of the selected contract scopes from the contract selection phase, the subject selection procedure is applied.

Labels called, capabilities, requirements and conditions are matched against to bring a Subject *into scope*. EndpointGroups have capabilities and requirements, while endpoints have conditions.

Requirements and Capabilities

When acting as a *provider*, EndpointGroups expose *capabilities,* which are labels representing specific pieces of functionality that can be exposed to other EndpointGroups that may meet functional requirements of those EndpointGroups.

When acting as a *consumer*, EndpointGroups expose *requirements*, which are labels that represent that the EndpointGroup requires some specific piece of functionality.

As an example, we might create a capability called "user-database" which indicates that an EndpointGroup contains endpoints that implement a database of users.

We might create a requirement also called "user-database" to indicate an EndpointGroup contains endpoints that will need to communicate with the endpoints that expose this service.

Note that in this example the requirement and capability have the same name, but the user need not follow this convention.

The matching provider selector (that was used by the provider EndpointGroup to select the contract) is examined to determine the capabilities exposed by the provider EndpointGroup for this contract scope.

The provider selector will have a list of capabilities either directly included in the provider selector or inherited from a parent selector or parent EndpointGroup. (See *Inheritance*).

Similarly, the matching consumer selector will expose a set of requirements.

Conditions

Endpoints can have *conditions*, which are labels representing some relevant piece of operational state related to the endpoint.

An example of a condition might be "malware-detected," or "authentication-succeeded." Conditions are used to affect how that particular endpoint can communicate.

To continue with our example, the "malware-detected" condition might cause an endpoint's connectivity to be cut off, while "authentication-succeeded" might open up communication with services that require an endpoint to be first authenticated and then forward its authentication credentials.

Clauses

Clauses perform the actual selection of subjects. A clause has lists of matchers in two categories. In order for a clause to become active, all lists of matchers must match. A matching clause will select all the subjects referenced by the clause. Note that an empty list of matchers counts as a match.

The first category is the consumer matchers, which match against the consumer EndpointGroup and endpoints. The consumer matchers are:

- Group Idenfication Constraint: Requirement matchers

  Matches against requirements in the matching consumer selector.

- Group Identification Constraint: GroupName

  Matches against the group name

- Consumer condition matchers

  Matches against conditions on endpoints in the consumer EndpointGroup

- Consumer Endpoint Identification Constraint

  Label based criteria for matching against endpoints. In this release this can be used to label endpoints based on IpPrefix.

The second category is the provider matchers, which match against the provider EndpointGroup and endpoints. The provider matchers are:

- Group Idenfication Constraint: Capability matchers

  Matches against capabilities in the matching provider selector.

- Group Identification Constraint: GroupName

  Matches against the group name

- Consumer condition matchers

  Matches against conditions on endpoints in the provider EndpointGroup

- Consumer Endpoint Identification Constraint

  Label based criteria for matching against endpoints. In this release this can be used to label endpoints based on IpPrefix.

Clauses have a list of subjects that apply when all the matchers in the clause match. The output of the subject selection phase logically is a set of subjects that are in scope for any particular pair of endpoints.

Rule Application

Now subjects have been selected that apply to the traffic between a particular set of endpoints, policy can be applied to allow endpoints to communicate. The applicable subjects from the previous step will each contain a set of rules.

Rules consist of a set of *classifiers* and a set of *actions*. Classifiers match against traffic between two endpoints. An example of a classifier would be something that matches against all TCP traffic on port 80, or one that matches against HTTP traffic containing a particular cookie. Actions are specific actions that need to be taken on the traffic before it reaches its destination. Actions could include tagging or encapsulating the traffic in some way, redirecting the traffic, or applying a *service function chain*.

Rules, subjects, and actions have an *order* parameter, where a lower order value means that a particular item will be applied first. All rules from a particular subject will be applied before the rules of any other subject, and all actions from a particular rule will be applied before the actions from another rule. If more than item has the same order

parameter, ties are broken with a lexicographic ordering of their names, with earlier names having logically lower order.

Matchers

Matchers specify a set of labels (which include requirements, capabilities, conditions, and qualities) to match against. There are several kinds of matchers that operate similarly:

- Quality matchers

  used in target selectors during the contract selection phase. Quality matchers provide a more advanced and flexible way to select contracts compared to a named selector.

- Requirement and capability matchers

  used in clauses during the subject selection phase to match against requirements and capabilities on EndpointGroups

- Condition matchers

  used in clauses during the subject selection phase to match against conditions on endpoints

A matcher is, at its heart, fairly simple. It will contain a list of label names, along with a *match type*. The match type can be either:

- "all"

  which means the matcher matches when all of its labels match

- "any"

  which means the matcher matches when any of its labels match,

- "none"

  which means the matcher matches when none of its labels match.

Note a *match all* matcher can be made by matching against an empty set of labels with a match type of "all."

Additionally each label to match can optionally include a relevant name field. For quality matchers, this is a target name. For capability and requirement matchers, this is a selector name. If the name field is specified, then the matcher will only match against targets or selectors with that name, rather than any targets or selectors.

Inheritance

Some objects in the system include references to parents, from which they will inherit definitions. The graph of parent references must be loop free. When resolving names, the resolution system must detect loops and raise an exception. Objects that are part of these loops may be considered as though they are not defined at all. Generally, inheritance works by simply importing the objects in the parent into the child object. When there are objects with the same name in the child object, then the child object will override the parent object according to rules which are specific to the type of object. We'll next explore the detailed rules for inheritance for each type of object

EndpointGroups

EndpointGroups will inherit all their selectors from their parent EndpointGroups. Selectors with the same names as selectors in the parent EndpointGroups will inherit their behavior as defined below.

Selectors

Selectors include provider named selectors, provider target selectors, consumer named selectors, and consumer target selectors. Selectors cannot themselves have parent selectors, but when selectors have the same name as a selector of

the same type in the parent EndpointGroup, then they will inherit from and override the behavior of the selector in the parent EndpointGroup.

Named Selectors

Named selectors will add to the set of contract IDs that are selected by the parent named selector.

Target Selectors

A target selector in the child EndpointGroup with the same name as a target selector in the parent EndpointGroup will inherit quality matchers from the parent. If a quality matcher in the child has the same name as a quality matcher in the parent, then it will inherit as described below under Matchers.

Contracts

Contracts will inherit all their targets, clauses and subjects from their parent contracts. When any of these objects have the same name as in the parent contract, then the behavior will be as defined below.

Targets

Targets cannot themselves have a parent target, but it may inherit from targets with the same name as the target in a parent contract. Qualities in the target will be inherited from the parent. If a quality with the same name is defined in the child, then this does not have any semantic effect except if the quality has its inclusion-rule parameter set to "exclude." In this case, then the label should be ignored for the purpose of matching against this target.

Subjects

Subjects cannot themselves have a parent subject, but it may inherit from a subject with the same name as the subject in a parent contract. The order parameter in the child subject, if present, will override the order parameter in the parent subject. The rules in the parent subject will be added to the rules in the child subject. However, the rules will not override rules of the same name. Instead, all rules in the parent subject will be considered to run with a higher order than all rules in the child; that is all rules in the child will run before any rules in the parent. This has the effect of overriding any rules in the parent without the potentially-problematic semantics of merging the ordering.

Clauses

Clauses cannot themselves have a parent clause, but it may inherit from a clause with the same name as the clause in a parent contract. The list of subject references in the parent clause will be added to the list of subject references in the child clause. This is just a union operation. A subject reference that refers to a subject name in the parent contract might have that name overridden in the child contract. Each of the matchers in the clause are also inherited by the child clause. Matchers in the child of the same name and type as a matcher from the parent will inherit from and override the parent matcher. See below under Matchers for more information.

Matchers

Matchers include quality matchers, condition matchers, requirement matchers, and capability matchers. Matchers cannot themselves have parent matchers, but when there is a matcher of the same name and type in the parent object, then the matcher in the child object will inherit and override the behavior of the matcher in the parent object. The match type, if specified in the child, overrides the value specified in the parent. Labels are also inherited from the parent object. If there is a label with the same name in the child object, this does not have any semantic effect except if the label has its inclusion-rule parameter set to "exclude." In this case, then the label should be ignored for the purpose of matching. Otherwise, the label with the same name will completely override the label from the parent.

Using the GBP UX interface

Overview

These following components make up this application and are described in more detail in following sections:

- Basic view

- Governance view
- Policy Expression view
- Wizard view

The **GBP** UX is access via:

```
http://<odl controller>:8181/index.html
```

Basic view

Basic view contains 5 navigation buttons which switch user to the desired section of application:

- Governance – switch to the Governance view (middle of graphic has the same function)
- Renderer configuration – switch to the Policy expression view with Renderers section expanded
- Policy expression – switch to the Policy expression view with Policy section expanded
- Operational constraints – placeholder for development in next release

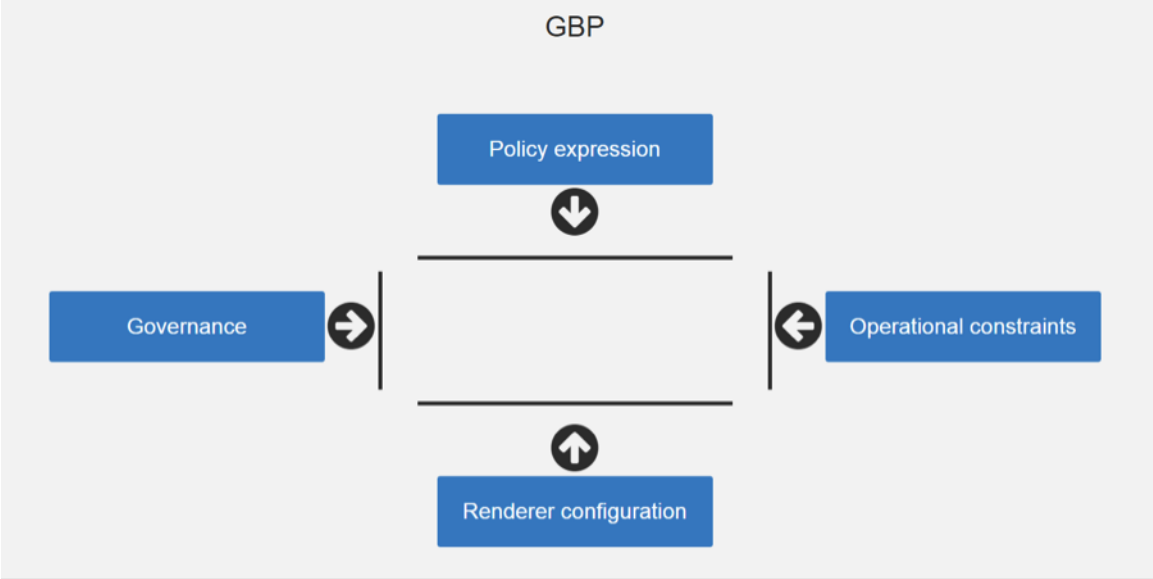


Fig. 1.39: Basic view

Governance view

Governance view consists from three columns.

Governance view – Basic view – Left column

In the left column is Health section with Exception and Conflict buttons with no functionality yet. This is a placeholder for development in further releases.

Governance view – Basic view – Middle column

In the top half of this section is select box with list of tenants for select. Once the tenant is selected, all sub sections in application operate and display data with actual selected tenant.

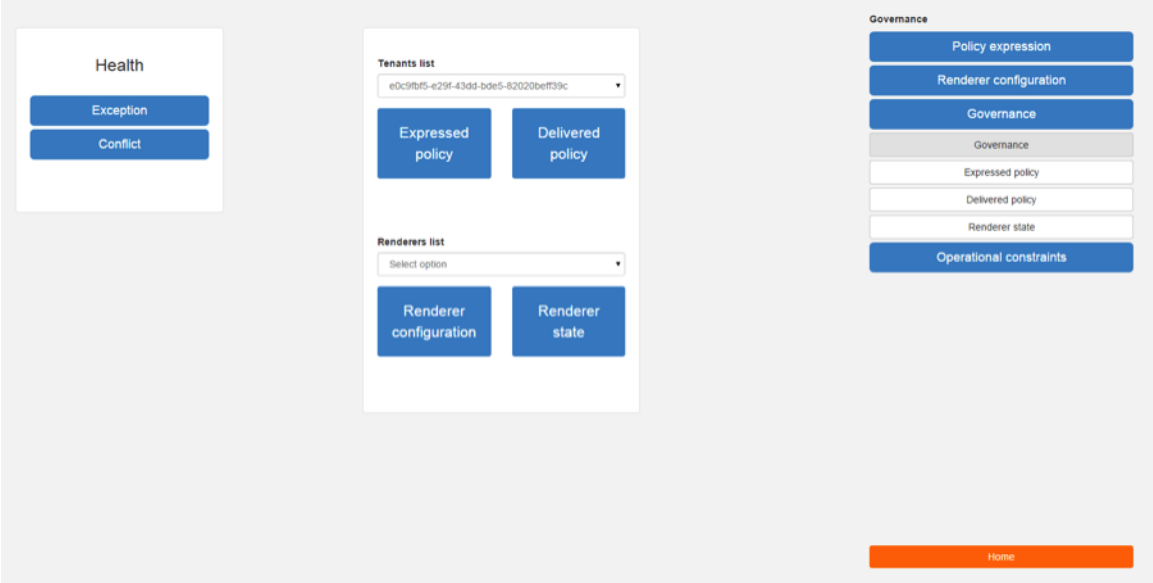


Fig. 1.40: Governance view

Below the select box are buttons which display Expressed or Delivered policy of Governance section. In the bottom half of this section is select box with list of renderers for select. There is currently only *OfOverlay* renderer available.

Below the select box is Renderer configuration button, which switch the app into the Policy expression view with Renderers section expanded for performing CRUD operations. Renderer state button display Renderer state view.

Governance view – Basic view – Right column

In the bottom part of the right section of Governance view is Home button which switch the app to the Basic view.

In the top part is situated navigation menu with four main sections.

Policy expression button expand/collapse sub menu with three main parts of Policy expression. By clicking on sub menu buttons, user will be switched into the Policy expressions view with appropriate section expanded for performing CRUD operations.

Renderer configuration button switches user into the Policy expressions view.

Governance button expand/collapse sub menu with four main parts of Governance section. Sub menu buttons of Governance section display appropriate section of Governance view.

Operational constraints have no functionality yet, and is a placeholder for development in further releases.

Below the menu is place for view info section which displays info about actual selected element from the topology (explained below).

Governance view – Expressed policy

In this view are displayed contracts with their consumed and provided EndpointGroups of actual selected tenant, which can be changed in select box in the upper left corner.

By single-clicking on any contract or EPG, the data of actual selected element will be shown in the right column below the menu. A Manage button launches a display wizard window for managing configuration of items such as *Service Function Chaining*.

Governance view – Delivered policy In this view are displayed subjects with their consumed and provided Endpoint-Groups of actual selected tenant, which can be changed in select box in the upper left corner.

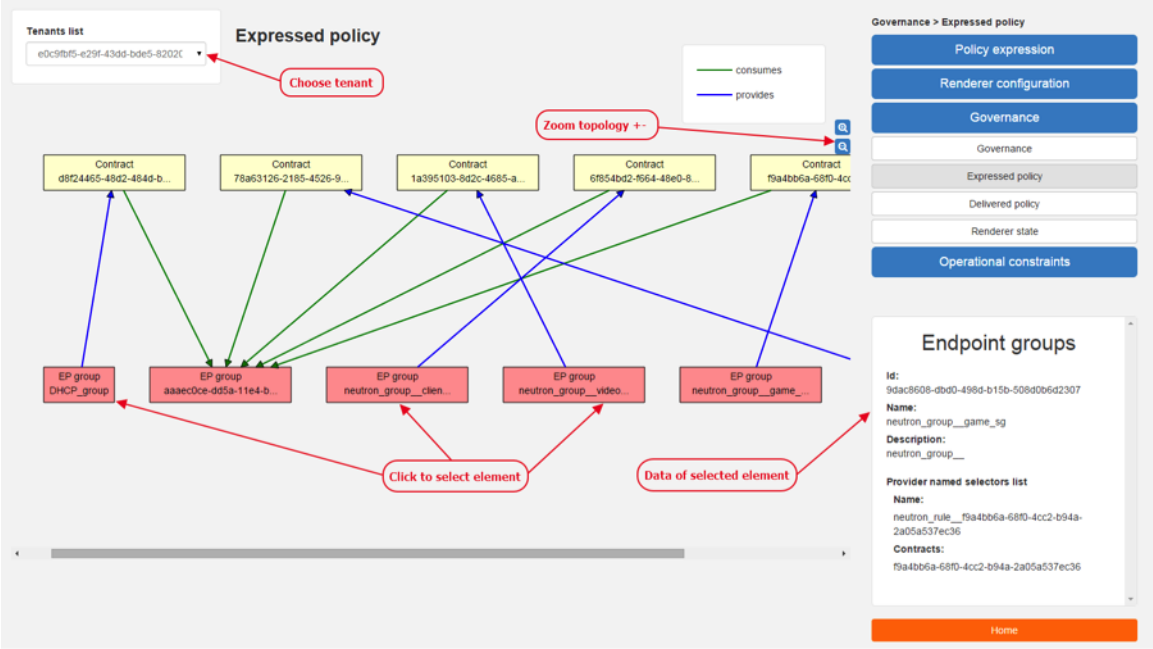


Fig. 1.41: Expressed policy

By single-clicking on any subject or EPG, the data of actual selected element will be shown in the right column below the menu.

By double-click on subject the subject detail view will be displayed with subject's rules of actual selected subject, which can be changed in select box in the upper left corner.

By single-clicking on rule or subject, the data of actual selected element will be shown in the right column below the menu.

By double-clicking on EPG in Delivered policy view, the EPG detail view will be displayed with EPG's endpoints of actual selected EPG, which can be changed in select box in the upper left corner.

By single-clicking on EPG or endpoint the data of actual selected element will be shown in the right column below the menu.

Governance view – Renderer state

In this part are displayed Subject feature definition data with two main parts: Action definition and Classifier definition.

By clicking on the down/right arrow in the circle is possible to expand/hide data of appropriate container or list. Next to the list node are displayed names of list's elements where one is always selected and element's data are shown (blue line under the name).

By clicking on names of children nodes is possible to select desired node and node's data will be displayed.

Policy expression view

In the left part of this view is placed topology of actual selected elements with the buttons for switching between types of topology at the bottom.

Right column of this view contains four parts. At the top of this column are displayed breadcrumbs with actual position in the application.

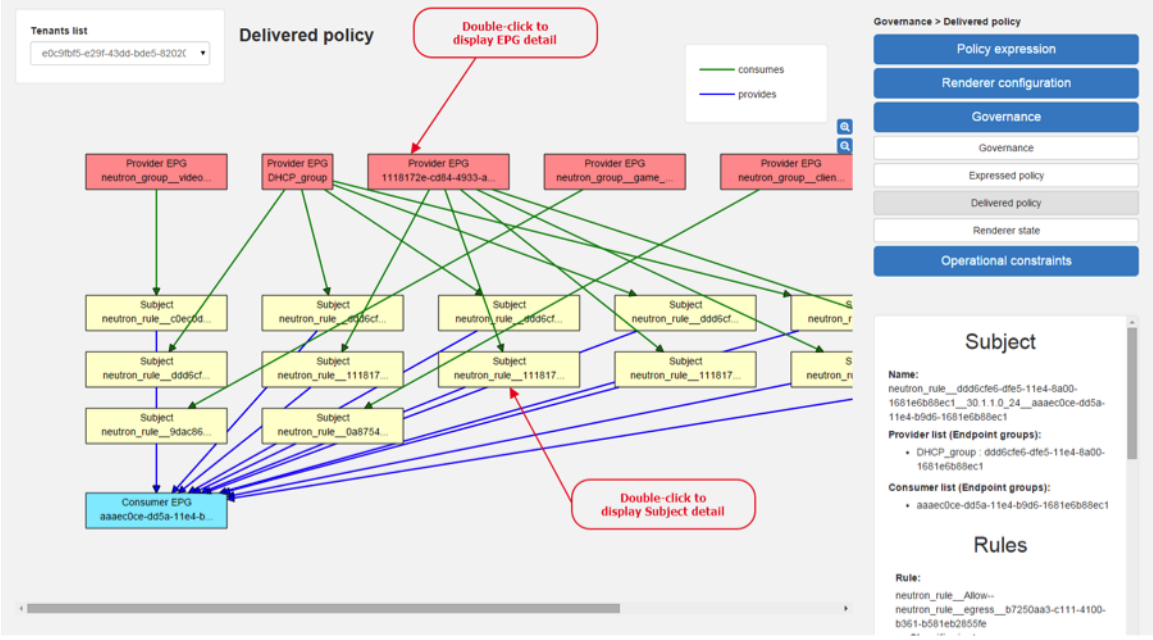


Fig. 1.42: Delivered policy

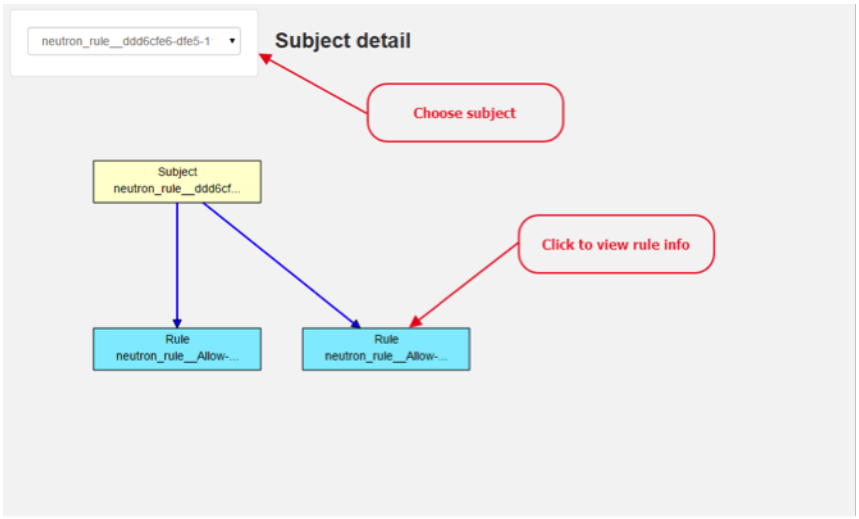


Fig. 1.43: Subject detail

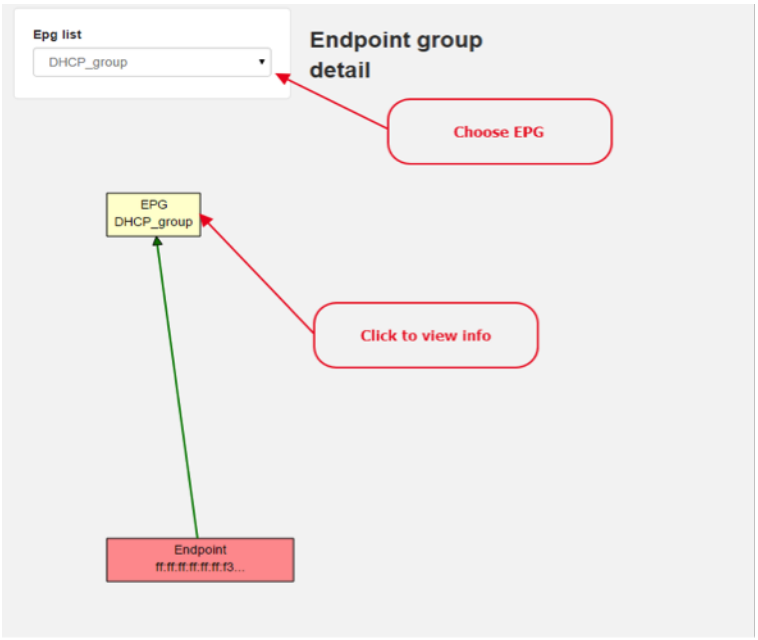


Fig. 1.44: EPG detail

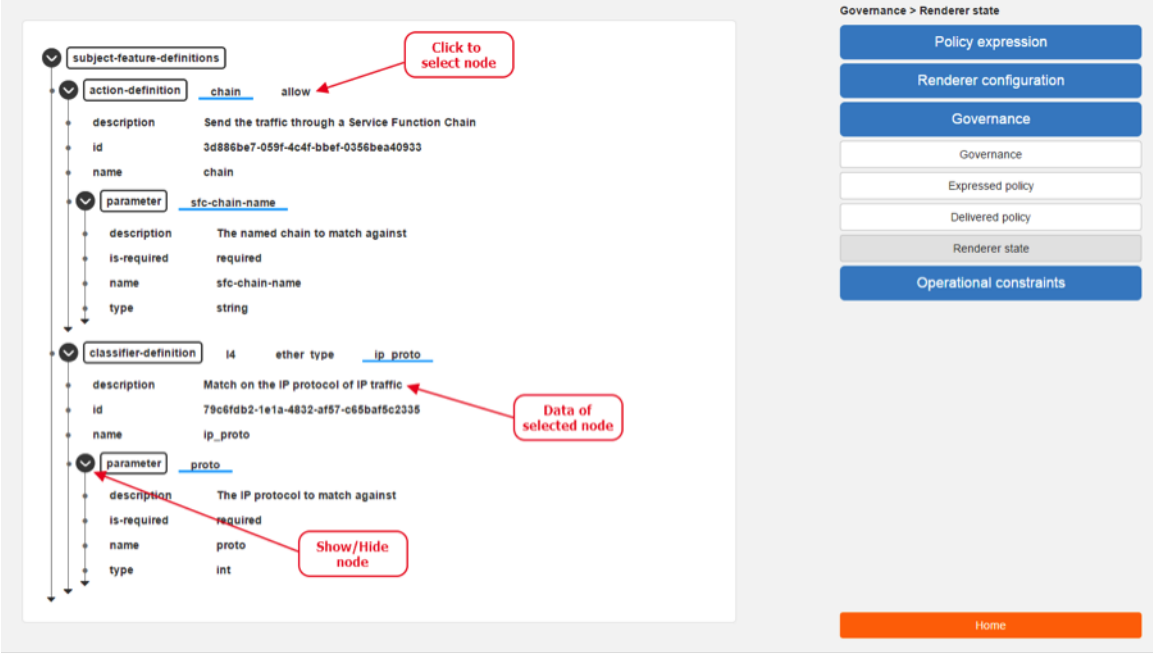


Fig. 1.45: Renderer state

Below the breadcrumbs is select box with list of tenants for select. In the middle part is situated navigation menu, which allows switch to the desired section for performing CRUD operations.

At the bottom is quick navigation menu with Access Model Wizard button which display Wizard view, Home button which switch application to the Basic view and occasionally Back button, which switch application to the upper section.

Policy expression - Navigation menu

To open Policy expression, select Policy expression from the GBP Home screen.

In the top of navigation box you can select the tenant from the tenants list to activate features addicted to selected tenant.

In the right menu, by default, the Policy menu section is expanded. Subitems of this section are modules for CRUD (creating, reading, updating and deleting) of tenants, EndpointGroups, contracts, L2/L3 objects.

- Section Renderers contains CRUD forms for Classifiers and Actions.
- Section Endpoints contains CRUD forms for Endpoint and L3 prefix endpoint.

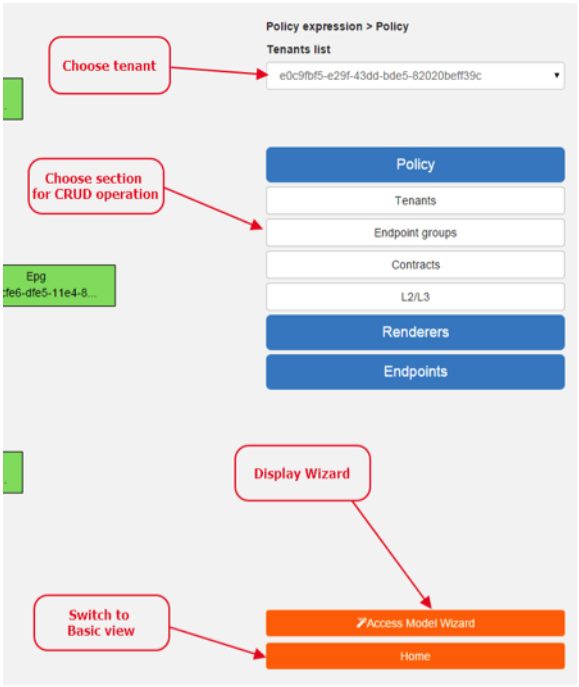


Fig. 1.46: Navigation menu

Policy expression - Types of topology

There are three different types of topology:

- Configured topology - EndpointGroups and contracts between them from CONFIG datastore
- Operational topology - displays same information but is based on operational data.
- L2/L3 - displays relationships between L3Contexts, L2 Bridge domains, L2 Flood domains and Subnets.

Policy expression - CRUD operations

In this part are described basic flows for viewing, adding, editing and deleting system elements like tenants, Endpoint-Groups etc.

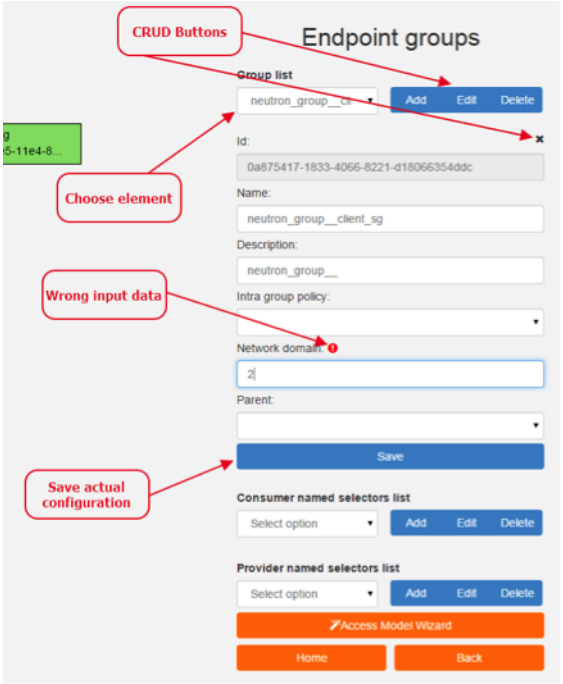


Fig. 1.47: CRUD operations

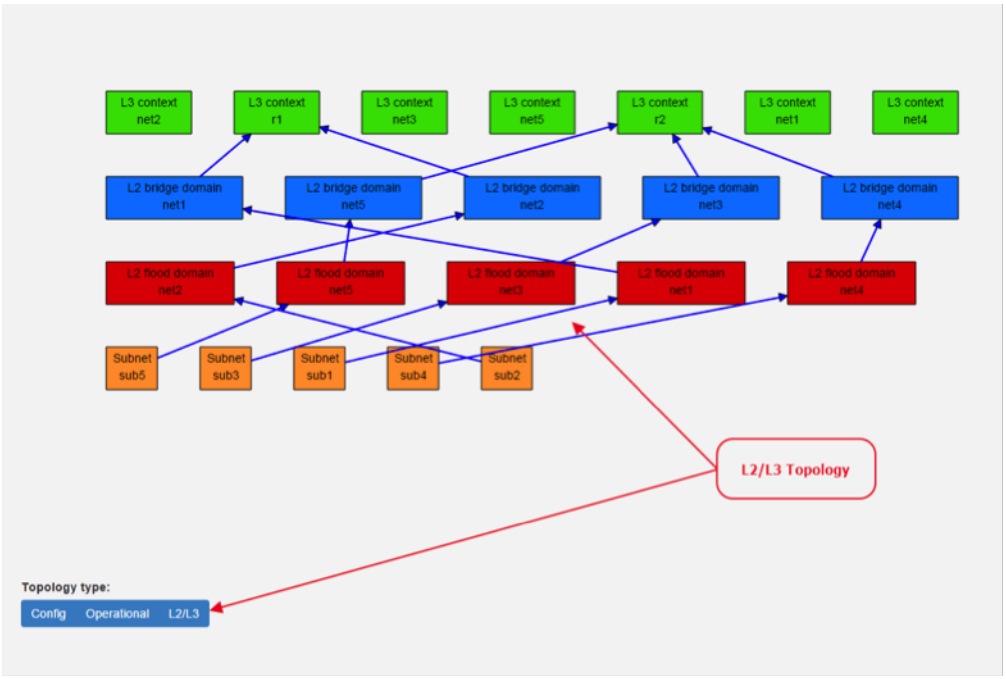


Fig. 1.48: L2/L3 Topology

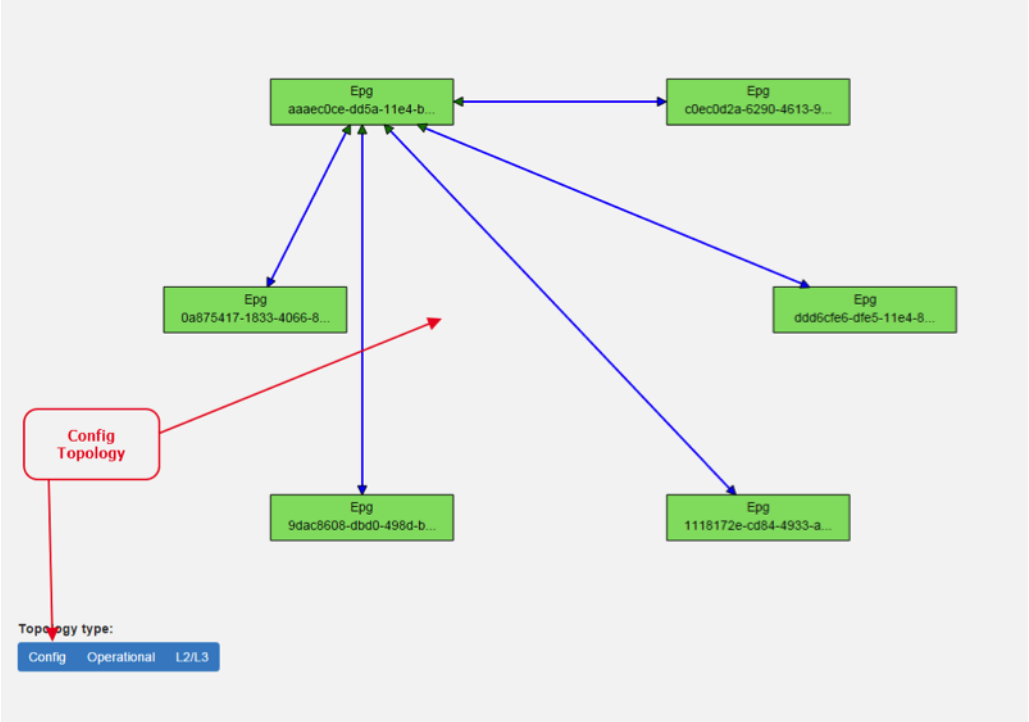


Fig. 1.49: Config Topology

Tenants

To edit tenant objects click the Tenants button in the right menu. You can see the CRUD form containing tenants list and control buttons.

To add new tenant, click the Add button This will display the form for adding a new tenant. After filling tenant attributes Name and Description click Save button. Saving of any object can be performed only if all the object attributes are filled correctly. If some attribute doesn't have correct value, exclamation mark with mouse-over tooltip will be displayed next to the label for the attribute. After saving of tenant the form will be closed and the tenants list will be set to default value.

To view an existing tenant, select the tenant from the select box Tenants list. The view form is read-only and can be closed by clicking cross mark in the top right of the form.

To edit selected tenant, click the Edit button, which will display the edit form for selected tenant. After editing the Name and Description of selected tenant click the Save button to save selected tenant. After saving of tenant the edit form will be closed and the tenants list will be set to default value.

To delete tenant select the tenant from the Tenants list and click Delete button.

To return to the Policy expression click Back button on the bottom of window.

EndpointGroups

For managing EndpointGroups (EPG) the tenant from the top Tenants list must be selected.

To add new EPG click Add button and after filling required attributes click Save button. After adding the EPG you can edit it and assign Consumer named selector or Provider named selector to it.

To edit EPG click the Edit button after selecting the EPG from Group list.

To add new Consumer named selector (CNS) click the Add button next to the Consumer named selectors list. While

CNS editing you can set one or more contracts for current CNS pressing the Plus button and selecting the contract from the Contracts list. To remove the contract, click on the cross mark next to the contract. Added CNS can be viewed, edited or deleted by selecting from the Consumer named selectors list and clicking the Edit and Delete buttons like with the EPG or tenants.

To add new Provider named selector (PNS) click the Add button next to the Provider named selectors list. While PNS editing you can set one or more contracts for current PNS pressing the Plus button and selecting the contract from the Contracts list. To remove the contract, click on the cross mark next to the contract. Added PNS can be viewed, edited or deleted by selecting from the Provider named selectors list and clicking the Edit and Delete buttons like with the EPG or tenants.

To delete EPG, CNS or PNS select it in selectbox and click the Delete button next to the selectbox.

Contracts

For managing contracts the tenant from the top Tenants list must be selected.

To add new Contract click Add button and after filling required fields click Save button.

After adding the Contract user can edit it by selecting in the Contracts list and clicking Edit button.

To add new Clause click Add button next to the Clause list while editing the contract. While editing the Clause after selecting clause from the Clause list user can assign clause subjects by clicking the Plus button next to the Clause subjects label. Adding and editing action must be submitted by pressing Save button. To manage Subjects you can use CRUD form like with the Clause list.

L2/L3

For managing L2/L3 the tenant from the top Tenants list must be selected.

To add L3 Context click the Add button next to the L3 Context list ,which will display the form for adding a new L3 Context. After filling L3 Context attributes click Save button. After saving of L3 Context, form will be closed and the L3 Context list will be set to default value.

To view an existing L3 Context, select the L3 Context from the select box L3 Context list. The view form is read-only and can be closed by clicking cross mark in the top right of the form.

If user wants to edit selected L3 Context, click the Edit button, which will display the edit form for selected L3 Context. After editing click the Save button to save selected L3 Context. After saving of L3 Context, the edit form will be closed and the L3 Context list will be set to default value.

To delete L3 Context, select it from the L3 Context list and click Delete button.

To add L2 Bridge Domain, click the Add button next to the L2 Bridge Domain list. This will display the form for adding a new L2 Bridge Domain. After filling L2 Bridge Domain attributes click Save button. After saving of L2 Bridge Domain, form will be closed and the L2 Bridge Domain list will be set to default value.

To view an existing L2 Bridge Domain, select the L2 Bridge Domain from the select box L2 Bridge Domain list. The view form is read-only and can be closed by clicking cross mark in the top right of the form.

If user wants to edit selected L2 Bridge Domain, click the Edit button, which will display the edit form for selected L2 Bridge Domain. After editing click the Save button to save selected L2 Bridge Domain. After saving of L2 Bridge Domain the edit form will be closed and the L2 Bridge Domain list will be set to default value.

To delete L2 Bridge Domain select it from the L2 Bridge Domain list and click Delete button.

To add L3 Flood Domain, click the Add button next to the L3 Flood Domain list. This will display the form for adding a new L3 Flood Domain. After filling L3 Flood Domain attributes click Save button. After saving of L3 Flood Domain, form will be closed and the L3 Flood Domain list will be set to default value.

To view an existing L3 Flood Domain, select the L3 Flood Domain from the select box L3 Flood Domain list. The view form is read-only and can be closed by clicking cross mark in the top right of the form.

If user wants to edit selected L3 Flood Domain, click the Edit button, which will display the edit form for selected L3 Flood Domain. After editing click the Save button to save selected L3 Flood Domain. After saving of L3 Flood Domain the edit form will be closed and the L3 Flood Domain list will be set to default value.

To delete L3 Flood Domain select it from the L3 Flood Domain list and click Delete button.

To add Subnet click the Add button next to the Subnet list. This will display the form for adding a new Subnet. After filling Subnet attributes click Save button. After saving of Subnet, form will be closed and the Subnet list will be set to default value.

To view an existing Subnet, select the Subnet from the select box Subnet list. The view form is read-only and can be closed by clicking cross mark in the top right of the form.

If user wants to edit selected Subnet, click the Edit button, which will display the edit form for selected Subnet. After editing click the Save button to save selected Subnet. After saving of Subnet the edit form will be closed and the Subnet list will be set to default value.

To delete Subnet select it from the Subnet list and click Delete button.

Classifiers

To add Classifier, click the Add button next to the Classifier list. This will display the form for adding a new Classifier. After filling Classifier attributes click Save button. After saving of Classifier, form will be closed and the Classifier list will be set to default value.

To view an existing Classifier, select the Classifier from the select box Classifier list. The view form is read-only and can be closed by clicking cross mark in the top right of the form.

If you want to edit selected Classifier, click the Edit button, which will display the edit form for selected Classifier. After editing click the Save button to save selected Classifier. After saving of Classifier the edit form will be closed and the Classifier list will be set to default value.

To delete Classifier select it from the Classifier list and click Delete button.

Actions

To add Action, click the Add button next to the Action list. This will display the form for adding a new Action. After filling Action attributes click Save button. After saving of Action, form will be closed and the Action list will be set to default value.

To view an existing Action, select the Action from the select box Action list. The view form is read-only and can be closed by clicking cross mark in the top right of the form.

If user wants to edit selected Action, click the Edit button, which will display the edit form for selected Action. After editing click the Save button to save selected Action. After saving of Action the edit form will be closed and the Action list will be set to default value.

To delete Action select it from the Action list and click Delete button.

Endpoint

To add Endpoint, click the Add button next to the Endpoint list. This will display the form for adding a new Endpoint. To add EndpointGroup assignment click the Plus button next to the label EndpointGroups. To add Condition click Plus button next to the label Condition. To add L3 Address click the Plus button next to the L3 Addresses label. After filling Endpoint attributes click Save button. After saving of Endpoint, form will be closed and the Endpoint list will be set to default value.

To view an existing Endpoint just, the Endpoint from the select box Endpoint list. The view form is read-only and can be closed by clicking cross mark in the top right of the form.

If you want to edit selected Endpoint, click the Edit button, which will display the edit form for selected Endpoint. After editing click the Save button to save selected Endpoint. After saving of Endpoint the edit form will be closed and the Endpoint list will be set to default value.

To delete Endpoint select it from the Endpoint list and click Delete button.

L3 prefix endpoint

To add L3 prefix endpoint, click the Add button next to the L3 prefix endpoint list. This will display the form for adding a new Endpoint. To add EndpointGroup assignment, click the Plus button next to the label EndpointGroups. To add Condition, click Plus button next to the label Condition. To add L2 gateway click the Plus button next to the L2 gateways label. To add L3 gateway, click the Plus button next to the L3 gateways label. After filling L3 prefix endpoint attributes click Save button. After saving of L3 prefix endpoint, form will be closed and the Endpoint list will be set to default value.

To view an existing L3 prefix endpoint, select the Endpoint from the select box L3 prefix endpoint list. The view form is read-only and can be closed by clicking cross mark in the top right of the form.

If you want to edit selected L3 prefix endpoint, click the Edit button, which will display the edit form for selected L3 prefix endpoint. After editing click the Save button to save selected L3 prefix endpoint. After saving of Endpoint the edit form will be closed and the Endpoint list will be set to default value.

To delete Endpoint select it from the L3 prefix endpoint list and click Delete button.

Wizard

Wizard provides quick method to send basic data to controller necessary for basic usage of GBP application. It is useful in the case that there aren't any data in controller. In the first tab is form for create tenant. The second tab is for CRUD operations with contracts and their sub elements such as subjects, rules, clauses, action refs and classifier refs. The last tab is for CRUD operations with EndpointGroups and their CNS and PNS. Created structure of data is possible to send by clicking on Submit button.

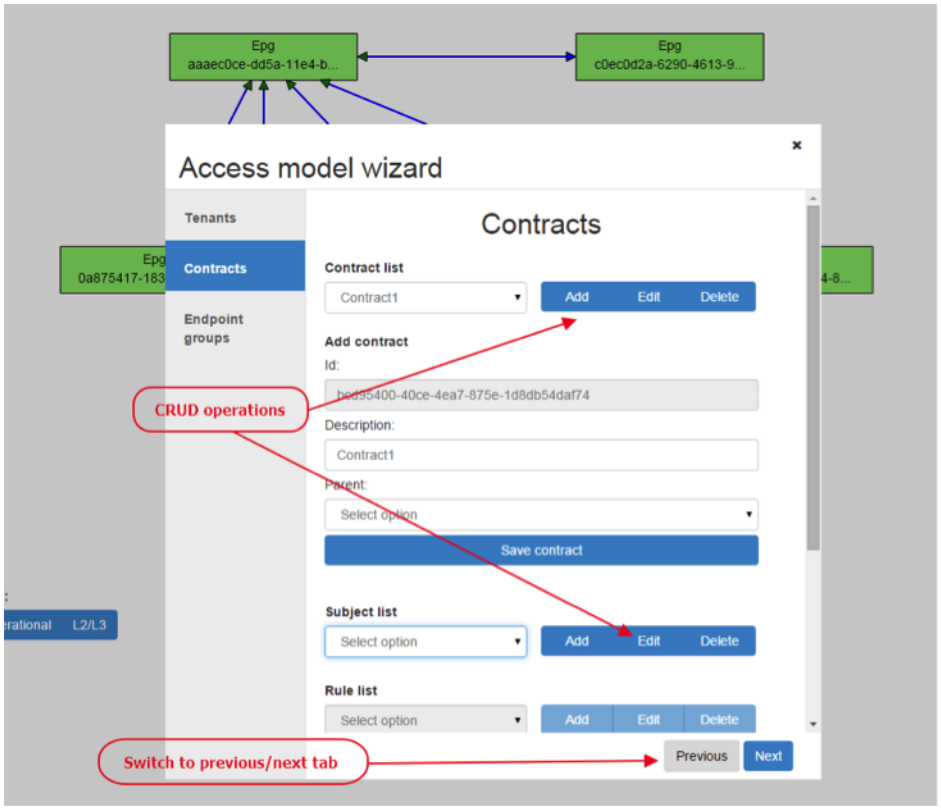


Fig. 1.50: Wizard

Using the GBP API

Please see:

- *Using the GBP OpenFlow Overlay (OfOverlay) renderer*
- *Policy Resolution*
- *Forwarding Model*
- *the **GBP** demo and development environments for tips*

It is recommended to use either:

- *Neutron mapper <gbp-neutron>*
- *the UX*

If the REST API must be used, and the above resources are not sufficient:

- feature:install odl-dluxapps-yangui
- browse to: `http://<odl-controller>:8181/index.html` and select YangUI from the left menu.

to explore the various **GBP** REST options

Using OpenStack with GBP

Overview

This section is for Application Developers and Network Administrators who are looking to integrate Group Based Policy with OpenStack.

To enable the **GBP** Neutron Mapper feature, at the Karaf console:

```
feature:install odl-groupbasedpolicy-neutronmapper
```

Neutron Mapper has the following dependencies that are automatically loaded:

```
odl-neutron-service
```

Neutron Northbound implementing REST API used by OpenStack

```
odl-groupbasedpolicy-base
```

Base **GBP** feature set, such as policy resolution, data model etc.

```
odl-groupbasedpolicy-ofoverlay
```

REST calls from OpenStack Neutron are by the Neutron NorthBound project.

GBP provides the implementation of the Neutron V2.0 API.

Features

List of supported Neutron entities:

- Port
- Network

  - Standard Internal
  - External provider L2/L3 network
- Subnet
- Security-groups
- Routers
  - Distributed functionality with local routing per compute
  - External gateway access per compute node (dedicated port required)
  - Multiple routers per tenant
- FloatingIP NAT
- IPv4/IPv6 support

The mapping of Neutron entities to **GBP** entities is as follows:

Neutron Port

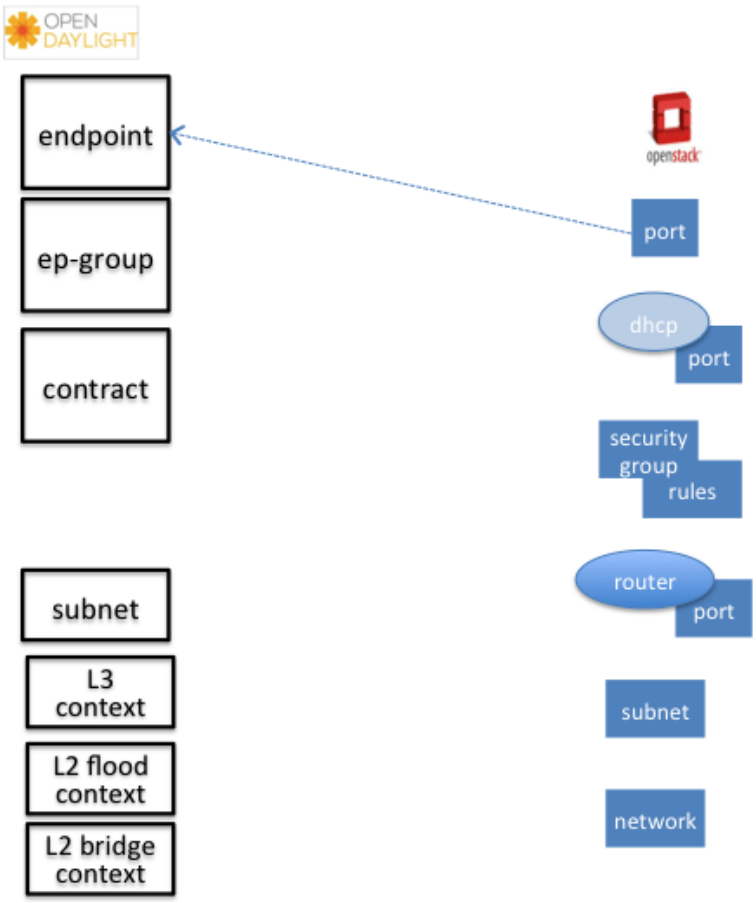


Fig. 1.51: Neutron Port

The Neutron port is mapped to an endpoint.

The current implementation supports one IP address per Neutron port.

An endpoint and L3-endpoint belong to multiple EndpointGroups if the Neutron port is in multiple Neutron Security Groups.

The key for endpoint is L2-bridge-domain obtained as the parent of L2-flood-domain representing Neutron network. The MAC address is from the Neutron port. An L3-endpoint is created based on L3-context (the parent of the L2-bridge-domain) and IP address of Neutron Port.

Neutron Network

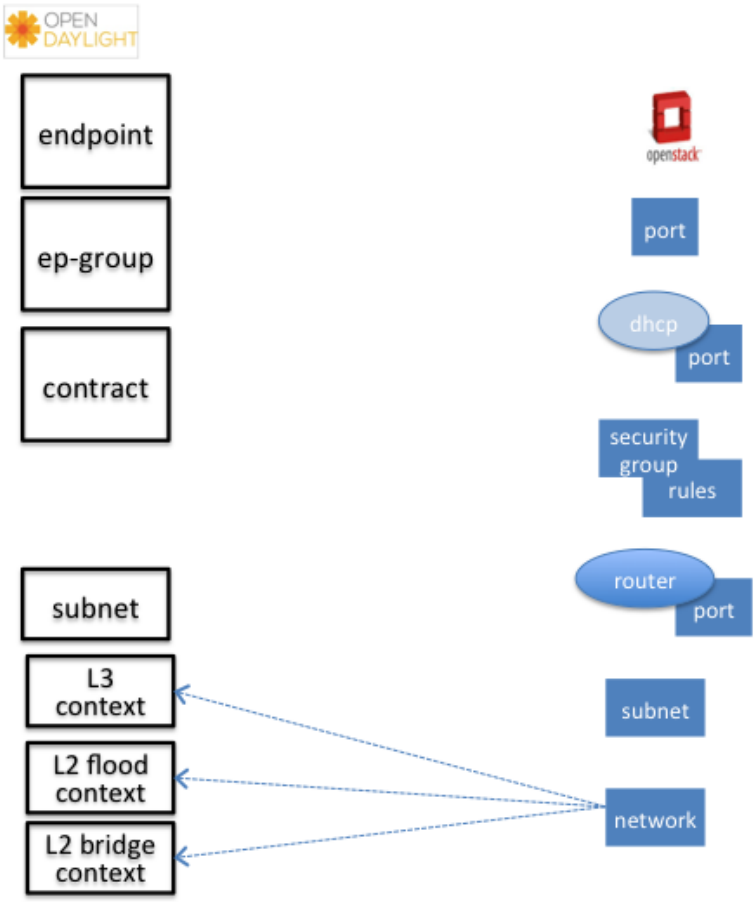


Fig. 1.52: Neutron Network

A Neutron network has the following characteristics:

- defines a broadcast domain
- defines a L2 transmission domain
- defines a L2 name space.

To represent this, a Neutron Network is mapped to multiple **GBP** entities. The first mapping is to an L2 flood-domain to reflect that the Neutron network is one flooding or broadcast domain. An L2-bridge-domain is then associated as the parent of L2 flood-domain. This reflects both the L2 transmission domain as well as the L2 addressing namespace.

The third mapping is to L3-context, which represents the distinct L3 address space. The L3-context is the parent of L2-bridge-domain.

Neutron Subnet

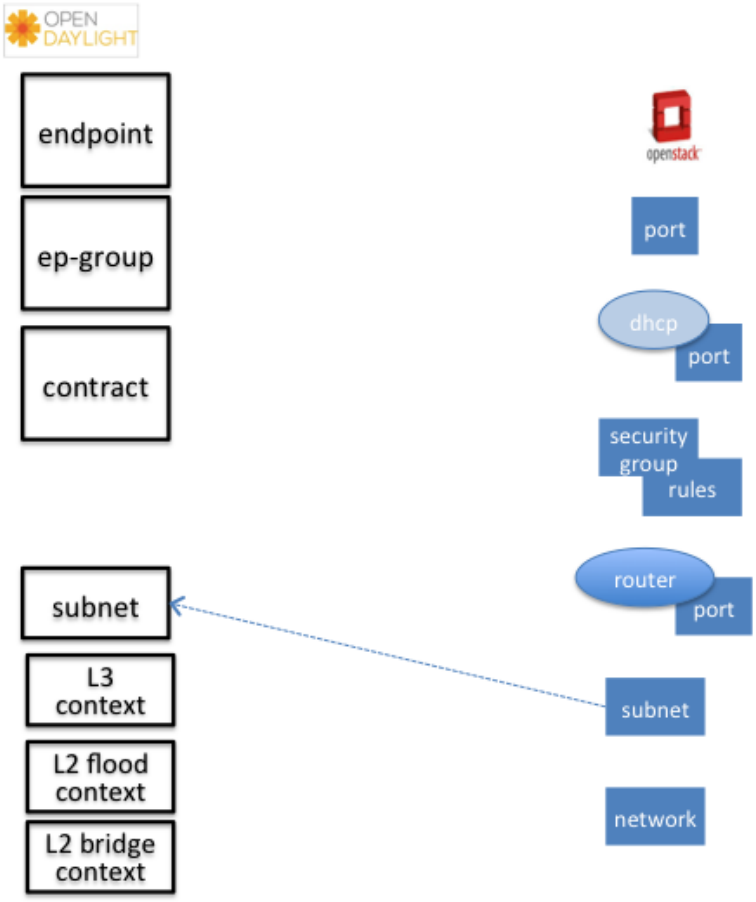


Fig. 1.53: Neutron Subnet

Neutron subnet is associated with a Neutron network. The Neutron subnet is mapped to a **GBP** subnet where the parent of the subnet is L2-flood-domain representing the Neutron network.

Neutron Security Group

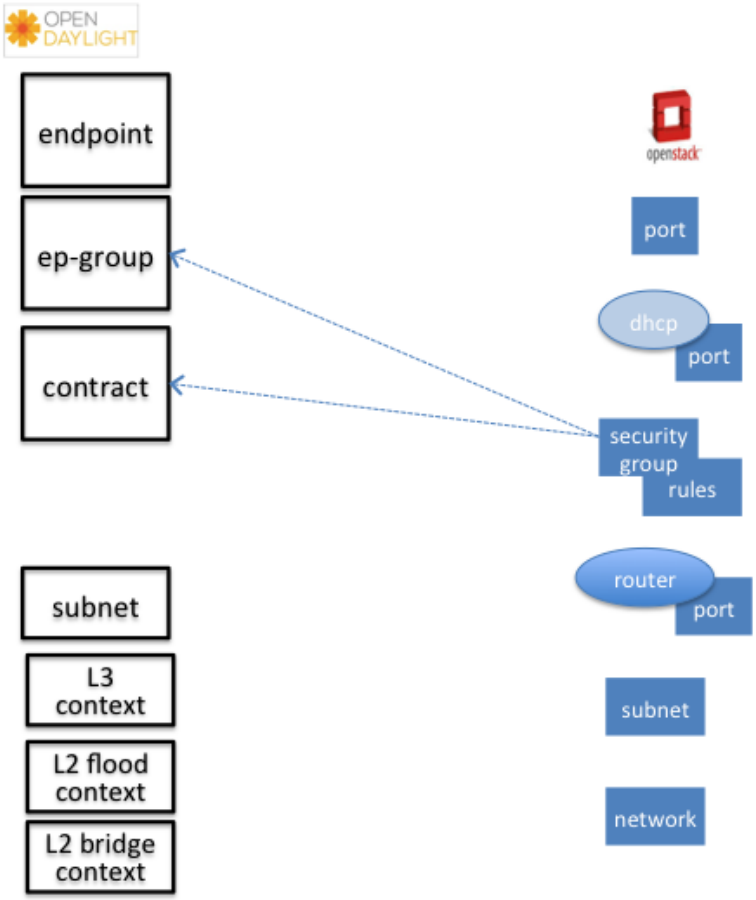


Fig. 1.54: Neutron Security Group and Rules

GBP entity representing Neutron security-group is EndpointGroup.

Infrastructure EndpointGroups

Neutron-mapper automatically creates EndpointGroups to manage key infrastructure items such as:

- DHCP EndpointGroup - contains endpoints representing Neutron DHCP ports
- Router EndpointGroup - contains endpoints representing Neutron router interfaces
- External EndpointGroup - holds L3-endpoints representing Neutron router gateway ports, also associated with FloatingIP ports.

Neutron Security Group Rules

This is the most involved amongst all the mappings because Neutron security-group-rules are mapped to contracts with clauses, subjects, rules, action-refs, classifier-refs, etc. Contracts are used between EndpointGroups representing Neutron Security Groups. For simplification it is important to note that Neutron security-group-rules are similar to a **GBP** rule containing:

- classifier with direction

- action of **allow**.

Neutron Routers

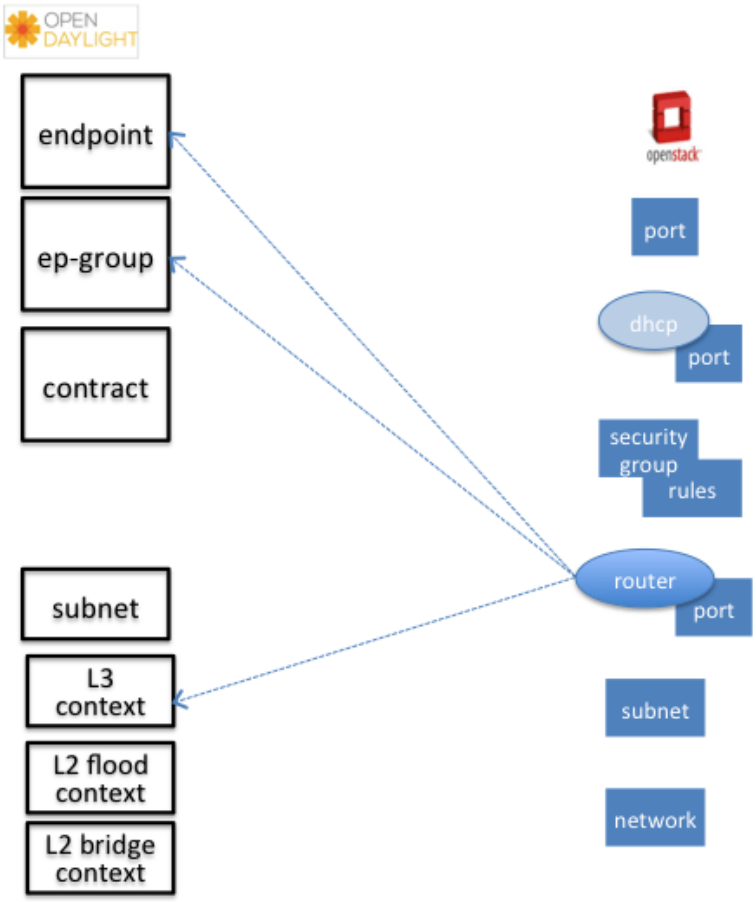


Fig. 1.55: Neutron Router

Neutron router is represented as a L3-context. This treats a router as a Layer3 namespace, and hence every network attached to it a part of that Layer3 namespace.

This allows for multiple routers per tenant with complete isolation.

The mapping of the router to an endpoint represents the router's interface or gateway port.

The mapping to an EndpointGroup represents the internal infrastructure EndpointGroups created by the **GBP** Neutron Mapper

When a Neutron router interface is attached to a network/subnet, that network/subnet and its associated endpoints or Neutron Ports are seamlessly added to the namespace.

Neutron FloatingIP

When associated with a Neutron Port, this leverages the *OfOverlay* renderer's NAT capabilities.

A dedicated *external* interface on each Nova compute host allows for disitributed external access. Each Nova instance associated with a FloatingIP address can access the external network directly without having to route via the Neutron controller, or having to enable any form of Neutron distributed routing functionality.

Assuming the gateway provisioned in the Neutron Subnet command for the external network is reachable, the combination of **GBP** Neutron Mapper and *OfOverlay renderer* will automatically ARP for this default gateway, requiring no user intervention.

Troubleshooting within GBP

Logging level for the mapping functionality can be set for package org.opendaylight.groupbasedpolicy.neutron.mapper. An example of enabling TRACE logging level on Karaf console:

```
log:set TRACE org.opendaylight.groupbasedpolicy.neutron.mapper
```

Neutron mapping example

As an example for mapping can be used creation of Neutron network, subnet and port. When a Neutron network is created 3 **GBP** entities are created: l2-flood-domain, l2-bridge-domain, l3-context.

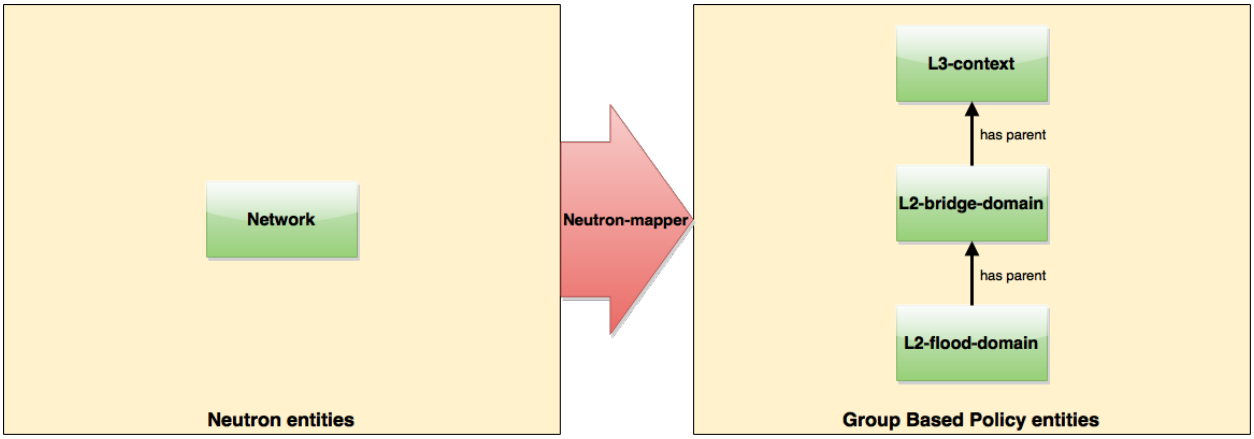


Fig. 1.56: Neutron network mapping

After an subnet is created in the network mapping looks like this.

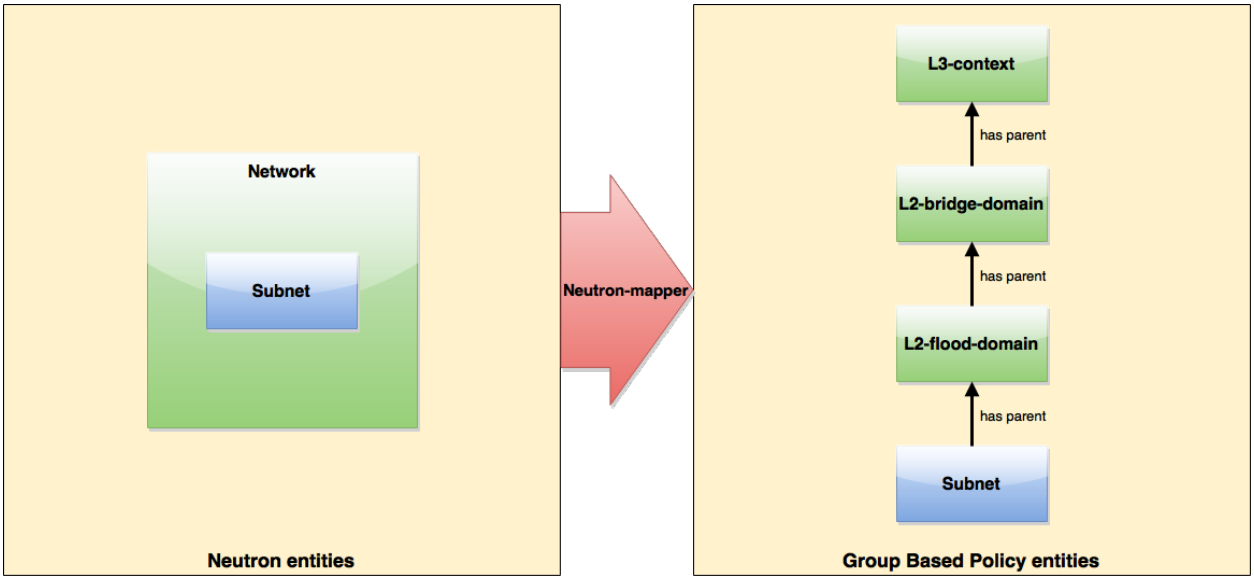


Fig. 1.57: Neutron subnet mapping

If an Neutron port is created in the subnet an endpoint and l3-endpoint are created. The endpoint has key composed from l2-bridge-domain and MAC address from Neutron port. A key of l3-endpoint is compesed from l3-context and IP address. The network containment of endpoint and l3-endpoint points to the subnet.

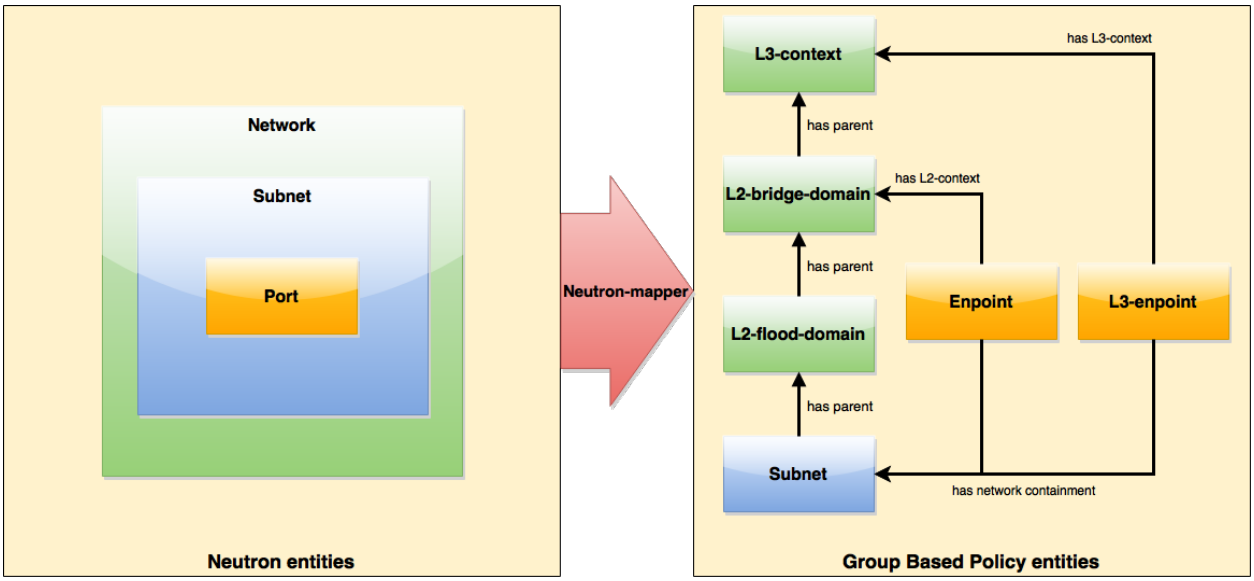


Fig. 1.58: Neutron port mapping

Configuring GBP Neutron

No intervention passed initial OpenStack setup is required by the user.

More information about configuration can be found in our DevStack demo environment on the GBP wiki.

Administering or Managing GBP Neutron

For consistencies sake, all provisioning should be performed via the Neutron API. (CLI or Horizon).

The mapped policies can be augmented via the **GBP** *UX*, to:

- Enable *Service Function Chaining*
- Add endpoints from outside of Neutron i.e. VMs/containers not provisioned in OpenStack
- Augment policies/contracts derived from Security Group Rules
- Overlay additional contracts or groupings

Tutorials

A DevStack demo environment can be found on the GBP wiki.

GBP Renderer manager

Overview

The GBP Renderer manager is an integral part of **GBP** base module. It dispatches information about endpoints' policy configuration to specific device renderer by writing a renderer policy configuration into the registered renderer's policy store.

Installing and Pre-requisites

Renderer manager is integrated into GBP base module, so no additional installation is required.

Architecture

Renderer manager gets data notifications about:

- Endoints (base-endpoint.yang)
- EndpointLocations (base-endpoint.yang)
- ResolvedPolicies (resolved-policy.yang)
- Forwarding (forwarding.yang)

Based on data from notifications it creates a configuration task for specific renderers by writing a renderer policy configuration into the registered renderer's policy store. Configuration is stored to CONF data store as Renderers (renderer.yang).

Configuration is signed with version number which is incremented by every change. All renderers are supposed to be on the same version. Renderer manager waits for all renderers to respond with version update in OPER data store. After a version of every renderer in OPER data store has the same value as the one in CONF data store, renderer manager moves to the next configuration with incremented version.

GBP Location manager

Overview

Location manager monitors information about Endpoint Location providers (see endpoint-location-provider.yang) and manages Endpoint locations in OPER data store accordingly.

Installing and Pre-requisites

Location manager is integrated into GBP base module, so no additional installation is required.

Architecture

The endpoint-locations container in OPER data store (see base-endpoint.yang) contains two lists for two types of EP location, namely address-endpoint-location and containment-endpoint-location. LocationResolver is a class that processes Location providers in CONF data store and puts location information to OPER data store.

When a new Location provider is created in CONF data store, its Address EP locations are being processed first, and their info is stored locally in accordance with processed Location provider's priority. Then a location of type

"absolute" with the highest priority is selected for an EP, and is put in OPER data store. If Address EP locations contain locations of type "relative", those are put to OPER data store.

If current Location provider contains Containment EP locations of type "relative", then those are put to OPER data store.

Similarly, when a Location provider is deleted, information of its locations is removed from the OPER data store.

Using the GBP OpenFlow Overlay (OfOverlay) renderer

Overview

The OpenFlow Overlay (OfOverlay) feature enables the OpenFlow Overlay renderer, which creates a network virtualization solution across nodes that host Open vSwitch software switches.

Installing and Pre-requisites

From the Karaf console in OpenDaylight:

```
feature:install odl-groupbasedpolicy-ofoverlay
```

This renderer is designed to work with OpenVSwitch (OVS) 2.1+ (although 2.3 is strongly recommended) and OpenFlow 1.3.

When used in conjunction with the *Neutron Mapper feature* no extra OfOverlay specific setup is required.

When this feature is loaded "standalone", the user is required to configure infrastructure, such as

- instantiating OVS bridges,
- attaching hosts to the bridges,
- and creating the VXLAN/VXLAN-GPE tunnel ports on the bridges.

The **GBP** OfOverlay renderer also supports a table offset option, to offset the pipeline post-table 0. The value of table offset is stored in the config datastore and it may be rewritten at runtime.

```
PUT http://{{controllerIp}}:8181/restconf/config/ofoverlay:of-overlay-config
{
    "of-overlay-config": {
        "gbp-ofoverlay-table-offset": 6
    }
}
```

The default value is set by changing: <gbp-ofoverlay-table-offset>0</gbp-ofoverlay-table-offset>

in file: distribution-karaf/target/assembly/etc/opendaylight/karaf/15-groupbasedpolicy-ofoverlay.xml

To avoid overwriting runtime changes, the default value is used only when the OfOverlay renderer starts and no other value has been written before.

OpenFlow Overlay Architecture

These are the primary components of **GBP**. The OfOverlay components are highlighted in red.

In terms of the inner components of the **GBP** OfOverlay renderer:

OfOverlay Renderer

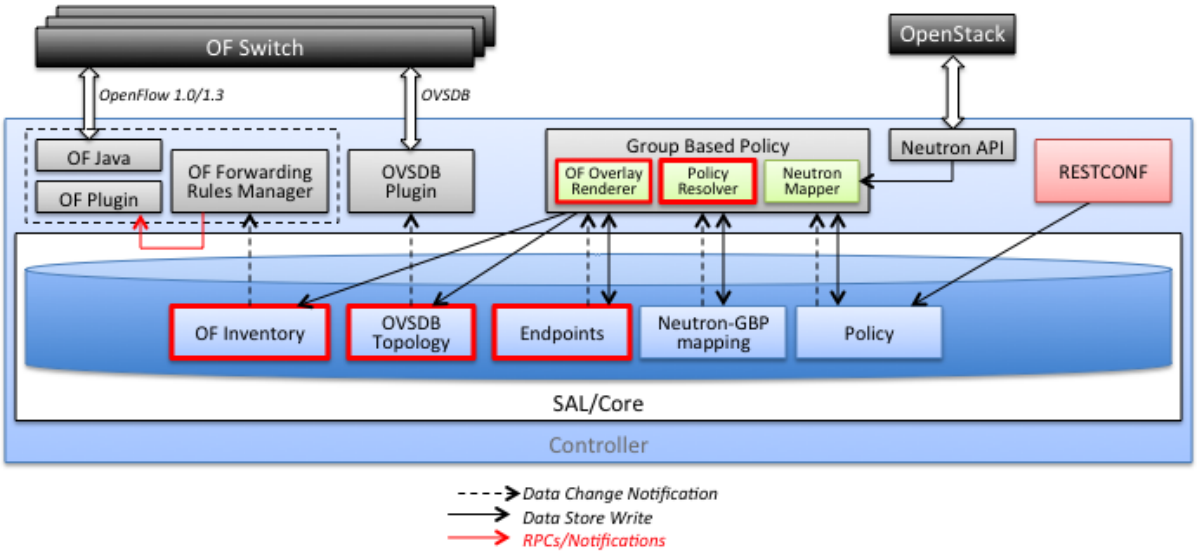


Fig. 1.59: OfOverlay within **GBP**

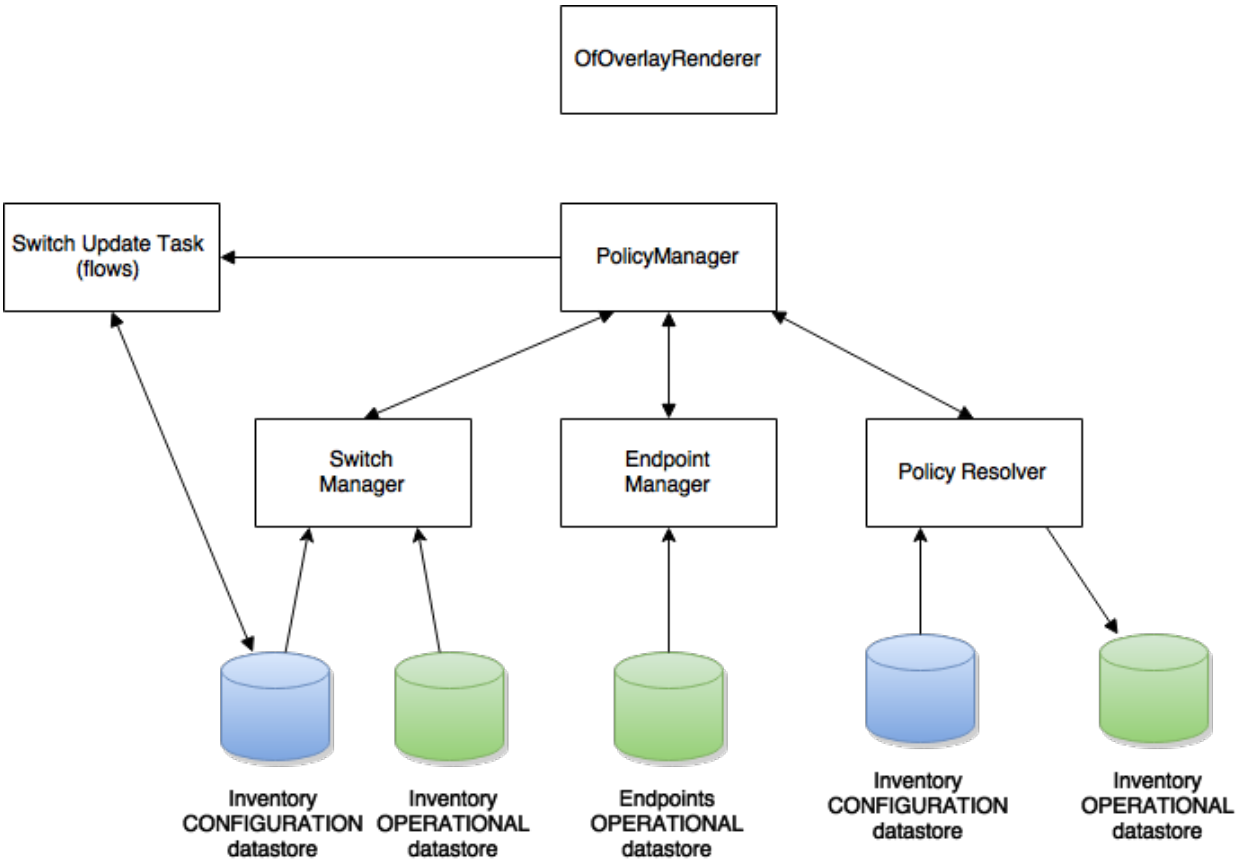


Fig. 1.60: OfOverlay expanded view:

Launches components below:

Policy Resolver

Policy resolution is completely domain independent, and the OfOverlay leverages process policy information internally. See Policy Resolution process.

It listens to inputs to the *Tenants* configuration datastore, validates tenant input, then writes this to the Tenants operational datastore.

From there an internal notification is generated to the PolicyManager.

In the next release, this will be moving to a non-renderer specific location.

Endpoint Manager

The endpoint repository operates in **orchestrated** mode. This means the user is responsible for the provisioning of endpoints via:

- *UX/GUI*
- REST API

Note: When using the *Neutron mapper* feature, everything is managed transparently via Neutron.

The Endpoint Manager is responsible for listening to Endpoint repository updates and notifying the Switch Manager when a valid Endpoint has been registered.

It also supplies utility functions to the flow pipeline process.

Switch Manager

The Switch Manager is purely a state manager.

Switches are in one of 3 states:

- DISCONNECTED
- PREPARING
- READY

Ready is denoted by a connected switch:

- having a tunnel interface
- having at least one endpoint connected.

In this way **GBP** is not writing to switches it has no business to.

Preparing simply means the switch has a controller connection but is missing one of the above *complete and necessary* conditions

Disconnected means a previously connected switch is no longer present in the Inventory operational datastore.

The OfOverlay leverages Nicira registers as follows:

- REG0 = Source EndpointGroup + Tenant ordinal
- REG1 = Source Conditions + Tenant ordinal
- REG2 = Destination EndpointGroup + Tenant ordinal
- REG3 = Destination Conditions + Tenant ordinal
- REG4 = Bridge Domain + Tenant ordinal

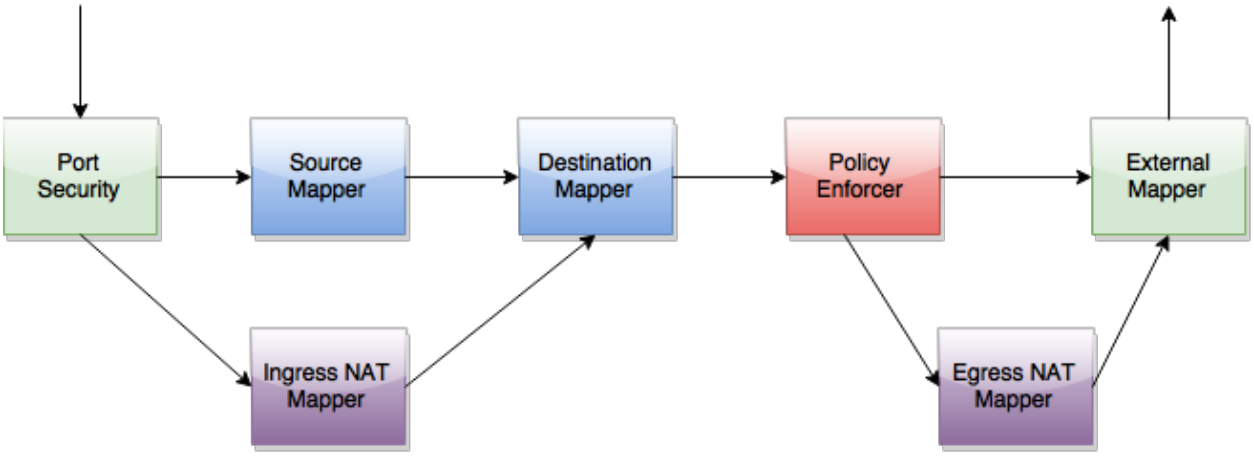


Fig. 1.61: OfOverlay Flow Pipeline

- REG5 = Flood Domain + Tenant ordinal
- REG6 = Layer 3 Context + Tenant ordinal

Port Security

Table 0 of the OpenFlow pipeline. Responsible for ensuring that only valid connections can send packets into the pipeline:

```
cookie=0x0, <snip> , priority=200,in_port=3 actions=goto_table:2
cookie=0x0, <snip> , priority=200,in_port=1 actions=goto_table:1
cookie=0x0, <snip> , priority=121,arp,in_port=5,dl_src=fa:16:3e:d5:b9:8d,arp_spa=10.1.
↪1.3 actions=goto_table:2
cookie=0x0, <snip> , priority=120,ip,in_port=5,dl_src=fa:16:3e:d5:b9:8d,nw_src=10.1.1.
↪3 actions=goto_table:2
cookie=0x0, <snip> , priority=115,ip,in_port=5,dl_src=fa:16:3e:d5:b9:8d,nw_dst=255.
↪255.255.255 actions=goto_table:2
cookie=0x0, <snip> , priority=112,ipv6 actions=drop
cookie=0x0, <snip> , priority=111, ip actions=drop
cookie=0x0, <snip> , priority=110,arp actions=drop
cookie=0x0, <snip> ,in_port=5,dl_src=fa:16:3e:d5:b9:8d actions=goto_table:2
cookie=0x0, <snip> , priority=1 actions=drop
```

Ingress from tunnel interface, go to Table *Source Mapper*:

```
cookie=0x0, <snip> , priority=200,in_port=3 actions=goto_table:2
```

Ingress from outside, goto Table *Ingress NAT Mapper*:

```
cookie=0x0, <snip> , priority=200,in_port=1 actions=goto_table:1
```

ARP from Endpoint, go to Table *Source Mapper*:

```
cookie=0x0, <snip> , priority=121,arp,in_port=5,dl_src=fa:16:3e:d5:b9:8d,arp_spa=10.1.
↪1.3 actions=goto_table:2
```

IPv4 from Endpoint, go to Table *Source Mapper*:

```
cookie=0x0, <snip> , priority=120,ip,in_port=5,dl_src=fa:16:3e:d5:b9:8d,nw_src=10.1.1.
↪3 actions=goto_table:2
```

DHCP DORA from Endpoint, go to Table *Source Mapper*:

```
cookie=0x0, <snip> , priority=115,ip,in_port=5,dl_src=fa:16:3e:d5:b9:8d,nw_dst=255.
↪255.255.255 actions=goto_table:2
```

Series of DROP tables with priority set to capture any non-specific traffic that should have matched above:

```
cookie=0x0, <snip> , priority=112,ipv6 actions=drop
cookie=0x0, <snip> , priority=111, ip actions=drop
cookie=0x0, <snip> , priority=110,arp actions=drop
```

"L2" catch all traffic not identified above:

```
cookie=0x0, <snip> ,in_port=5,dl_src=fa:16:3e:d5:b9:8d actions=goto_table:2
```

Drop Flow:

```
cookie=0x0, <snip> , priority=1 actions=drop
```

Ingress NAT Mapper

Table *offset* +1.

ARP responder for external NAT address:

```
cookie=0x0, <snip> , priority=150,arp,arp_tpa=192.168.111.51,arp_op=1␣
↪actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:58:c3:dd->eth_
↪src,load:0x2->NXM_OF_ARP_OP[],move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],
↪load:0xfa163e58c3dd->NXM_NX_ARP_SHA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],
↪load:0xc0a86f33->NXM_OF_ARP_SPA[],IN_PORT
```

Translate from Outside to Inside and perform same functions as SourceMapper.

```
cookie=0x0, <snip> , priority=100,ip,nw_dst=192.168.111.51 actions=set_field:10.1.1.2-
↪>ip_dst,set_field:fa:16:3e:58:c3:dd->eth_dst,load:0x2->NXM_NX_REG0[],load:0x1->NXM_
↪NX_REG1[],load:0x4->NXM_NX_REG4[],load:0x5->NXM_NX_REG5[],load:0x7->NXM_NX_REG6[],
↪load:0x3->NXM_NX_TUN_ID[0..31],goto_table:3
```

Source Mapper

Table *offset* +2.

Determines based on characteristics from the ingress port, which:

- EndpointGroup(s) it belongs to
- Forwarding context
- Tunnel VNID ordinal

Establishes tunnels at valid destination switches for ingress.

Ingress Tunnel established at remote node with VNID Ordinal that maps to Source EPG, Forwarding Context etc:

```
cookie=0x0, <snip>, priority=150,tun_id=0xd,in_port=3 actions=load:0xc->NXM_NX_REG0[],
↪load:0xffffff->NXM_NX_REG1[],load:0x4->NXM_NX_REG4[],load:0x5->NXM_NX_REG5[],
↪load:0x7->NXM_NX_REG6[],goto_table:3
```

Maps endpoint to Source EPG, Forwarding Context based on ingress port, and MAC:

```
cookie=0x0, <snip> , priority=100,in_port=5,dl_src=fa:16:3e:b4:b4:b1 actions=load:0xc-
↪>NXM_NX_REG0[],load:0x1->NXM_NX_REG1[],load:0x4->NXM_NX_REG4[],load:0x5->NXM_NX_
↪REG5[],load:0x7->NXM_NX_REG6[],load:0xd->NXM_NX_TUN_ID[0..31],goto_table:3
```

Generic drop:

```
cookie=0x0, duration=197.622s, table=2, n_packets=0, n_bytes=0, priority=1␣
↪actions=drop
```

Destination Mapper

Table *offset* +3.

Determines based on characteristics of the endpoint:

- EndpointGroup(s) it belongs to
- Forwarding context
- Tunnel Destination value

Manages routing based on valid ingress nodes ARP'ing for their default gateway, and matches on either gateway MAC or destination endpoint MAC.

ARP for default gateway for the 10.1.1.0/24 subnet:

```
cookie=0x0, <snip> , priority=150,arp,reg6=0x7,arp_tpa=10.1.1.1,arp_op=1␣
↪actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:28:4c:82->eth_
↪src,load:0x2->NXM_OF_ARP_OP[],move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],
↪load:0xfa163e284c82->NXM_NX_ARP_SHA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],
↪load:0xa010101->NXM_OF_ARP_SPA[],IN_PORT
```

Broadcast traffic destined for GroupTable:

```
cookie=0x0, <snip> , priority=140,reg5=0x5,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00␣
↪actions=load:0x5->NXM_NX_TUN_ID[0..31],group:5
```

Layer3 destination matching flows, where priority=100+masklength. Since **GBP** now support L3Prefix endpoint, we can set default routes etc:

```
cookie=0x0, <snip>, priority=132,ip,reg6=0x7,dl_dst=fa:16:3e:b4:b4:b1,nw_dst=10.1.1.3␣
↪actions=load:0xc->NXM_NX_REG2[],load:0x1->NXM_NX_REG3[],load:0x5->NXM_NX_REG7[],set_
↪field:fa:16:3e:b4:b4:b1->eth_dst,dec_ttl,goto_table:4
```

Layer2 destination matching flows, designed to be caught only after last IP flow (lowest priority IP flow is 100):

```
cookie=0x0, duration=323.203s, table=3, n_packets=4, n_bytes=168, priority=50,
↪reg4=0x4,dl_dst=fa:16:3e:58:c3:dd actions=load:0x2->NXM_NX_REG2[],load:0x1->NXM_NX_
↪REG3[],load:0x2->NXM_NX_REG7[],goto_table:4
```

General drop flow: cookie=0x0, duration=323.207s, table=3, n_packets=6, n_bytes=588, priority=1 actions=drop

Policy Enforcer

Table *offset* +4.

Once the Source and Destination EndpointGroups are assigned, policy is enforced based on resolved rules.

In the case of *Service Function Chaining*, the encapsulation and destination for traffic destined to a chain, is discovered and enforced.

Policy flow, allowing IP traffic between EndpointGroups:

```
cookie=0x0, <snip> , priority=64998,ip,reg0=0x8,reg1=0x1,reg2=0xc,reg3=0x1␣
↪actions=goto_table:5
```

Egress NAT Mapper

Table *offset* +5.

Performs NAT function before Egressing OVS instance to the underlay network.

Inside to Outside NAT translation before sending to underlay:

```
cookie=0x0, <snip> , priority=100,ip,reg6=0x7,nw_src=10.1.1.2 actions=set_field:192.
↪168.111.51->ip_src,goto_table:6
```

External Mapper

Table *offset* +6.

Manages post-policy enforcement for endpoint specific destination effects. Specifically for *Service Function Chaining*, which is why we can support both symmetric and asymmetric chains and distributed ingress/egress classification.

Generic allow:

```
cookie=0x0, <snip>, priority=100 actions=output:NXM_NX_REG7[]
```

Configuring OpenFlow Overlay via REST

Note: Please see the *UX* section on how to configure **GBP** via the GUI.

Endpoint

```
POST http://{{controllerIp}}:8181/restconf/operations/endpoint:register-endpoint
{
    "input": {
        "endpoint-group": "<epg0>",
        "endpoint-groups" : ["<epg1>","<epg2>"],
        "network-containment" : "<fowarding-model-context1>",
        "l2-context": "<bridge-domain1>",
        "mac-address": "<mac1>",
        "l3-address": [
            {
                "ip-address": "<ipaddress1>",
                "l3-context": "<l3_context1>"
            }
        ],
        "*ofoverlay:port-name*": "<ovs port name>",
        "tenant": "<tenant1>"
    }
}
```

Note: The usage of "port-name" preceded by "ofoverlay". In OpenDaylight, base datastore objects can be *augmented*. In **GBP**, the base endpoint model has no renderer specifics, hence can be leveraged across multiple renderers.

OVS Augmentations to Inventory

```
PUT http://{{controllerIp}}:8181/restconf/config/opendaylight-inventory:nodes/
{
    "opendaylight-inventory:nodes": {
        "node": [
            {
                "id": "openflow:123456",
                "ofoverlay:tunnel": [
                    {
                        "tunnel-type": "overlay:tunnel-type-vxlan",
                        "ip": "<ip_address_of_ovs>",
                        "port": 4789,
                        "node-connector-id": "openflow:123456:1"
                    }
                ]
            },
            {
                "id": "openflow:654321",
                "ofoverlay:tunnel": [
                    {
                        "tunnel-type": "overlay:tunnel-type-vxlan",
                        "ip": "<ip_address_of_ovs>",
                        "port": 4789,
                        "node-connector-id": "openflow:654321:1"
                    }
                ]
            }
        ]
    }
}
```

Tenants see *Policy Resolution* and *Forwarding Model* for details:

```
{
  "policy:tenant": {
    "contract": [
      {
        "clause": [
          {
            "name": "allow-http-clause",
            "subject-refs": [
              "allow-http-subject",
              "allow-icmp-subject"
            ]
          }
        ],
        "id": "<id>",
        "subject": [
          {
            "name": "allow-http-subject",
            "rule": [
              {
                "classifier-ref": [
                  {
                    "direction": "in",
                    "name": "http-dest"
                  },
                  {
                    "direction": "out",
```

```
                    "name": "http-src"
                  }
                ],
                "action-ref": [
                  {
                    "name": "allow1",
                    "order": 0
                  }
                ],
                "name": "allow-http-rule"
              }
            ]
          },
          {
            "name": "allow-icmp-subject",
            "rule": [
              {
                "classifier-ref": [
                  {
                    "name": "icmp"
                  }
                ],
                "action-ref": [
                  {
                    "name": "allow1",
                    "order": 0
                  }
                ],
                "name": "allow-icmp-rule"
              }
            ]
          }
        ]
      }
    ],
    "endpoint-group": [
      {
        "consumer-named-selector": [
          {
            "contract": [
              "<id>"
            ],
            "name": "<name>"
          }
        ],
        "id": "<id>",
        "provider-named-selector": []
      },
      {
        "consumer-named-selector": [],
        "id": "<id>",
        "provider-named-selector": [
          {
            "contract": [
              "<id>"
            ],
            "name": "<name>"
          }
```

```
        ]
      }
    ],
    "id": "<id>",
    "l2-bridge-domain": [
      {
        "id": "<id>",
        "parent": "<id>"
      }
    ],
    "l2-flood-domain": [
      {
        "id": "<id>",
        "parent": "<id>"
      },
      {
        "id": "<id>",
        "parent": "<id>"
      }
    ],
    "l3-context": [
      {
        "id": "<id>"
      }
    ],
    "name": "GBPPOC",
    "subject-feature-instances": {
      "classifier-instance": [
        {
          "classifier-definition-id": "<id>",
          "name": "http-dest",
          "parameter-value": [
            {
              "int-value": "6",
              "name": "proto"
            },
            {
              "int-value": "80",
              "name": "destport"
            }
          ]
        },
        {
          "classifier-definition-id": "<id>",
          "name": "http-src",
          "parameter-value": [
            {
              "int-value": "6",
              "name": "proto"
            },
            {
              "int-value": "80",
              "name": "sourceport"
            }
          ]
        },
        {
          "classifier-definition-id": "<id>",
```

```
          "name": "icmp",
          "parameter-value": [
            {
              "int-value": "1",
              "name": "proto"
            }
          ]
        }
      ],
      "action-instance": [
        {
          "name": "allow1",
          "action-definition-id": "<id>"
        }
      ]
    },
    "subnet": [
      {
        "id": "<id>",
        "ip-prefix": "<ip_prefix>",
        "parent": "<id>",
        "virtual-router-ip": "<ip address>"
      },
      {
        "id": "<id>",
        "ip-prefix": "<ip prefix>",
        "parent": "<id>",
        "virtual-router-ip": "<ip address>"
      }
    ]
  }
}
```

Tutorials

Comprehensive tutorials, along with a demonstration environment leveraging Vagrant can be found on the GBP wiki

Using the GBP eBPF IO Visor Agent renderer

Overview

The IO Visor renderer feature enables container endpoints (e.g. Docker, LXC) to leverage GBP policies.

The renderer interacts with a IO Visor module from the Linux Foundation IO Visor project.

Installing and Pre-requisites

From the Karaf console in OpenDaylight:

```
feature:install odl-groupbasedpolicy-iovisor odl-restconf
```

Installation details, usage, and other information for the IO Visor GBP module can be found here: IO Visor github repo for IO Modules

Using the GBP FaaS renderer

Overview

The FaaS renderer feature enables leveraging the FaaS project as a GBP renderer.

Installing and Pre-requisites

From the Karaf console in OpenDaylight:

```
feature:install odl-groupbasedpolicy-faas
```

More information about FaaS can be found here: https://wiki.opendaylight.org/view/FaaS:GBPIntegration

Using Service Function Chaining (SFC) with GBP Neutron Mapper and OfOverlay

Overview

Please refer to the Service Function Chaining project for specifics on SFC provisioning and theory.

GBP allows for the use of a chain, by name, in policy.

This takes the form of an *action* in **GBP**.

Using the *GBP demo and development environment* as an example:

In the topology above, a symmetrical chain between H35_2 and H36_3 could take path:

H35_2 to sw1 to sff1 to sf1 to sff1 to sff2 to sf2 to sff2 to sw6 to H36_3

If symmetric chaining was desired, the return path is:

If asymmetric chaining was desired, the return path could be direct, or an **entirely different chain**.

All these scenarios are supported by the integration.

In the **Subject Feature Instance** section of the tenant config, we define the instances of the classifier definitions for ICMP and HTTP:

```
"subject-feature-instances": {
  "classifier-instance": [
    {
      "name": "icmp",
      "parameter-value": [
        {
          "name": "proto",
          "int-value": 1
        }
      ]
    },
    {
      "name": "http-dest",
      "parameter-value": [
        {
          "int-value": "6",
          "name": "proto"
        },
```

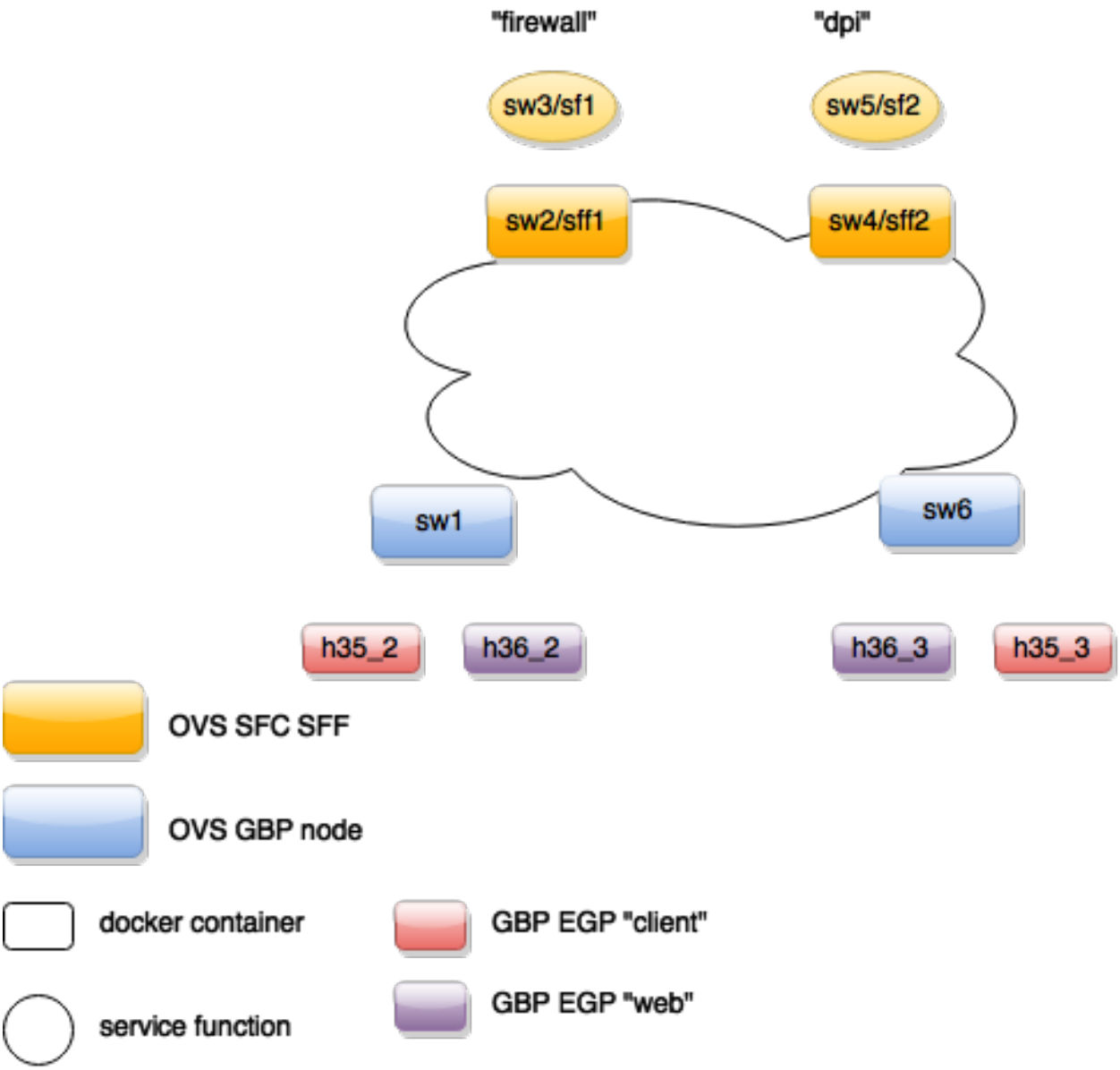


Fig. 1.62: GBP and SFC integration environment

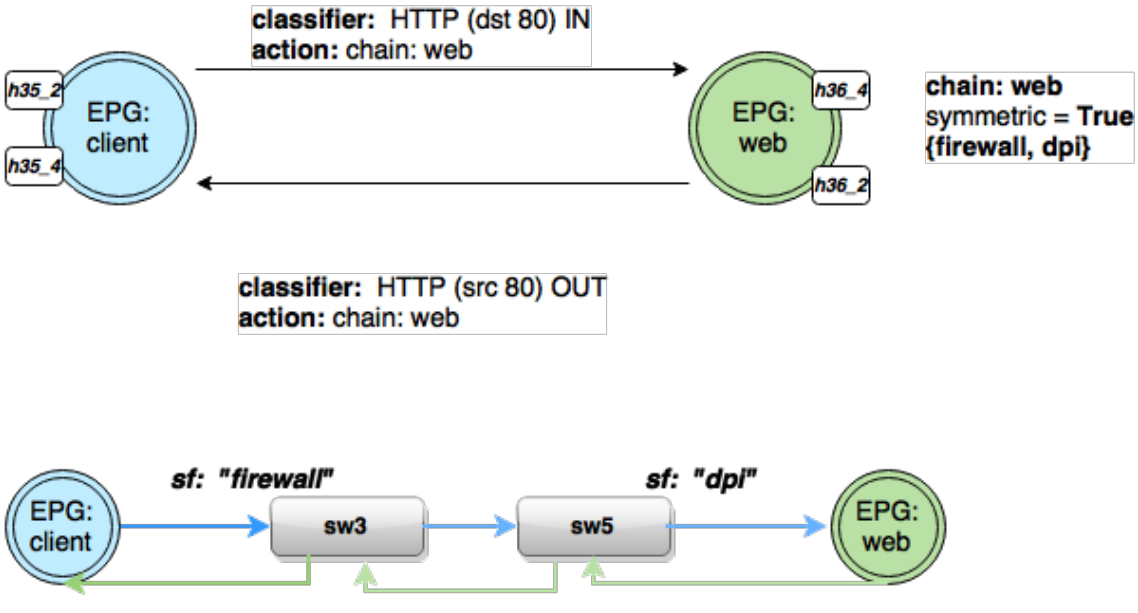


Fig. 1.63: GBP and SFC symmetric chain environment

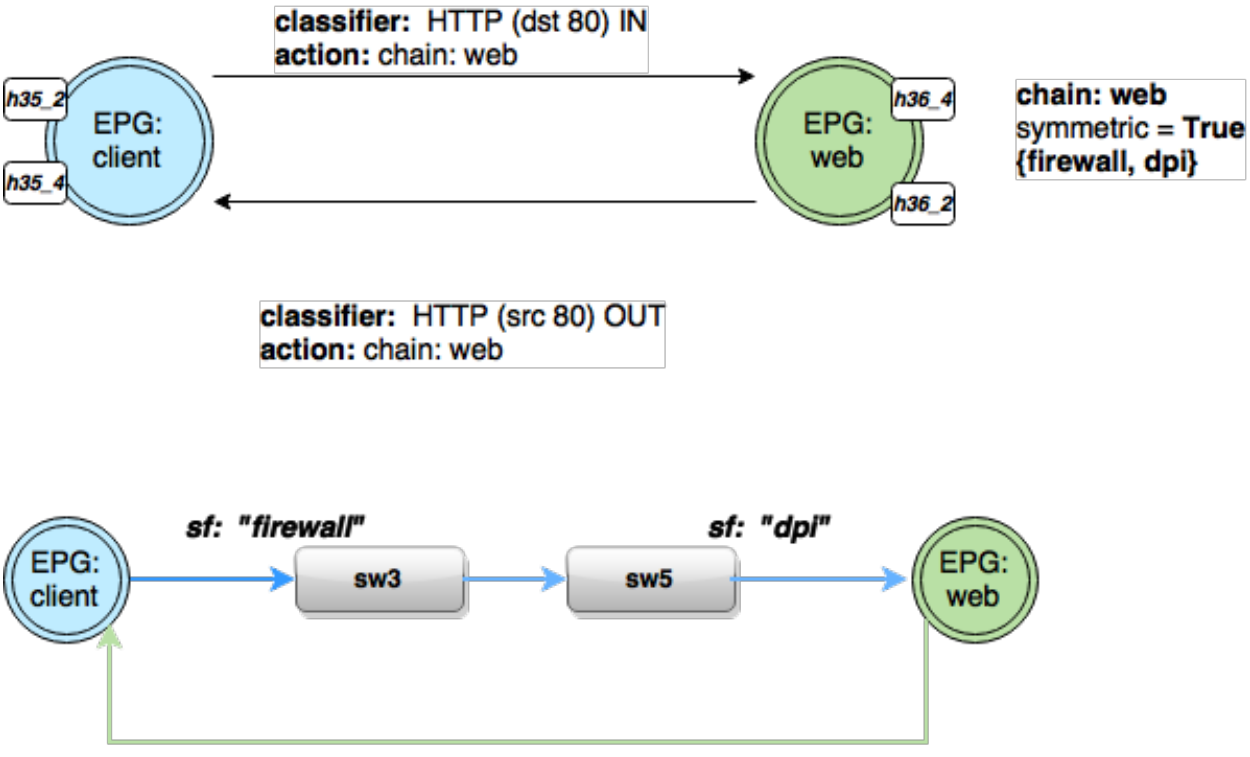


Fig. 1.64: GBP and SFC assymmetric chain environment

```
        {
          "int-value": "80",
          "name": "destport"
        }
      ]
    },
    {
      "name": "http-src",
      "parameter-value": [
        {
          "int-value": "6",
          "name": "proto"
        },
        {
          "int-value": "80",
          "name": "sourceport"
        }
      ]
    }
  ],
```

Then the action instances to associate to traffic that matches classifiers are defined.

Note the *SFC chain name* must exist in SFC, and is validated against the datastore once the tenant configuration is entered, before entering a valid tenant configuration into the operational datastore (which triggers policy resolution).

```
  "action-instance": [
    {
      "name": "chain1",
      "parameter-value": [
        {
          "name": "sfc-chain-name",
          "string-value": "SFCGBP"
        }
      ]
    },
    {
      "name": "allow1",
    }
  ]
},
```

When ICMP is matched, allow the traffic:

```
"contract": [
  {
    "subject": [
      {
        "name": "icmp-subject",
        "rule": [
          {
            "name": "allow-icmp-rule",
            "order" : 0,
            "classifier-ref": [
              {
                "name": "icmp"
              }
            ],
```

```
            "action-ref": [
              {
                "name": "allow1",
                "order": 0
              }
            ]
          }

        ]
      },
```

When HTTP is matched, **in** to the provider of the contract with a TCP destination port of 80 (HTTP) or the HTTP request. The chain action is triggered, and similarly **out** from the provider for traffic with TCP source port of 80 (HTTP), or the HTTP response.

```
{
  "name": "http-subject",
  "rule": [
    {
      "name": "http-chain-rule-in",
      "classifier-ref": [
        {
          "name": "http-dest",
          "direction": "in"
        }
      ],
      "action-ref": [
        {
          "name": "chain1",
          "order": 0
        }
      ]
    },
    {
      "name": "http-chain-rule-out",
      "classifier-ref": [
        {
          "name": "http-src",
          "direction": "out"
        }
      ],
      "action-ref": [
        {
          "name": "chain1",
          "order": 0
        }
      ]
    }
  ]
}
```

To enable asymmetrical chaining, for instance, the user desires that HTTP requests traverse the chain, but the HTTP response does not, the HTTP response is set to *allow* instead of chain:

```
{
  "name": "http-chain-rule-out",
  "classifier-ref": [
    {
```

```
      "name": "http-src",
      "direction": "out"
    }
  ],
  "action-ref": [
    {
      "name": "allow1",
      "order": 0
    }
  ]
}
```

Demo/Development environment

The **GBP** project for this release has two demo/development environments.

- Docker based GBP and GBP+SFC integration Vagrant environment
- DevStack based GBP+Neutron integration Vagrant environment

Demo @ GBP wiki

L2 Switch User Guide

Overview

The L2 Switch project provides Layer2 switch functionality.

L2 Switch Architecture

- Packet Handler
  - Decodes the packets coming to the controller and dispatches them appropriately
- Loop Remover
  - Removes loops in the network
- Arp Handler
  - Handles the decoded ARP packets
- Address Tracker
  - Learns the Addresses (MAC and IP) of entities in the network
- Host Tracker
  - Tracks the locations of hosts in the network
- L2 Switch Main
  - Installs flows on each switch based on network traffic

Configurable parameters in L2 Switch

The sections below give details about the configuration settings for the components that can be configured.

The process to change the configuration has been changed with the introduction of Blueprint in the Boron release. Please refer to *Change configuration in L2 Switch* for an example illustrating how to change the configurations.

Configurable parameters in Loop Remover

- l2switch/loopremover/implementation/src/main/yang/loop-remover-config.yang
  - is-install-lldp-flow
    - "true" means a flow that sends all LLDP packets to the controller will be installed on each switch
    - "false" means this flow will not be installed
    - default value is true
  - lldp-flow-table-id
    - The LLDP flow will be installed on the specified flow table of each switch
    - This field is only relevant when "is-install-lldp-flow" is set to "true"
    - default value is 0
  - lldp-flow-priority
    - The LLDP flow will be installed with the specified priority
    - This field is only relevant when "is-install-lldp-flow" is set to "true"
    - default value is 100
  - lldp-flow-idle-timeout
    - The LLDP flow will timeout (removed from the switch) if the flow doesn't forward a packet for *x* seconds
    - This field is only relevant when "is-install-lldp-flow" is set to "true"
    - default value is 0
  - lldp-flow-hard-timeout
    - The LLDP flow will timeout (removed from the switch) after *x* seconds, regardless of how many packets it is forwarding
    - This field is only relevant when "is-install-lldp-flow" is set to "true"
    - default value is 0
  - graph-refresh-delay
    - A graph of the network is maintained and gets updated as network elements go up/down (i.e. links go up/down and switches go up/down)
    - After a network element going up/down, it waits *graph-refresh-delay* seconds before recomputing the graph
    - A higher value has the advantage of doing less graph updates, at the potential cost of losing some packets because the graph didn't update immediately.
    - A lower value has the advantage of handling network topology changes quicker, at the cost of doing more computation.

        * default value is 1000

Configurable parameters in Arp Handler

- l2switch/arphandler/src/main/yang/arp-handler-config.yang
  - is-proactive-flood-mode
    * "true" means that flood flows will be installed on each switch. With this flood flow, each switch will flood a packet that doesn't match any other flows.
      · Advantage: Fewer packets are sent to the controller because those packets are flooded to the network.
      · Disadvantage: A lot of network traffic is generated.
    * "false" means the previously mentioned flood flows will not be installed. Instead an ARP flow will be installed on each switch that sends all ARP packets to the controller.
      · Advantage: Less network traffic is generated.
      · Disadvantage: The controller handles more packets (ARP requests & replies) and the ARP process takes longer than if there were flood flows.
    * default value is true
  - flood-flow-table-id
    * The flood flow will be installed on the specified flow table of each switch
    * This field is only relevant when "is-proactive-flood-mode" is set to "true"
    * default value is 0
  - flood-flow-priority
    * The flood flow will be installed with the specified priority
    * This field is only relevant when "is-proactive-flood-mode" is set to "true"
    * default value is 2
  - flood-flow-idle-timeout
    * The flood flow will timeout (removed from the switch) if the flow doesn't forward a packet for *x* seconds
    * This field is only relevant when "is-proactive-flood-mode" is set to "true"
    * default value is 0
  - flood-flow-hard-timeout
    * The flood flow will timeout (removed from the switch) after *x* seconds, regardless of how many packets it is forwarding
    * This field is only relevant when "is-proactive-flood-mode" is set to "true"
    * default value is 0
  - arp-flow-table-id
    * The ARP flow will be installed on the specified flow table of each switch
    * This field is only relevant when "is-proactive-flood-mode" is set to "false"
    * default value is 0

  - arp-flow-priority
    - The ARP flow will be installed with the specified priority
    - This field is only relevant when "is-proactive-flood-mode" is set to "false"
    - default value is 1
  - arp-flow-idle-timeout
    - The ARP flow will timeout (removed from the switch) if the flow doesn't forward a packet for *x* seconds
    - This field is only relevant when "is-proactive-flood-mode" is set to "false"
    - default value is 0
  - arp-flow-hard-timeout
    - The ARP flow will timeout (removed from the switch) after *arp-flow-hard-timeout* seconds, regardless of how many packets it is forwarding
    - This field is only relevant when "is-proactive-flood-mode" is set to "false"
    - default value is 0

Configurable parameters in Address Tracker

- l2switch/addresstracker/implementation/src/main/yang/address-tracker-config.yang
  - timestamp-update-interval
    - A last-seen timestamp is associated with each address. This last-seen timestamp will only be updated after *timestamp-update-interval* milliseconds.
    - A higher value has the advantage of performing less writes to the database.
    - A lower value has the advantage of knowing how fresh an address is.
    - default value is 600000
  - observe-addresses-from
    - IP and MAC addresses can be observed/learned from ARP, IPv4, and IPv6 packets. Set which packets to make these observations from.
    - default value is arp

Configurable parameters in L2 Switch Main

- l2switch/l2switch-main/src/main/yang/l2switch-config.yang
  - is-install-dropall-flow
    - "true" means a drop-all flow will be installed on each switch, so the default action will be to drop a packet instead of sending it to the controller
    - "false" means this flow will not be installed
    - default value is true
  - dropall-flow-table-id
    - The dropall flow will be installed on the specified flow table of each switch

    - This field is only relevant when "is-install-dropall-flow" is set to "true"
    - default value is 0
  - dropall-flow-priority
    - The dropall flow will be installed with the specified priority
    - This field is only relevant when "is-install-dropall-flow" is set to "true"
    - default value is 0
  - dropall-flow-idle-timeout
    - The dropall flow will timeout (removed from the switch) if the flow doesn't forward a packet for *x* seconds
    - This field is only relevant when "is-install-dropall-flow" is set to "true"
    - default value is 0
  - dropall-flow-hard-timeout
    - The dropall flow will timeout (removed from the switch) after *x* seconds, regardless of how many packets it is forwarding
    - This field is only relevant when "is-install-dropall-flow" is set to "true"
    - default value is 0
  - is-learning-only-mode
    - "true" means that the L2 Switch will only be learning addresses. No additional flows to optimize network traffic will be installed.
    - "false" means that the L2 Switch will react to network traffic and install flows on the switches to optimize traffic. Currently, MAC-to-MAC flows are installed.
    - default value is false
  - reactive-flow-table-id
    - The reactive flow will be installed on the specified flow table of each switch
    - This field is only relevant when "is-learning-only-mode" is set to "false"
    - default value is 0
  - reactive-flow-priority
    - The reactive flow will be installed with the specified priority
    - This field is only relevant when "is-learning-only-mode" is set to "false"
    - default value is 10
  - reactive-flow-idle-timeout
    - The reactive flow will timeout (removed from the switch) if the flow doesn't forward a packet for *x* seconds
    - This field is only relevant when "is-learning-only-mode" is set to "false"
    - default value is 600
  - reactive-flow-hard-timeout
    - The reactive flow will timeout (removed from the switch) after *x* seconds, regardless of how many packets it is forwarding

* This field is only relevant when "is-learning-only-mode" is set to "false"
* default value is 300

Change configuration in L2 Switch

Note: For more information on Blueprint in OpenDaylight, see this wiki page.

The following is an example on how to change the configurations of the L2 Switch components.

Use Case: Change the L2 switch from proactive flood mode to reactive mode.

Option 1: (external xml file)

1. Navigate to etc folder under download distribution
2. Create following directory structure:

```
mkdir - p opendaylight/datastore/initial/config
```

3. Create a new xml file corresponding to `<yang module name>_<container name>.xml`:

```
vi arp-handler-config_arp-handler-config.xml
```

4. Add following contents to the created file:

```
<?xml version="1.0" encoding="UTF-8"?>
  <arp-handler-config xmlns="urn:opendaylight:packet:arp-handler-config">
  <is-proactive-flood-mode>false</is-proactive-flood-mode>
</arp-handler-config>
```

5. Restart the controller which injects the configurations.

Option 2: (REST URL)

1. Make the following REST call
   - *URL:* `http://{{LOCALIP}}:8181/restconf/config/arp-handler-config:arp-handler-config/`
   - *Content-Type:* application/json
   - *Body*:

```
{
  "arp-handler-config":
  {
    "is-proactive-flood-mode":false
  }
}
```

   - *Expected Result:* 201 Created
2. Restart the controller to see updated configurations. With out a restart new configurations will be merged with old configurations which is not desirable.

Running the L2 Switch

To run the L2 Switch inside the OpenDaylight distribution simply install the `odl-l2switch-switch-ui` feature;

```
feature:install odl-l2switch-switch-ui
```

Create a network using mininet

```
sudo mn --controller=remote,ip=<Controller IP> --topo=linear,3 --switch ovsk,
↪protocols=OpenFlow13
sudo mn --controller=remote,ip=127.0.0.1 --topo=linear,3 --switch ovsk,
↪protocols=OpenFlow13
```

The above command will create a virtual network consisting of 3 switches. Each switch will connect to the controller located at the specified IP, i.e. 127.0.0.1

```
sudo mn --controller=remote,ip=127.0.0.1 --mac --topo=linear,3 --switch ovsk,
↪protocols=OpenFlow13
```

The above command has the "mac" option, which makes it easier to distinguish between Host MAC addresses and Switch MAC addresses.

Generating network traffic using mininet

```
h1 ping h2
```

The above command will cause host1 (h1) to ping host2 (h2)

```
pingall
```

pingall will cause each host to ping every other host.

Checking Address Observations

Address Observations are added to the Inventory data tree.

The Address Observations on a Node Connector can be checked through a browser or a REST Client.

```
http://10.194.126.91:8080/restconf/operational/opendaylight-inventory:nodes/node/
↪openflow:1/node-connector/openflow:1:1
```

Checking Hosts

Host information is added to the Topology data tree.

- Host address
- Attachment point (link) to a node/switch

This host information and attachment point information can be checked through a browser or a REST Client.

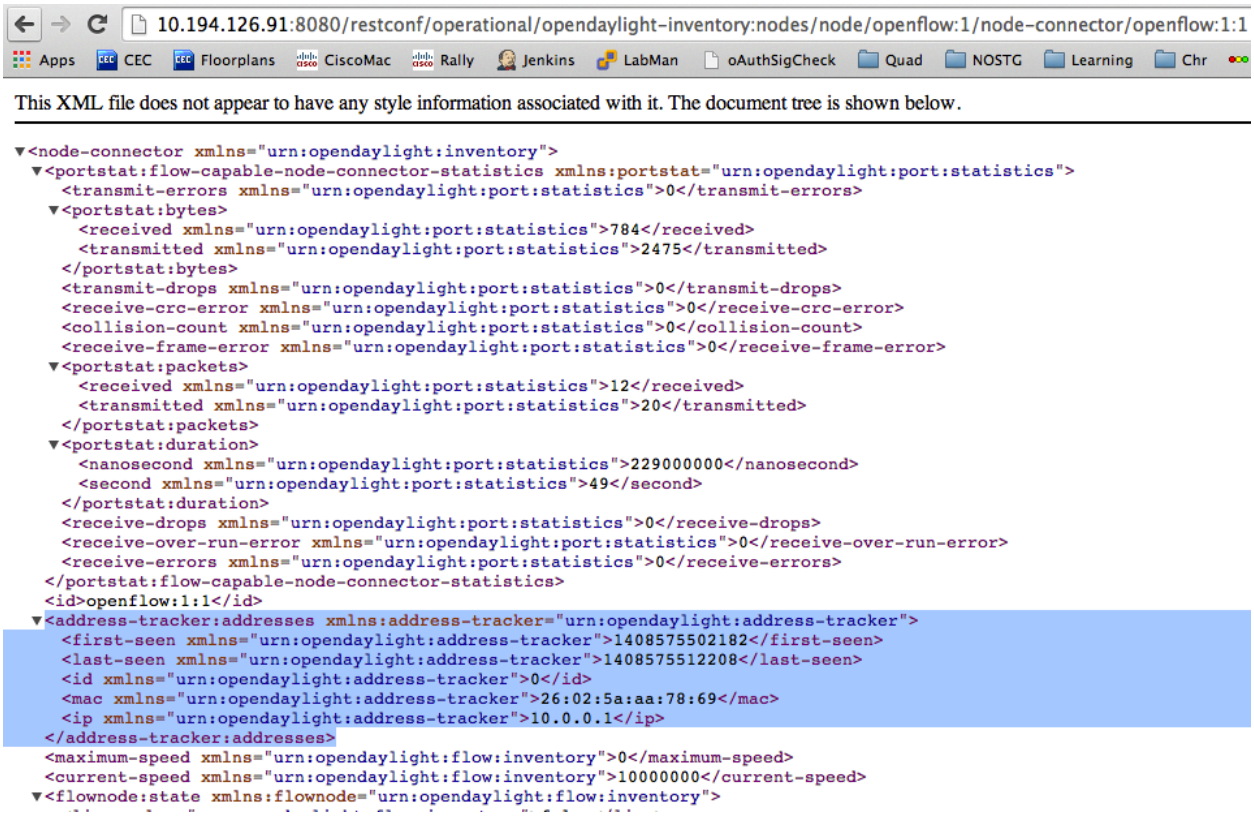


Fig. 1.65: Address Observations

```
http://10.194.126.91:8080/restconf/operational/network-topology:network-topology/
↪topology/flow:1/
```

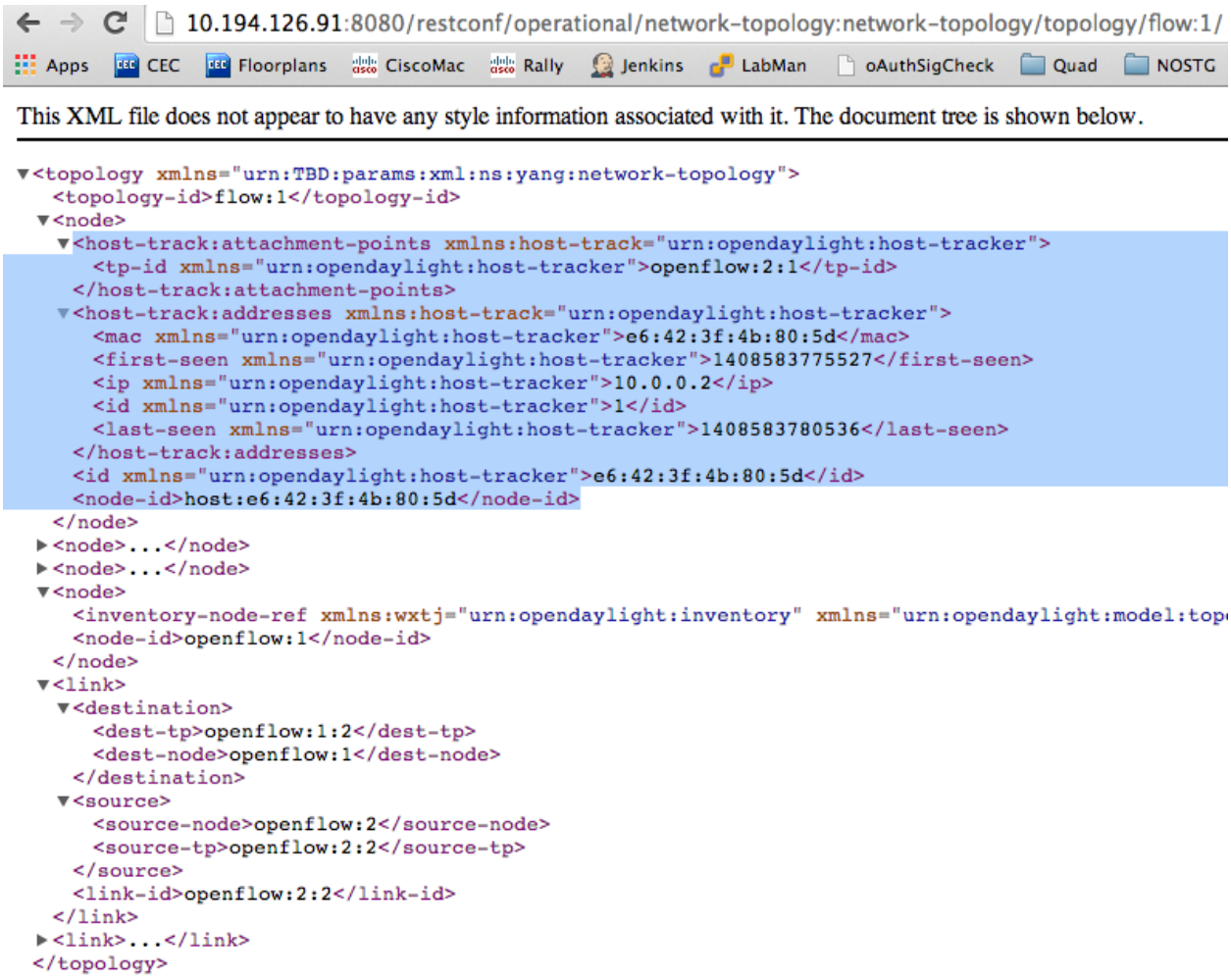


Fig. 1.66: Hosts

Checking STP status of each link

STP Status information is added to the Inventory data tree.

- A status of "forwarding" means the link is active and packets are flowing on it.
- A status of "discarding" means the link is inactive and packets are not sent over it.

The STP status of a link can be checked through a browser or a REST Client.

```
http://10.194.126.91:8080/restconf/operational/opendaylight-inventory:nodes/node/
↪openflow:1/node-connector/openflow:1:2
```

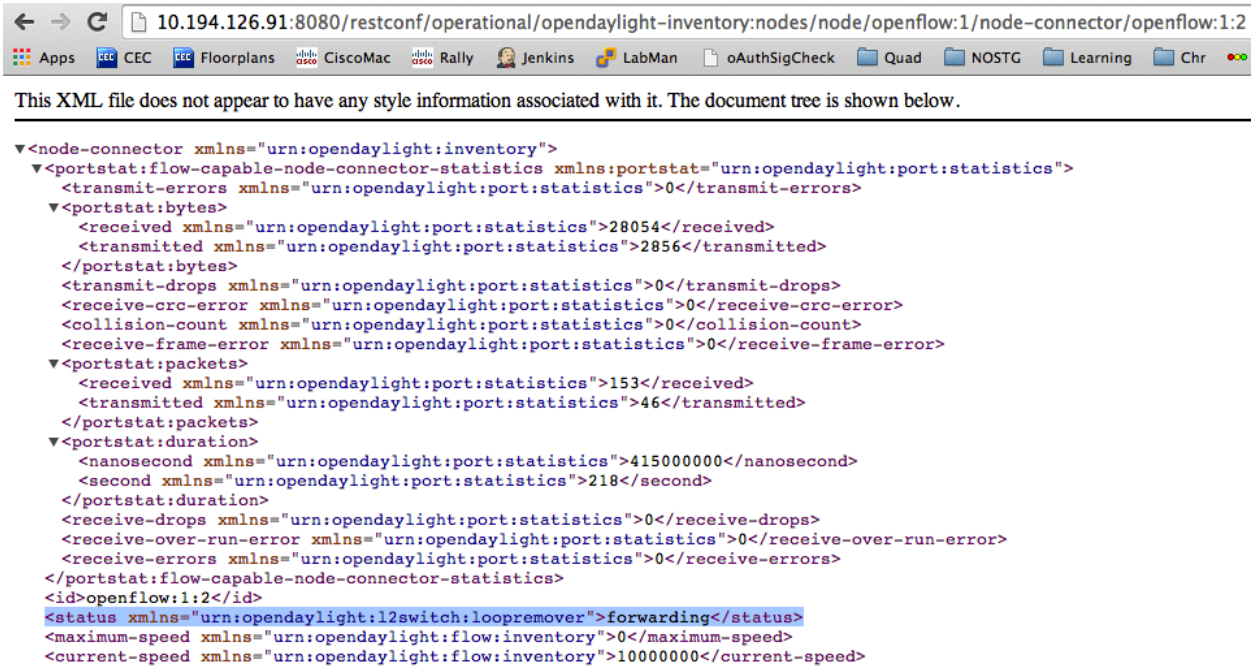


Fig. 1.67: STP status

Miscellaneous mininet commands

```
link s1 s2 down
```

This will bring the link between switch1 (s1) and switch2 (s2) down

```
link s1 s2 up
```

This will bring the link between switch1 (s1) and switch2 (s2) up

```
link s1 h1 down
```

This will bring the link between switch1 (s1) and host1 (h1) down

Link Aggregation Control Protocol User Guide

Overview

This section contains information about how to use the LACP plugin project with OpenDaylight, including configurations.

Link Aggregation Control Protocol Architecture

The LACP Project within OpenDaylight implements Link Aggregation Control Protocol (LACP) as an MD-SAL service module and will be used to auto-discover and aggregate multiple links between an OpenDaylight controlled network and LACP-enabled endpoints or switches. The result is the creation of a logical channel, which represents the aggregation of the links. Link aggregation provides link resiliency and bandwidth aggregation. This implementation adheres to IEEE Ethernet specification 802.3ad.

Configuring Link Aggregation Control Protocol

This feature can be enabled in the Karaf console of the OpenDaylight Karaf distribution by issuing the following command:

```
feature:install odl-lacp-ui
```

Note:

1. Ensure that legacy (non-OpenFlow) switches are configured with LACP mode active with a long timeout to allow for the LACP plugin in OpenDaylight to respond to its messages.
2. Flows that want to take advantage of LACP-configured Link Aggregation Groups (LAGs) must explicitly use a OpenFlow group table entry created by the LACP plugin. The plugin only creates group table entries, it does not program any flows on its own.

Administering or Managing Link Aggregation Control Protocol

LACP-discovered network inventory and network statistics can be viewed using the following REST APIs.

1. List of aggregators available for a node:

   ```
   http://<ControllerIP>:8181/restconf/operational/opendaylight-inventory:nodes/node/
   ↪<node-id>
   ```

   Aggregator information will appear within the `<lacp-aggregators>` XML tag.

2. To view only the information of an aggregator:

   ```
   http://<ControllerIP>:8181/restconf/operational/opendaylight-inventory:nodes/node/
   ↪<node-id>/lacp-aggregators/<agg-id>
   ```

   The group ID associated with the aggregator can be found inside the `<lag-groupid>` XML tag.

   The group table entry information for the `<lag-groupid>` added for the aggregator is also available in the `opendaylight-inventory` node database.

3. To view physical port information.

   ```
   http://<ControllerIP>:8181/restconf/operational/opendaylight-inventory:nodes/node/
   ↪<node-id>/node-connector/<node-connector-id>
   ```

   Ports that are associated with an aggregator will have the tag `<lacp-agg-ref>` updated with valid aggregator information.

Tutorials

The below tutorial demonstrates LACP LAG creation for a sample mininet topology.

Sample LACP Topology creation on Mininet

```
sudo mn --controller=remote,ip=<Controller IP> --topo=linear,1 --switch ovsk,
↪protocols=OpenFlow13
```

The above command will create a virtual network consisting of a switch and a host. The switch will be connected to the controller.

Once the topology is discovered, verify the presence of a flow entry with "dl_type" set to "0x8809" to handle LACP packets using the below ovs-ofctl command:

```
ovs-ofctl -O OpenFlow13 dump-flows s1
 OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x3000000000000001e, duration=60.067s, table=0, n_packets=0, n_bytes=0,␣
↪priority=5,dl_dst=01:80:c2:00:00:02,dl_type=0x8809 actions=CONTROLLER:65535
```

Configure an additional link between the switch (s1) and host (h1) using the below command on mininet shell to aggregate 2 links:

```
mininet> py net.addLink(s1, net.get('h1'))
mininet> py s1.attach('s1-eth2')
```

The LACP module will listen for LACP control packets that are generated from legacy switch (non-OpenFlow enabled). In our example, host (h1) will act as a LACP packet generator. In order to generate the LACP control packets, a bond interface has to be created on the host (h1) with mode type set to LACP with long-timeout. To configure bond interface, create a new file bonding.conf under the /etc/modprobe.d/ directory and insert the below lines in this new file:

```
alias bond0 bonding
options bonding mode=4
```

Here mode=4 is referred to LACP and the default timeout is set to long.

Enable bond interface and associate both physical interface h1-eth0 & h1-eth1 as members of the bond interface on host (h1) using the below commands on the mininet shell:

```
mininet> py net.get('h1').cmd('modprobe bonding')
mininet> py net.get('h1').cmd('ip link add bond0 type bond')
mininet> py net.get('h1').cmd('ip link set bond0 address <bond-mac-address>')
mininet> py net.get('h1').cmd('ip link set h1-eth0 down')
mininet> py net.get('h1').cmd('ip link set h1-eth0 master bond0')
mininet> py net.get('h1').cmd('ip link set h1-eth1 down')
mininet> py net.get('h1').cmd('ip link set h1-eth1 master bond0')
mininet> py net.get('h1').cmd('ip link set bond0 up')
```

Once the bond0 interface is up, the host (h1) will send LACP packets to the switch (s1). The LACP Module will then create a LAG through exchange of LACP packets between the host (h1) and switch (s1). To view the bond interface output on the host (h1) side:

```
mininet> py net.get('h1').cmd('cat /proc/net/bonding/bond0')
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)
Bonding Mode: IEEE 802.3ad Dynamic link aggregation
Transmit Hash Policy: layer2 (0)
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0
802.3ad info
LACP rate: slow
Min links: 0
Aggregator selection policy (ad_select): stable
Active Aggregator Info:
        Aggregator ID: 1
```

```
        Number of ports: 2
        Actor Key: 33
        Partner Key: 27
        Partner Mac Address: 00:00:00:00:01:01
```

```
Slave Interface: h1-eth0
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 00:00:00:00:00:11
Aggregator ID: 1
Slave queue ID: 0
```

```
Slave Interface: h1-eth1
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 00:00:00:00:00:12
Aggregator ID: 1
Slave queue ID: 0
```

A corresponding group table entry would be created on the OpenFlow switch (s1) with "type" set to "select" to perform the LAG functionality. To view the group entries:

```
mininet>ovs-ofctl -O Openflow13 dump-groups s1
OFPST_GROUP_DESC reply (OF1.3) (xid=0x2):
 group_id=60169,type=select,bucket=weight:0,actions=output:1,output:2
```

To apply the LAG functionality on the switches, the flows should be configured with action set to GroupId instead of output port. A sample add-flow configuration with output action set to GroupId:

```
sudo ovs-ofctl -O Openflow13 add-flow s1 dl_type=0x0806,dl_src=SRC_MAC,dl_dst=DST_MAC,
↪actions=group:60169
```

LISP Flow Mapping User Guide

Overview

Locator/ID Separation Protocol

Locator/ID Separation Protocol (LISP) is a technology that provides a flexible map-and-encap framework that can be used for overlay network applications such as data center network virtualization and Network Function Virtualization (NFV).

LISP provides the following name spaces:

- Endpoint Identifiers (EIDs)
- Routing Locators (RLOCs)

In a virtualization environment EIDs can be viewed as virtual address space and RLOCs can be viewed as physical network address space.

The LISP framework decouples network control plane from the forwarding plane by providing:

- A data plane that specifies how the virtualized network addresses are encapsulated in addresses from the underlying physical network.
- A control plane that stores the mapping of the virtual-to-physical address spaces, the associated forwarding policies and serves this information to the data plane on demand.

Network programmability is achieved by programming forwarding policies such as transparent mobility, service chaining, and traffic engineering in the mapping system; where the data plane elements can fetch these policies on demand as new flows arrive. This chapter describes the LISP Flow Mapping project in OpenDaylight and how it can be used to enable advanced SDN and NFV use cases.

LISP data plane Tunnel Routers are available at OpenOverlayRouter.org in the open source community on the following platforms:

- Linux
- Android
- OpenWRT

For more details and support for LISP data plane software please visit the OOR web site.

LISP Flow Mapping Service

The LISP Flow Mapping service provides LISP Mapping System services. This includes LISP Map-Server and LISP Map-Resolver services to store and serve mapping data to data plane nodes as well as to OpenDaylight applications. Mapping data can include mapping of virtual addresses to physical network address where the virtual nodes are reachable or hosted at. Mapping data can also include a variety of routing policies including traffic engineering and load balancing. To leverage this service, OpenDaylight applications and services can use the northbound REST API to define the mappings and policies in the LISP Mapping Service. Data plane devices capable of LISP control protocol can leverage this service through a southbound LISP plugin. LISP-enabled devices must be configured to use this OpenDaylight service as their Map Server and/or Map Resolver.

The southbound LISP plugin supports the LISP control protocol (Map-Register, Map-Request, Map-Reply messages), and can also be used to register mappings in the OpenDaylight mapping service.

LISP Flow Mapping Architecture

The following figure shows the various LISP Flow Mapping modules.

A brief description of each module is as follows:

- **DAO (Data Access Object):** This layer separates the LISP logic from the database, so that we can separate the map server and map resolver from the specific implementation of the mapping database. Currently we have an implementation of this layer with an in-memory HashMap, but it can be switched to any other key/value store and you only need to implement the ILispDAO interface.
- **Map Server:** This module processes the adding or registration of authentication tokens (keys) and mappings. For a detailed specification of LISP Map Server, see LISP.
- **Map Resolver:** This module receives and processes the mapping lookup queries and provides the mappings to requester. For a detailed specification of LISP Map Server, see LISP.
- **RPC/RESTCONF:** This is the auto-generated RESTCONF-based northbound API. This module enables defining key-EID associations as well as adding mapping information through the Map Server. Key-EID associations and mappings can also be queried via this API.
- **GUI:** This module enables adding and querying the mapping service through a GUI based on ODL DLUX.

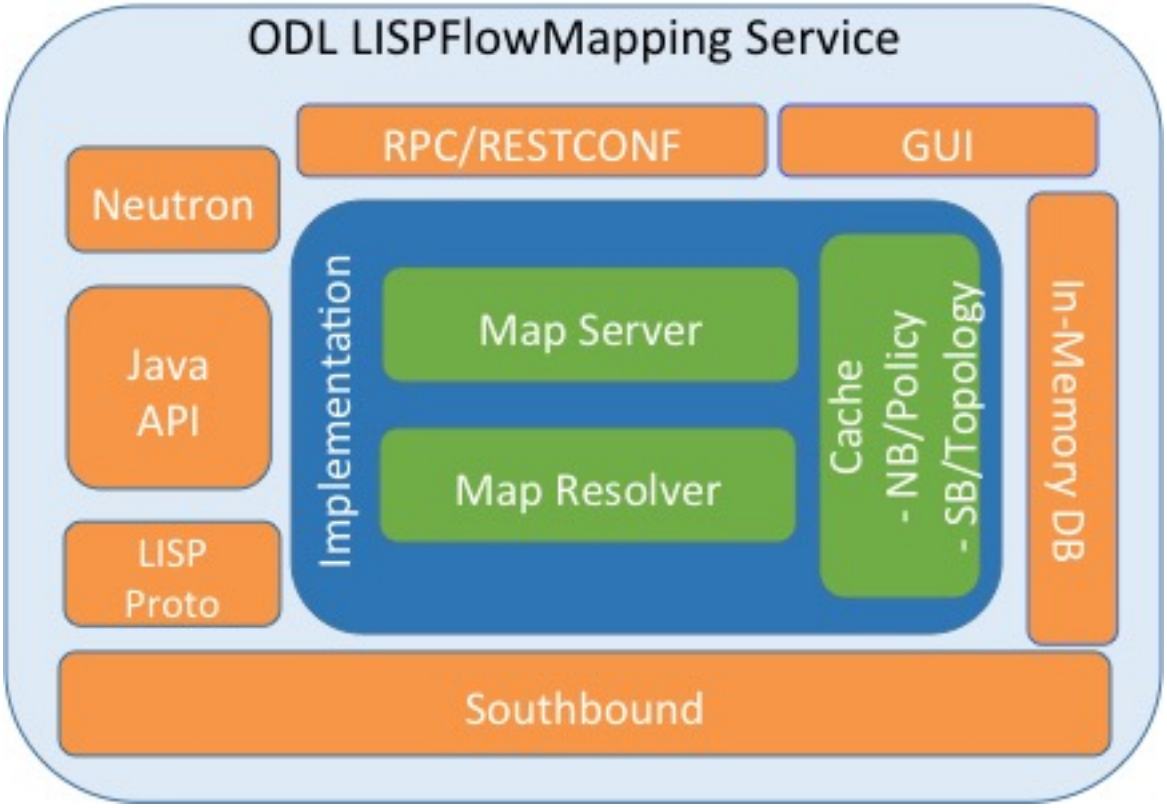


Fig. 1.68: LISP Mapping Service Internal Architecture

- **Neutron:** This module implements the OpenDaylight Neutron Service APIs. It provides integration between the LISP service and the OpenDaylight Neutron service, and thus OpenStack.
- **Java API:** The API module exposes the Map Server and Map Resolver capabilities via a Java API.
- **LISP Proto:** This module includes LISP protocol dependent data types and associated processing.
- **In Memory DB:** This module includes the in memory database implementation of the mapping service.
- **LISP Southbound Plugin:** This plugin enables data plane devices that support LISP control plane protocol (see LISP) to register and query mappings to the LISP Flow Mapping via the LISP control plane protocol.

Configuring LISP Flow Mapping

In order to use the LISP mapping service for registering EID to RLOC mappings from northbound or southbound, keys have to be defined for the EID prefixes first. Once a key is defined for an EID prefix, it can be used to add mappings for that EID prefix multiple times. If the service is going to be used to process Map-Register messages from the southbound LISP plugin, the same key must be used by the data plane device to create the authentication data in the Map-Register messages for the associated EID prefix.

The `etc/custom.properties` file in the Karaf distribution allows configuration of several OpenDaylight parameters. The LISP service has the following properties that can be adjusted:

lisp.smr (default: *false*) Enables/disables the Solicit-Map-Request (SMR) functionality. SMR is a method to notify changes in an EID-to-RLOC mapping to "subscribers". The LISP service considers all Map-Request's source RLOC as a subscriber to the requested EID prefix, and will send an SMR control message to that RLOC if the mapping changes.

lisp.elpPolicy (default: *default*) Configures how to build a Map-Reply southbound message from a mapping containing an Explicit Locator Path (ELP) RLOC. It is used for compatibility with dataplane devices that don't understand the ELP LCAF format. The *default* setting doesn't alter the mapping, returning all RLOCs unmodified. The *both* setting adds a new RLOC to the mapping, with a lower priority than the ELP, that is the next hop in the service chain. To determine the next hop, it searches the source RLOC of the Map-Request in the ELP, and chooses the next hop, if it exists, otherwise it chooses the first hop. The *replace* setting adds a new RLOC using the same algorithm as the *both* setting, but using the origin priority of the ELP RLOC, which is removed from the mapping.

lisp.lookupPolicy (default: *northboundFirst*) Configures the mapping lookup algorithm. When set to *northboundFirst* mappings programmed through the northbound API will take precedence. If no northbound programmed mappings exist, then the mapping service will return mappings registered through the southbound plugin, if any exists. When set to *northboundAndSouthbound* the mapping programmed by the northbound is returned, updated by the up/down status of these mappings as reported by the southbound (if existing).

lisp.mappingMerge (default: *false*) Configures the merge policy on the southbound registrations through the LISP SB Plugin. When set to *false*, only the latest mapping registered through the SB plugin is valid in the southbound mapping database, independent of which device it came from. When set to *true*, mappings for the same EID registered by different devices are merged together and a union of the locators is maintained as the valid mapping for that EID.

Textual Conventions for LISP Address Formats

In addition to the more common IPv4, IPv6 and MAC address data types, the LISP control plane supports arbitrary Address Family Identifiers assigned by IANA, and in addition to those the LISP Canoncal Address Format (LCAF).

The LISP Flow Mapping project in OpenDaylight implements support for many of these different address formats, the full list being summarized in the following table. While some of the address formats have well defined and widely used textual representation, many don't. It became necessary to define a convention to use for text rendering of all

implemented address types in logs, URLs, input fields, etc. The below table lists the supported formats, along with their AFI number and LCAF type, including the prefix used for disambiguation of potential overlap, and examples output.

Name	AFI	LCAF	Prefix	Text Rendering
No Address	0	•	no:	No Address Present
IPv4 Prefix	1	•	ipv4:	192.0.2.0/24
IPv6 Prefix	2	•	ipv6:	2001:db8::/32
MAC Address	16389	•	mac:	00:00:5E:00:53:00
Distinguished Name	17	•	dn:	stringAsIs
AS Number	18	•	as:	AS64500
AFI List	16387	1	list:	{192.0.2.1,192.0.2.2,2001:db8::1 }
Instance ID	16387	2	•	[223] 192.0.2.0/24
Application Data	16387	4	appdata:	192.0.2.1!128!17!80-81!6667-7000
Explicit Locator Path	16387	10	elp:	{192.0.2.1→192.0.2.2|lps→192.0.2.3}
Source/Destination Key	16387	12	srcdst:	192.0.2.1/32|192.0.2.2/32
Key/Value Address Pair	16387	15	kv:	192.0.2.1192.0.2.2
Service Path	16387	N/A	sp:	42(3)

Table: LISP Address Formats

Please note that the forward slash character `/` typically separating IPv4 and IPv6 addresses from the mask length is transformed into `%2f` when used in a URL.

Karaf commands

In this section we will discuss two types of Karaf commands: built-in, and LISP specific. Some built-in commands are quite useful, and are needed for the tutorial, so they will be discussed here. A reference of all LISP specific commands, added by the LISP Flow Mapping project is also included. They are useful mostly for debugging.

Useful built-in commands

`help` Lists all available command, with a short description of each.

`help <command_name>` Show detailed help about a specific command.

`feature:list [-i]` Show all locally available features in the Karaf container. The `-i` option lists only features that are currently installed. It is possible to use `| grep` to filter the output (for all commands, not just this one).

`feature:install <feature_name>` Install feature `feature_name`.

log:set <level> <class> Set the log level for `class` to `level`. The default log level for all classes is INFO. For debugging, or learning about LISP internals it is useful to run `log:set TRACE org.opendaylight.lispflowmapping` right after Karaf starts up.

log:display Outputs the log file to the console, and returns control to the user.

log:tail Continuously shows log output, requires `Ctrl+C` to return to the console.

LISP specific commands

The available lisp commands can always be obtained by `help mappingservice`. Currently they are:

mappingservice:addkey Add the default password `password` for the IPv4 EID prefix 0.0.0.0/0 (all addresses). This is useful when experimenting with southbound devices, and using the REST interface would be combersome for whatever reason.

mappingservice:mappings Show the list of all mappings stored in the internal non-persistent data store (the DAO), listing the full data structure. The output is not human friendly, but can be used for debugging.

LISP Flow Mapping Karaf Features

LISP Flow Mapping has the following Karaf features that can be installed from the Karaf console:

odl-lispflowmapping-msmr This includes the core features required to use the LISP Flow Mapping Service such as mapping service and the LISP southbound plugin.

odl-lispflowmapping-ui This includes the GUI module for the LISP Mapping Service.

odl-lispflowmapping-neutron This is the experimental Neutron provider module for LISP mapping service.

Tutorials

This section provides a tutorial demonstrating various features in this service. We have included tutorials using two forwarding platforms:

1. Using Open Overlay Router (OOR)
2. Using FD.io

Both have different approaches to create the overlay but ultimately do the same job. Details of both approaches have been explained below.

Creating a LISP overlay with OOR

This section provides instructions to set up a LISP network of three nodes (one "client" node and two "server" nodes) using OOR as data plane LISP nodes and the LISP Flow Mapping project from OpenDaylight as the LISP programmable mapping system for the LISP network.

Overview

The steps shown below will demonstrate setting up a LISP network between a client and two servers, then performing a failover between the two "server" nodes.

Prerequisites

- **The OpenDaylight Karaf Distribution** (download)
- **The Postman Chrome App**: the most convenient way to follow along this tutorial is to use the Postman App to edit and send the requests. The project git repository hosts a collection of the requests that are used in this tutorial in the `resources/tutorial/OOR/Beryllium_Tutorial.json.postman_collection` file. You can import this file to Postman by clicking *Import* at the top, choosing *Download from link* and then entering the following URL: https://git.opendaylight.org/gerrit/gitweb?p=lispflowmapping.git;a=blob_plain;f=resources/tutorial/OOR/Beryllium_Tutorial.json.postman_collection;hb=refs/heads/stable/nitrogen. Alternatively, you can save the file on your machine, or if you have the repository checked out, you can import from there. You will need to create a new Postman Environment and define some variables within: `controllerHost` set to the hostname or IP address of the machine running the OpenDaylight instance, and `restconfPort` to 8181, if you didn't modify the default controller settings.
- **OOR version 1.0 or later** The README.md lists the dependencies needed to build it from source.
- **A virtualization platform**

Target Environment

The three LISP data plane nodes and the LISP mapping system are assumed to be running in Linux virtual machines, which have the `eth0` interface in NAT mode to allow outside internet access and `eth1` connected to a host-only network, with the following IP addresses (please adjust configuration files, JSON examples, etc. accordingly if you're using another addressing scheme):

Node	Node Type	IP Address
controller	OpenDaylight	192.168.16.11
client	OOR	192.168.16.30
server1	OOR	192.168.16.31
server2	OOR	192.168.16.32
service-node	OOR	192.168.16.33

Table: Nodes in the tutorial

The figure below gives a sketch of network topology that will be used in the tutorial.

In LISP terminology **client**, **server1** and **server2** are mobile nodes (MN in OOR), **controller** is a MS/MR and **service-node** is a RTR.

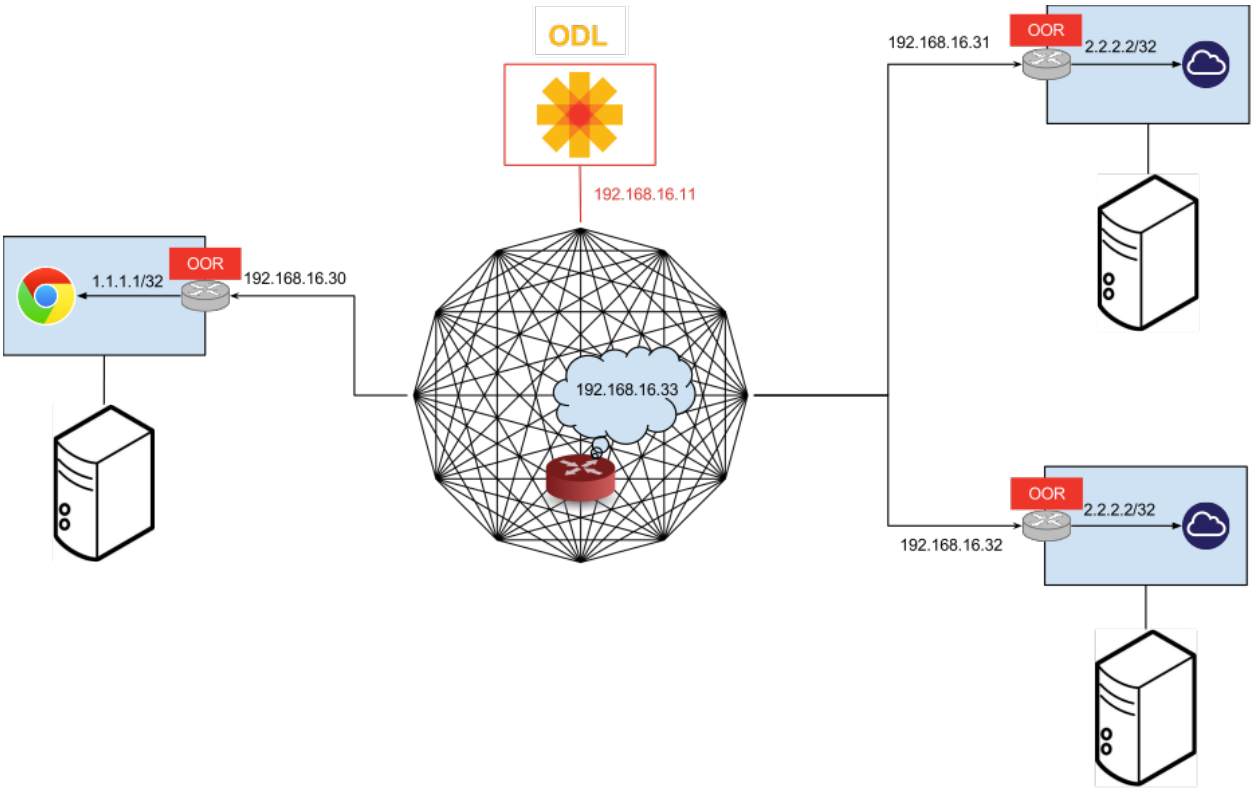
Instructions

The below steps use the command line tool cURL to talk to the LISP Flow Mapping RPC REST API. This is so that you can see the actual request URLs and body content on the page.

1. Install and run the OpenDaylight distribution on the controller VM. Please follow the general OpenDaylight Installation Guide for this step. Once the OpenDaylight controller is running install the *odl-lispflowmapping-msmr* feature from the Karaf CLI:

   ```
   feature:install odl-lispflowmapping-msmr
   ```

   It takes quite a while to load and initialize all features and their dependencies. It's worth running the command `log:tail` in the Karaf console to see when the log output is winding down, and continue with the tutorial after that.



2. Install OOR on the **client**, **server1**, **server2**, and **service-node** VMs following the installation instructions from the OOR README file.

3. Configure the OOR installations from the previous step. Take a look at the `oor.conf.example` to get a general idea of the structure of the conf file. First, check if the file `/etc/oor.conf` exists. If the file doesn't exist, create the file `/etc/oor.conf`. Set the EID in `/etc/oor.conf` file from the IP address space selected for your virtual/LISP network. In this tutorial the EID of the **client** is set to 1.1.1.1/32, and that of **server1** and **server2** to 2.2.2.2/32.

4. Set the RLOC interface to `eth1` in each `oor.conf` file. LISP will determine the RLOC (IP address of the corresponding VM) based on this interface.

5. Set the Map-Resolver address to the IP address of the **controller**, and on the **client** the Map-Server too. On **server1** and **server2** remove the Map-Server configuration, so that it doesn't interfere with the mappings on the controller, since we're going to program them manually.

6. Modify the "key" parameter in each `oor.conf` file to a key/password of your choice (*password* in this tutorial).

Note: The `resources/tutorial/OOR` directory in the project git repository has the files used in the tutorial checked in, so you can just copy the files to `/etc/oor.conf` on the respective VMs. You will also find the JSON files referenced below in the same directory.

---

7. Define a key and EID prefix association in OpenDaylight using the RPC REST API for the **client** EID (1.1.1.1/32) to allow registration from the southbound. Since the mappings for the server EID will be configured from the REST API, no such association is necessary. Run the below command on the **controller** (or any machine that can reach **controller**, by replacing *localhost* with the IP address of **controller**).

```
curl -u "admin":"admin" -H "Content-type: application/json" -X PUT \
    http://localhost:8181/restconf/config/odl-mappingservice:mapping-database/
↪virtual-network-identifier/0/authentication-key/ipv4:1.1.1.1%2f32/ \
```

```
    --data @add-key.json
```

where the content of the *add-key.json* file is the following:

```
{
    "authentication-key": {
        "eid-uri": "ipv4:1.1.1.1/32",
        "eid": {
            "address-type": "ietf-lisp-address-types:ipv4-prefix-afi",
            "ipv4-prefix": "1.1.1.1/32"
        },
        "mapping-authkey": {
            "key-string": "password",
            "key-type": 1
        }
    }
}
```

8. Verify that the key is added properly by requesting the following URL:

```
curl -u "admin":"admin" -H "Content-type: application/json" -X GET \
    http://localhost:8181/restconf/config/odl-mappingservice:mapping-database/
↪virtual-network-identifier/0/authentication-key/ipv4:1.1.1.1%2f32/
```

The output the above invocation should look like this:

```
{
    "authentication-key":[
        {
            "eid-uri":"ipv4:1.1.1.1/32",
            "eid":{
                "ipv4-prefix":"1.1.1.1/32",
                "address-type":"ietf-lisp-address-types:ipv4-prefix-afi"
            },
            "mapping-authkey":{
                "key-string":"password"
                ,"key-type":1
            }
        }
    ]
}
```

9. Run the `oor` OOR daemon on all VMs:

```
oor -f /etc/oor.conf
```

For more information on accessing OOR logs, take a look at OOR README

10. The **client** OOR node should now register its EID-to-RLOC mapping in OpenDaylight. To verify you can lookup the corresponding EIDs via the REST API

```
curl -u "admin":"admin" -H "Content-type: application/json" -X GET \
    http://localhost:8181/restconf/operational/odl-mappingservice:mapping-
↪database/virtual-network-identifier/0/mapping/ipv4:1.1.1.1%2f32/southbound/
```

An alternative way for retrieving mappings from OpenDaylight using the southbound interface is using the lig open source tool.

11. Register the EID-to-RLOC mapping of the server EID 2.2.2.2/32 to the controller, pointing to **server1** and **server2** with a higher priority for **server1**

```
curl -u "admin":"admin" -H "Content-type: application/json" -X PUT \
    http://localhost:8181/restconf/config/odl-mappingservice:mapping-database/
↪virtual-network-identifier/0/mapping/ipv4:2.2.2.2%2f32/northbound/ \
    --data @mapping.json
```

where the *mapping.json* file looks like this:

```
{
    "mapping": {
        "eid-uri": "ipv4:2.2.2.2/32",
        "origin": "northbound",
        "mapping-record": {
            "recordTtl": 1440,
            "action": "NoAction",
            "authoritative": true,
            "eid": {
                "address-type": "ietf-lisp-address-types:ipv4-prefix-afi",
                "ipv4-prefix": "2.2.2.2/32"
            },
            "LocatorRecord": [
                {
                    "locator-id": "server1",
                    "priority": 1,
                    "weight": 1,
                    "multicastPriority": 255,
                    "multicastWeight": 0,
                    "localLocator": true,
                    "rlocProbed": false,
                    "routed": true,
                    "rloc": {
                        "address-type": "ietf-lisp-address-types:ipv4-afi",
                        "ipv4": "192.168.16.31"
                    }
                },
                {
                    "locator-id": "server2",
                    "priority": 2,
                    "weight": 1,
                    "multicastPriority": 255,
                    "multicastWeight": 0,
                    "localLocator": true,
                    "rlocProbed": false,
                    "routed": true,
                    "rloc": {
                        "address-type": "ietf-lisp-address-types:ipv4-afi",
                        "ipv4": "192.168.16.32"
                    }
                }
            ]
        }
    }
}
```

Here the priority of the second RLOC (192.168.16.32 - **server2**) is 2, a higher numeric value than the priority of 192.168.16.31, which is 1. This policy is saying that **server1** is preferred to **server2** for reaching EID 2.2.2.2/32.

Note that lower priority value has higher preference in LISP.

12. Verify the correct registration of the 2.2.2.2/32 EID:

```
curl -u "admin":"admin" -H "Content-type: application/json" -X GET \
    http://localhost:8181/restconf/config/odl-mappingservice:mapping-database/
↪virtual-network-identifier/0/mapping/ipv4:2.2.2.2%2f32/northbound/
```

13. Now the LISP network is up. To verify, log into the **client** VM and ping the server EID:

```
ping 2.2.2.2
```

14. Let's test fail-over now. Suppose you had a service on **server1** which became unavailable, but **server1** itself is still reachable. LISP will not automatically fail over, even if the mapping for 2.2.2.2/32 has two locators, since both locators are still reachable and uses the one with the higher priority (lowest priority value). To force a failover, we need to set the priority of **server2** to a lower value. Using the file mapping.json above, swap the priority values between the two locators (lines 14 and 28 in *mapping.json*) and repeat the request from step 11. You can also repeat step 12 to see if the mapping is correctly registered. If you leave the ping on, and monitor the traffic using wireshark, you can see that the ping traffic to 2.2.2.2 will be diverted from the **server1** RLOC to the **server2** RLOC.

    With the default OpenDaylight configuration the failover should be near instantaneous (we observed 3 lost pings in the worst case), because of the LISP Solicit-Map-Request (SMR) mechanism that can ask a LISP data plane element to update its mapping for a certain EID (enabled by default). It is controlled by the `lisp.smr` variable in `etc/custom.porperties`. When enabled, any mapping change from the RPC interface will trigger an SMR packet to all data plane elements that have requested the mapping in the last 24 hours (this value was chosen because it's the default TTL of Cisco IOS xTR mapping registrations). If disabled, ITRs keep their mappings until the TTL specified in the Map-Reply expires.

15. To add a service chain into the path from the client to the server, we can use an Explicit Locator Path, specifying the **service-node** as the first hop and **server1** (or **server2**) as the second hop. The following will achieve that:

```
curl -u "admin":"admin" -H "Content-type: application/json" -X PUT \
    http://localhost:8181/restconf/config/odl-mappingservice:mapping-database/
↪virtual-network-identifier/0/mapping/ipv4:2.2.2.2%2f32/northbound/ \
    --data @elp.json
```

    where the *elp.json* file is as follows:

```
{
    "mapping": {
        "eid-uri": "ipv4:2.2.2.2/32",
        "origin": "northbound",
        "mapping-record": {
            "recordTtl": 1440,
            "action": "NoAction",
            "authoritative": true,
            "eid": {
                "address-type": "ietf-lisp-address-types:ipv4-prefix-afi",
                "ipv4-prefix": "2.2.2.2/32"
            },
            "LocatorRecord": [
                {
                    "locator-id": "ELP",
                    "priority": 1,
                    "weight": 1,
                    "multicastPriority": 255,
                    "multicastWeight": 0,
```

```
                    "localLocator": true,
                    "rlocProbed": false,
                    "routed": true,
                    "rloc": {
                        "address-type": "ietf-lisp-address-types:explicit-locator-
↪path-lcaf",
                        "explicit-locator-path": {
                            "hop": [
                                {
                                    "hop-id": "service-node",
                                    "address": "192.168.16.33",
                                    "lrs-bits": "strict"
                                },
                                {
                                    "hop-id": "server1",
                                    "address": "192.168.16.31",
                                    "lrs-bits": "strict"
                                }
                            ]
                        }
                    }
                }
            ]
        }
    }
}
```

After the mapping for 2.2.2.2/32 is updated with the above, the ICMP traffic from **client** to **server1** will flow through the **service-node**. You can confirm this in the OOR logs, or by sniffing the traffic on either the **service-node** or **server1**. Note that service chains are unidirectional, so unless another ELP mapping is added for the return traffic, packets will go from **server1** to **client** directly.

16. Suppose the **service-node** is actually a firewall, and traffic is diverted there to support access control lists (ACLs). In this tutorial that can be emulated by using `iptables` firewall rules in the **service-node** VM. To deny traffic on the service chain defined above, the following rule can be added:

```
iptables -A OUTPUT --dst 192.168.16.31 -j DROP
```

The ping from the **client** should now have stopped.

In this case the ACL is done on the destination RLOC. There is an effort underway in the OOR community to allow filtering on EIDs, which is the more logical place to apply ACLs.

17. To delete the rule and restore connectivity on the service chain, delete the ACL by issuing the following command:

```
iptables -D OUTPUT --dst 192.168.16.31 -j DROP
```

which should restore connectivity.

Creating a simple LISP overlay with FD.io

In this section, we use the Overlay Network Engine (ONE) project in FD.io to facilitate fully scripted setup and testing of a LISP/VXLAN-GPE network. Overlay Network Engine (ONE) is a FD.io project that enables programmable dynamic software defined overlays. Details about this project can be found in ONE wiki.

The steps shown below will demonstrate setting up a LISP network between a client and a server using VPP. We demonstrate how to use VPP lite to build a IP4 LISP overlay on an Ubuntu host using namespaces and af_packet interfaces. All configuration files used in the tutorials can be found here.

Prerequisites

- **The OpenDaylight Karaf Distribution** (download)
- **The Postman Chrome App**: Please follow the *instructions* and import postman collection from the following URL: https://git.opendaylight.org/gerrit/gitweb?p=lispflowmapping.git;a=blob;f=resources/tutorial/FD_io/lfm_vpp.postman_collection.json;hb=refs/heads/stable/nitrogen.
- **Vagrant** (optional): Download it from Vagrant website and follow the setup instructions.

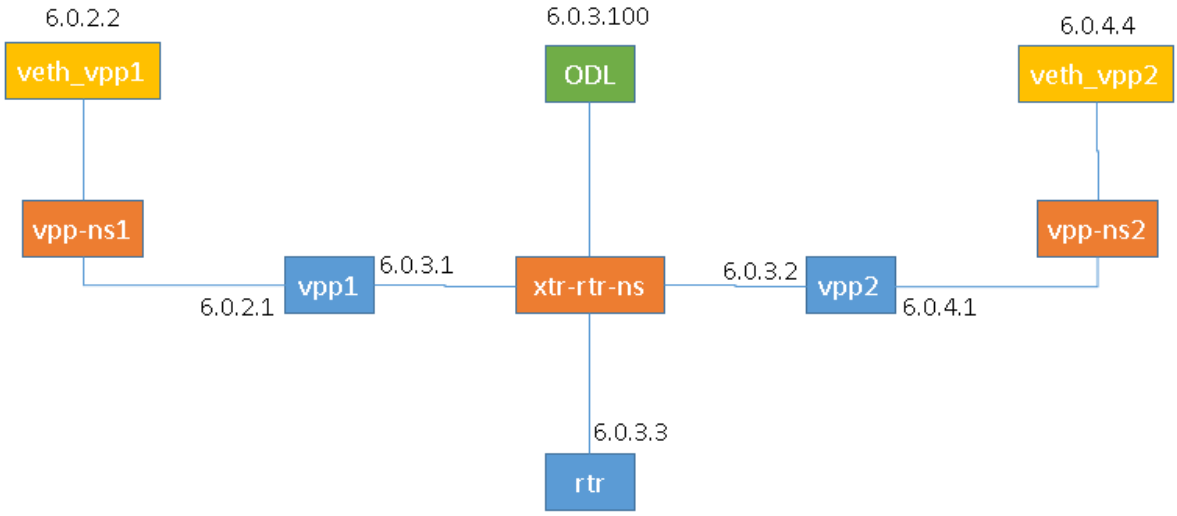
Target Environment

Unlike the case with OOR, we use network namespace functionality of Linux to create the overlay in this case. The following table contains ip addresses of nodes in the overlay topology used in the tutorial. Our objective will be to create this topology and be able to ping from client to server through an intermediary hop, **service node**, which is a `rtr node` providing the service of re-encapsulation. So, all the packets from client to server will be through this **service node**.

Node	Node Type	IP Address
controller	OpenDaylight	6.0.3.100
client	VPP	6.0.2.2
server	VPP	6.0.4.4
service node	VPP	6.0.3.3

Table: Nodes in the tutorial

The figure below gives a sketch of network topology that will be used in the tutorial.



Instructions

Follow the instructions below sequentially.

1. Pull the VPP code anonymously using:

```
git clone https://gerrit.fd.io/r/vpp
```

2. Then, use the vagrant file from repository to build virtual machine with proper environment.

```
cd vpp/build-root/vagrant/
vagrant up
vagrant ssh
```

3. In case there is any error from `vagrant up`, try `vargant ssh`. if it works, no worries. If it still doesn't work, you can try any Ubuntu virtual machine. Or sometimes there is an issue with the Vagrant properly copying the VPP repo code from the host VM after the first installation. In that case `/vpp` doesn't exist. In both cases, follow the instructions from below.

   (a) Clone the code in `/` directory. So, the codes will be in `/vpp`.

   (b) **Run the following commands:**

```
cd /vpp/build-root
make distclean
./bootstrap.sh
make V=0 PLATFORM=vpp TAG=vpp install-deb
sudo dpkg -i /vpp/build-root/*.deb
```

   Alternative and more detailed build instructions can be found in VPP's wiki

4. By now, you should have a Ubuntu VM with VPP repository in `/vpp` with `sudo` access. Now, we need VPP Lite build. The following commands builds VPP Lite.

```
cd /vpp
export PLATFORM=vpp_lite
make build
```

   Successful build create the binary in `/vpp/build-root/install-vpp_lite_debug-native/vpp/bin`

5. Install bridge-utils and ethtool if needed by using following commands:

```
sudo apt-get install bridge-utils ethtool
```

6. Now, install and run OpenDaylight on the VM. Please follow the general OpenDaylight Installation Guide for this step from *Installing OpenDaylight*. Before running OpenDaylight, we need to change the configuration for RTR to work. Update `etc/custom.properties` with the `lisp.elpPolicy` to be replace.

```
lisp.elpPolicy = replace
```

   Then, run OpenDaylight. For details regarding configuring LISP Flow Mapping, please take a look at *Configuring LISP Flow Mapping*. Once the OpenDaylight controller is running install the *odl-lispflowmapping-msmr* feature from the Karaf CLI:

```
feature:install odl-lispflowmapping-msmr
```

It may take quite a while to load and initialize all features and their dependencies. It's worth running the command `log:tail` in the Karaf console to see when the log output is winding down, and continue with the tutorial after that.

7. For setting up VPP, get the files from `resources/tutorial/FD_io` folder of the lispflowmapping repo. The files can also be found here. Copy the `vpp1.config`, `vpp2.config` and `rtr.config` files in `/etc/vpp/lite/`.

8. In this example, VPP doesn't make any southbound map registers to OpenDaylight. So, we add the mappings directly from northbound. For that, we need to add the mappings to OpenDaylight via RESTCONF API.

   Register EID-to-RLOC mapping of the Client EID 6.0.2.0/24.

```
curl -u "admin":"admin" -H "Content-type: application/json" -X PUT \
    http://localhost:8181/restconf/config/odl-mappingservice:mapping-database/
↪virtual-network-identifier/0/mapping/ipv4:6.0.2.0%2f24/northbound/ \
    --data @epl1.json
```

   Content of epl1.json:

```
{
    "mapping": {
        "eid-uri": "ipv4:6.0.2.0/24",
        "origin": "northbound",
        "mapping-record": {
            "recordTtl": 1440,
            "action": "NoAction",
            "authoritative": true,
            "eid": {
                    "address-type": "ietf-lisp-address-types:ipv4-prefix-afi",
                    "ipv4-prefix": "6.0.2.0/24"
            },
            "LocatorRecord": [
                {
                    "locator-id": "ELP",
                    "priority": 1,
                    "weight": 1,
                    "multicastPriority": 255,
                    "multicastWeight": 0,
                    "localLocator": true,
                    "rlocProbed": false,
                    "routed": false,
                    "rloc": {
                        "address-type": "ietf-lisp-address-types:explicit-locator-
↪path-lcaf",
                        "explicit-locator-path": {
                            "hop": [
                                {
                                    "hop-id": "Hop 1",
                                    "address": "6.0.3.3",
                                    "lrs-bits": "lookup rloc-probe strict"
                                },
                                {
                                    "hop-id": "Hop 2",
                                    "address": "6.0.3.1",
                                    "lrs-bits": "lookup strict"
                                }
                            ]
                        }
```

```
                    }
                }
            ]
        }
    }
}
```

Similarly add EID-to-RLOC mapping of the Server EID 6.0.4.0/24.

```
curl -u "admin":"admin" -H "Content-type: application/json" -X PUT \
    http://localhost:8181/restconf/config/odl-mappingservice:mapping-database/
↪virtual-network-identifier/0/mapping/ipv4:6.0.4.0%2f24/northbound/ \
    --data @elp2.json
```

Content of elp2.json:

```
{
    "mapping": {
        "eid-uri": "ipv4:6.0.4.0/24",
        "origin": "northbound",
        "mapping-record": {
            "recordTtl": 1440,
            "action": "NoAction",
            "authoritative": true,
            "eid": {
                    "address-type": "ietf-lisp-address-types:ipv4-prefix-afi",
                    "ipv4-prefix": "6.0.4.0/24"
            },
            "LocatorRecord": [
                {
                    "locator-id": "ELP",
                    "priority": 1,
                    "weight": 1,
                    "multicastPriority": 255,
                    "multicastWeight": 0,
                    "localLocator": true,
                    "rlocProbed": false,
                    "routed": false,
                    "rloc": {
                        "address-type": "ietf-lisp-address-types:explicit-locator-
↪path-lcaf",
                        "explicit-locator-path": {
                            "hop": [
                                {
                                    "hop-id": "Hop 1",
                                    "address": "6.0.3.3",
                                    "lrs-bits": "lookup rloc-probe strict"
                                },
                                {
                                    "hop-id": "Hop 2",
                                    "address": "6.0.3.2",
                                    "lrs-bits": "lookup strict"
                                }
                            ]
                        }
                    }
                }
            ]
```

```
        }
    }
}
```

The JSON files regarding these can be found in here. Even though there is no southbound registration for mapping to OpenDaylight, using northbound policy we can specify mappings, when Client requests for the Server eid, Client gets a reply from OpenDaylight.

9. Assuming all files have been created and OpenDaylight has been configured as explained above, execute the host script you've created or the `topology_setup.sh` script from here.
10. If all goes well, you can now test connectivity between the namespaces with:

```
sudo ip netns exec vpp-ns1 ping 6.0.4.4
```

11. Traffic and control plane message exchanges can be checked with a wireshark listening on the odl interface.
12. **Important:** Delete the topology by running the `topology_setup.sh` with `clean` argument.

```
sudo ./topology_setup.sh clean
```

Creating a LISP overlay with Cisco IOS-XE

This section describes how to create a simple LISP overlay using the Cisco IOS-XE network operating system as the data plane software running on the Cisco CSR 1000v Series Cloud Services Router.

Prerequisites

- **The OpenDaylight Karaf Distribution** (download)
- **CSR1Kv image with Cisco IOS-XE version 03.13.00.S or later** (download; the instructions have been tested on version 03.15.00.S).
- **A virtualization platform** supported by CSR1Kv images (VMware ESXi, Citrix XenServer, KVM, and Microsoft Hyper-V).

Target Environment

The CSR1Kv images are configured with one management interface (`GigabitEthernet1`), and another interface (`GigabitEthernet2`) connected to a host-only network on the virtualization platform, while the LISP mapping system is assumed to be running in a Linux virtual machine, which has the `eth0` interface in NAT mode to allow outside internet access and `eth1` connected to the host-only network, with the following IP addresses (please adjust configuration files, JSON examples, etc. accordingly if you're using another addressing scheme):

Node	Node Type	IP Address
controller	OpenDaylight	192.168.16.11
client	CSR1Kv	192.168.16.30
server	CSR1Kv	192.168.16.31

Table: Nodes in the tutorial

The scenario and EID allocation is the same as the OOR scenario, except that there is no **server2** and **service-node** (for now).

Before this tutorial can be followed, basic connectivity between the Linux VM and the CSRs should work on the host-only network.

Instructions

The below steps use the command line tool cURL to talk to the LISP Flow Mapping RPC REST API. This is so that you can see the actual request URLs and body content on the page. The easy way is to just use Postman.

1. Install and run the OpenDaylight distribution on the controller VM. Please follow the general OpenDaylight Installation Guide from *Installing OpenDaylight* for this step. Once the OpenDaylight controller is running install the *odl-lispflowmapping-msmr* feature from the Karaf CLI:

   ```
   feature:install odl-lispflowmapping-msmr
   ```

   It takes quite a while to load and initialize all features and their dependencies. It's worth running the command `log:tail` in the Karaf console to see when the log output is winding down, and continue with the tutorial after that.

2. Create the **client** and **server** VMs following the installation instructions from the CSR1Kv Configuration Guide.

3. Define a key and EID prefix association in OpenDaylight using the RPC REST API for the **client** and **server** EIDs (1.1.1.1/32 and 2.2.2.2/32 respectively) to allow registration from the southbound. Run the below command on the **controller** (or any machine that can reach **controller**, by replacing *localhost* with the IP address of **controller**).

   ```
   curl -u "admin":"admin" -H "Content-type: application/json" -X PUT \
       http://localhost:8181/restconf/config/odl-mappingservice:mapping-database/
   ↪virtual-network-identifier/0/authentication-key/ipv4:1.1.1.1%2f32/ \
       --data @add-key.json
   ```

   where the content of the *add-key.json* file is the following:

   ```
   {
       "authentication-key": {
           "eid-uri": "ipv4:1.1.1.1/32",
           "eid": {
               "address-type": "ietf-lisp-address-types:ipv4-prefix-afi",
               "ipv4-prefix": "1.1.1.1/32"
           },
           "mapping-authkey": {
               "key-string": "password",
               "key-type": 1
           }
       }
   }
   ```

   The same should be done for 2.2.2.2/32 too.

4. Verify that the key is added properly by requesting the following URL:

   ```
   curl -u "admin":"admin" -H "Content-type: application/json" -X GET \
       http://localhost:8181/restconf/config/odl-mappingservice:mapping-database/
   ↪virtual-network-identifier/0/authentication-key/ipv4:1.1.1.1%2f32/
   ```

   The output the above invocation should look like this:

```
{
    "authentication-key":[
        {
            "eid-uri":"ipv4:1.1.1.1/32",
            "eid":{
                "ipv4-prefix":"1.1.1.1/32",
                "address-type":"ietf-lisp-address-types:ipv4-prefix-afi"
            },
            "mapping-authkey":{
                "key-string":"password"
                ,"key-type":1
            }
        }
    ]
}
```

5. Configure the CSR installations from the previous step. The EID needs to be configured on a loopback interface (except when the CSR is used as a router not a simple client like in this tutorial and the EID is assigned to a real interface).

```
interface Loopback0
 ip address 1.1.1.1 255.255.255.255
```

6. The LISP specific configuration goes to a `router lisp` section in the configuration. A `locator-set` defines the list of locators with their priorities and weights, either statically, or better yet, as an interface name:

```
locator-set rloc-network
 IPv4-interface GigabitEthernet2 priority 1 weight 1
 exit
```

7. To make sure a Map-Request is using the above defined `rloc-network` locator set, the following configuration is used:

```
map-request itr-rlocs rloc-network
```

8. Each Instance ID needs its own configuration. For the default Instance ID of 0, the following configuration is needed for a besic setup:

```
eid-table default instance-id 0
 database-mapping 1.1.1.1/32 locator-set rloc-network
 map-cache 0.0.0.0/0 map-request
 no ipv4 map-cache-persistent
 ipv4 itr map-resolver 192.168.16.11
 ipv4 itr
 ipv4 etr map-server 192.168.16.11 key password
 ipv4 etr
 exit
```

`database-mapping` defines the EID prefix the router will register in the mapping system and which locator set it will use (`rloc-network` in this case, which was defined in step 6).

The next line creates a static map-cache entry for the whole IPv4 EID space, causing a Map-Request to be triggered for every destination (that is not directly connected on some interface).

LISP routers save their map cache to a fie which is used to restore previous state on reboot. To avoid confusion due to state restored from a previous run, `no ipv4 map-cache-persistent` can be used to disable this behavior for non-production testing environments.

A `map-resolver` is then defined, where Map-Requests will be directed to for mapping lookups, and then a `map-server` association with a shared secret key.

9. Here's the full configuration that needs to be pasted into the configuration of the **client** to follow this tutorial:

```
interface Loopback0
 ip address 1.1.1.1 255.255.255.255
!
router lisp
 locator-set rloc-network
  IPv4-interface GigabitEthernet2 priority 1 weight 1
  exit
 !
 map-request itr-rlocs rloc-network
 eid-table default instance-id 0
  database-mapping 1.1.1.1/32 locator-set rloc-network
  map-cache 0.0.0.0/0 map-request
  no ipv4 map-cache-persistent
  ipv4 itr map-resolver 192.168.16.11
  ipv4 itr
  ipv4 etr map-server 192.168.16.11 key password
  ipv4 etr
  exit
 !
 exit
```

Configuring the **server** is done by replacing `1.1.1.1` with `2.2.2.2` in the above configuration snippet.

10. The CSR nodes should now register their EID-to-RLOC mappings to OpenDaylight. To verify, the corresponding EIDs can be looked up via the REST API:

```
curl -u "admin":"admin" -H "Content-type: application/json" -X GET \
    http://localhost:8181/restconf/operational/odl-mappingservice:mapping-
↪database/virtual-network-identifier/0/mapping/ipv4:1.1.1.1%2f32/southbound/
```

An alternative way for retrieving mappings from OpenDaylight using the southbound interface is using the lig open source tool.

Yet another different way is to use the OpenDaylight mappingservice CLI, and type the following at the Karaf prompt:

```
mappingservice:mappings
```

This needs the *odl-lispflowmapping-mappingservice-shell* feature to be loaded. The output is intended for debugging purposes and shows the full Java objects stored in the map-cache.

11. Now the LISP network is up. It can be verified by pinging the **server** EID from the **client** CSR EID:

```
ping 2.2.2.2 source 1.1.1.1
```

LISP Flow Mapping Support

For support the lispflowmapping project can be reached by emailing the developer mailing list: lispflowmapping-dev@lists.opendaylight.org or on the #opendaylight-lispflowmapping IRC channel on irc.freenode.net.

Additional information is also available on the Lisp Flow Mapping wiki

Clustering in LISP Flow Mapping

Documentation regarding setting up a 3-node OpenDaylight cluster is described at following odl wiki page.

To turn on clustering in LISP Flow Mapping it is necessary:

- run script **deploy.py** script. This script is in integration-test project placed at *tools/clustering/cluster-deployer/deploy.py*. A whole deploy.py command can looks like:

{path_to_integration_test_project}/tools/clustering/cluster-deployer/**deploy.py**
–**distribution** {path_to_distribution_in_zip_format}
–**rootdir** {dir_at_remote_host_where_copy_odl_distribution}
–**hosts** {ip1},{ip2},{ip3}
–**clean**
–**template** lispflowmapping
–**rf** 3
–**user** {user_name_of_remote_hosts}
–**password** {password_to_remote_hosts}

Running this script will cause that specified **distribution** to be deployed to remote **hosts** specified through their IP adresses with using credentials (**user** and **password**). The distribution will be copied to specified **rootdir**. As part of the deployment, a **template** which contains a set of controller files which are different from standard ones. In this case it is specified in
{path_to_integration_test_project}/tools/clustering/cluster-deployer/lispflowmapping directory.
Lispflowmapping templates are part of integration-test project. There are 5 template files:

- akka.conf.template
- jolokia.xml.template
- module-shards.conf.template
- modules.conf.template
- org.apache.karaf.features.cfg.template

After copying the distribution, it is unzipped and started on all of specified **hosts** in cluster aware manner.

Remarks

It is necessary to have:

- **unzip** program installed on all of the host
- set all remote hosts /etc/sudoers files to not **requiretty** (should only matter on debian hosts)

NEtwork MOdeling (NEMO)

This section describes how to use the NEMO feature in OpenDaylight and contains contains configuration, administration, and management sections for the feature.

Overview

With the network becoming more complicated, users and applications must handle more complex configurations to deploy new services. NEMO project aims to simplify the usage of network by providing a new intent northbound interface (NBI). Instead of tons of APIs, users/applications just need to describe their intent without caring about complex physical devices and implementation means. The intent will be translated into detailed configurations on the devices in the NEMO engine. A typical scenario is user just need to assign which nodes to implement an VPN, without considering which technique is used.

NEMO Engine Architecture

- NEMO API * The NEMO API provide users the NEMO model, which guides users how to construct the instance of intent, and how to construct the instance of predefined types.
- NEMO REST * The NEMO REST provides users REST APIs to access NEMO engine, that is, user could transmit the intent instance to NEMO engine through basic REST methods.
- NEMO UI * The NEMO UI provides users a visual interface to deploy service with NEMO model, and display the state in DLUX UI.

Installing NEMO engine

To install NEMO engine, download OpenDaylight and use the Karaf console to install the following feature:

odl-nemo-engine-ui

Administering or Managing NEMO Engine

After install features NEMO engine used, user could use NEMO to express his intent with NEMO UI or REST APIs in apidoc.

Go to `http://{controller-ip}:8181/index.html`. In this interface, user could go to NEMO UI, and use the tabs and input box to input intent, and see the state of intent deployment with the image.

Go to `http://{controller-ip}:8181/apidoc/explorer/index.html`. In this interface, user could REST methods “POST”, “PUT”,”GET” and “DELETE” to deploy intent or query the state of deployment.

Tutorials

Below are tutorials for NEMO Engine.

Using NEMO Engine

The purpose of the tutorial is to describe how to use use UI to deploy intent.

Overview

This tutorial will describe how to use the NEMO UI to check the operated resources, the steps to deploy service, and the ultimate state.

Prerequisites

To understand the tutorial well, we hope there are a physical or virtual network exist, and OpenDaylight with NEMO engine must be deployed in one host.

Target Environment

The intent expressed by NEMO model is depended on network resources, so user need to have enough resources to use, or else, the deployment of intent will fail.

Instructions

- Run the OpenDaylight distribution and install odl-nemo-engine-ui from the Karaf console.
- Go to `http://{controller-ip}:8181/index.html`, and sign in.
- Go the NEMO UI interface. And Register a new user with user name, password, and tenant.
- Check the existing resources to see if it is consistent with yours.
- Deploy service with NEMO model by the create intent menu.

NETCONF User Guide

Overview

NETCONF is an XML-based protocol used for configuration and monitoring devices in the network. The base NETCONF protocol is described in RFC-6241.

NETCONF in OpenDaylight:.

OpenDaylight supports the NETCONF protocol as a northbound server as well as a southbound plugin. It also includes a set of test tools for simulating NETCONF devices and clients.

Southbound (netconf-connector)

The NETCONF southbound plugin is capable of connecting to remote NETCONF devices and exposing their configuration/operational datastores, RPCs and notifications as MD-SAL mount points. These mount points allow applications and remote users (over RESTCONF) to interact with the mounted devices.

In terms of RFCs, the connector supports:

- RFC-6241
- RFC-5277
- RFC-6022
- draft-ietf-netconf-yang-library-06

Netconf-connector is fully model-driven (utilizing the YANG modeling language) so in addition to the above RFCs, it supports any data/RPC/notifications described by a YANG model that is implemented by the device.

Tip: NETCONF southbound can be activated by installing `odl-netconf-connector-all` Karaf feature.

Netconf-connector configuration

There are 2 ways for configuring netconf-connector: NETCONF or RESTCONF. This guide focuses on using RESTCONF.

Default configuration

The default configuration contains all the necessary dependencies (file: 01-netconf.xml) and a single instance of netconf-connector (file: 99-netconf-connector.xml) called **controller-config** which connects itself to the NETCONF northbound in OpenDaylight in a loopback fashion. The connector mounts the NETCONF server for config-subsystem in order to enable RESTCONF protocol for config-subsystem. This RESTCONF still goes via NETCONF, but using RESTCONF is much more user friendly than using NETCONF.

Spawning additional netconf-connectors while the controller is running

Preconditions:

1. OpenDaylight is running
2. In Karaf, you must have the netconf-connector installed (at the Karaf prompt, type: `feature:install odl-netconf-connector-all`); the loopback NETCONF mountpoint will be automatically configured and activated
3. Wait until log displays following entry: RemoteDevice{controller-config}: NETCONF connector initialized successfully

To configure a new netconf-connector you need to send following request to RESTCONF:

POST http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules

Headers:

- Accept application/xml
- Content-Type application/xml

```
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
  <type xmlns:prefix=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
↪prefix:sal-netconf-connector</type>
  <name>new-netconf-device</name>
  <address xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">127.0.0.1
↪</address>
  <port xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf
↪">830</port>
  <username xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">admin</
↪username>
  <password xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">admin</
↪password>
  <tcp-only xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">false</
↪tcp-only>
  <event-executor xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
```

```
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:netty">
↪prefix:netty-event-executor</type>
    <name>global-event-executor</name>
  </event-executor>
  <binding-registry xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding
↪">prefix:binding-broker-osgi-registry</type>
    <name>binding-osgi-broker</name>
  </binding-registry>
  <dom-registry xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:dom">
↪prefix:dom-broker-osgi-registry</type>
    <name>dom-broker</name>
  </dom-registry>
  <client-dispatcher xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:config:netconf
↪">prefix:netconf-client-dispatcher</type>
    <name>global-netconf-dispatcher</name>
  </client-dispatcher>
  <processing-executor xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:threadpool">
↪prefix:threadpool</type>
    <name>global-netconf-processing-executor</name>
  </processing-executor>
  <keepalive-executor xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:threadpool">
↪prefix:scheduled-threadpool</type>
    <name>global-netconf-ssh-scheduled-executor</name>
  </keepalive-executor>
</module>
```

This spawns a new netconf-connector which tries to connect to (or mount) a NETCONF device at 127.0.0.1 and port 830. You can check the configuration of config-subsystem's configuration datastore. The new netconf-connector will now be present there. Just invoke:

GET http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules

The response will contain the module for new-netconf-device.

Right after the new netconf-connector is created, it writes some useful metadata into the datastore of MD-SAL under the network-topology subtree. This metadata can be found at:

GET http://localhost:8181/restconf/operational/network-topology:network-topology/

Information about connection status, device capabilities, etc. can be found there.

Connecting to a device not supporting NETCONF monitoring

The netconf-connector in OpenDaylight relies on ietf-netconf-monitoring support when connecting to remote NETCONF device. The ietf-netconf-monitoring support allows netconf-connector to list and download all YANG schemas that are used by the device. NETCONF connector can only communicate with a device if it knows the set of used schemas (or at least a subset). However, some devices use YANG models internally but do not support NETCONF

monitoring. Netconf-connector can also communicate with these devices, but you have to side load the necessary yang models into OpenDaylight's YANG model cache for netconf-connector. In general there are 2 situations you might encounter:

1. NETCONF device does not support ietf-netconf-monitoring but it does list all its YANG models as capabilities in HELLO message

This could be a device that internally uses only ietf-inet-types YANG model with revision 2010-09-24. In the HELLO message that is sent from this device there is this capability reported:

```
urn:ietf:params:xml:ns:yang:ietf-inet-types?module=ietf-inet-types&revision=2010-09-24
```

For such devices you only need to put the schema into folder cache/schema inside your Karaf distribution.

Important: The file with YANG schema for ietf-inet-types has to be called ietf-inet-types@2010-09-24.yang. It is the required naming format of the cache.

---

2. NETCONF device does not support ietf-netconf-monitoring and it does NOT list its YANG models as capabilities in HELLO message

Compared to device that lists its YANG models in HELLO message, in this case there would be no capability with ietf-inet-types in the HELLO message. This type of device basically provides no information about the YANG schemas it uses so its up to the user of OpenDaylight to properly configure netconf-connector for this device.

Netconf-connector has an optional configuration attribute called yang-module-capabilities and this attribute can contain a list of "YANG module based" capabilities. So by setting this configuration attribute, it is possible to override the "yang-module-based" capabilities reported in HELLO message of the device. To do this, we need to modify the configuration of netconf-connector by adding this XML (It needs to be added next to the address, port, username etc. configuration elements):

```
<yang-module-capabilities xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
  <capability xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    urn:ietf:params:xml:ns:yang:ietf-inet-types?module=ietf-inet-types&amp;
↪revision=2010-09-24
  </capability>
</yang-module-capabilities>
```

Remember to also put the YANG schemas into the cache folder.

Note: For putting multiple capabilities, you just need to replicate the capability xml element inside yang-module-capability element. Capability element is modeled as a leaf-list. With this configuration, we would make the remote device report usage of ietf-inet-types in the eyes of netconf-connector.

Reconfiguring Netconf-Connector While the Controller is Running

It is possible to change the configuration of a running module while the whole controller is running. This example will continue where the last left off and will change the configuration for the brand new netconf-connector after it was spawned. Using one RESTCONF request, we will change both username and password for the netconf-connector.

To update an existing netconf-connector you need to send following request to RESTCONF:

PUT http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules/module/odl-sal-netconf-connector-cfg:sal-netconf-connector/new-netconf-device

```
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
  <type xmlns:prefix=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
↪prefix:sal-netconf-connector</type>
  <name>new-netconf-device</name>
  <username xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">bob</
↪username>
  <password xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">passwd</
↪password>
  <tcp-only xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">false</
↪tcp-only>
  <event-executor xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:netty">
↪prefix:netty-event-executor</type>
    <name>global-event-executor</name>
  </event-executor>
  <binding-registry xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding
↪">prefix:binding-broker-osgi-registry</type>
    <name>binding-osgi-broker</name>
  </binding-registry>
  <dom-registry xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:dom">
↪prefix:dom-broker-osgi-registry</type>
    <name>dom-broker</name>
  </dom-registry>
  <client-dispatcher xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:config:netconf
↪">prefix:netconf-client-dispatcher</type>
    <name>global-netconf-dispatcher</name>
  </client-dispatcher>
  <processing-executor xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:threadpool">
↪prefix:threadpool</type>
    <name>global-netconf-processing-executor</name>
  </processing-executor>
  <keepalive-executor xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:threadpool">
↪prefix:scheduled-threadpool</type>
    <name>global-netconf-ssh-scheduled-executor</name>
  </keepalive-executor>
</module>
```

Since a PUT is a replace operation, the whole configuration must be specified along with the new values for username and password. This should result in a 2xx response and the instance of netconf-connector called new-netconf-device

will be reconfigured to use username bob and password passwd. New configuration can be verified by executing:

GET http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules/module/odl-sal-netconf-connector-cfg:sal-netconf-connector/new-netconf-device

With new configuration, the old connection will be closed and a new one established.

Destroying Netconf-Connector While the Controller is Running

Using RESTCONF one can also destroy an instance of a module. In case of netconf-connector, the module will be destroyed, NETCONF connection dropped and all resources will be cleaned. To do this, simply issue a request to following URL:

DELETE http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules/module/odl-sal-netconf-connector-cfg:sal-netconf-connector/new-netconf-device

The last element of the URL is the name of the instance and its predecessor is the type of that module (In our case the type is **sal-netconf-connector** and name **new-netconf-device**). The type and name are actually the keys of the module list.

Netconf-connector configuration with MD-SAL

It is also possible to configure new NETCONF connectors directly through MD-SAL with the usage of the network-topology model. You can configure new NETCONF connectors both through the NETCONF server for MD-SAL (port 2830) or RESTCONF. This guide focuses on RESTCONF.

Tip: To enable NETCONF connector configuration through MD-SAL install either the `odl-netconf-topology` or `odl-netconf-clustered-topology` feature. We will explain the difference between these features later.

Preconditions

1. OpenDaylight is running
2. In Karaf, you must have the `odl-netconf-topology` or `odl-netconf-clustered-topology` feature installed.
3. Feature `odl-restconf` must be installed
4. Wait until log displays following entry:

   ```
   Successfully pushed configuration snapshot 02-netconf-topology.xml(odl-netconf-
   ↪topology,odl-netconf-topology)
   ```

   or until

   ```
   GET http://localhost:8181/restconf/operational/network-topology:network-topology/
   ↪topology/topology-netconf/
   ```

   returns a non-empty response, for example:

```
<topology xmlns="urn:TBD:params:xml:ns:yang:network-topology">
  <topology-id>topology-netconf</topology-id>
</topology>
```

Spawning new NETCONF connectors

To create a new NETCONF connector you need to send the following request to RESTCONF:

```
PUT http://localhost:8181/restconf/config/network-topology:network-topology/topology/
↪topology-netconf/node/new-netconf-device
```

Headers:

- Accept: application/xml
- Content-Type: application/xml

Payload:

```
<node xmlns="urn:TBD:params:xml:ns:yang:network-topology">
  <node-id>new-netconf-device</node-id>
  <host xmlns="urn:opendaylight:netconf-node-topology">127.0.0.1</host>
  <port xmlns="urn:opendaylight:netconf-node-topology">17830</port>
  <username xmlns="urn:opendaylight:netconf-node-topology">admin</username>
  <password xmlns="urn:opendaylight:netconf-node-topology">admin</password>
  <tcp-only xmlns="urn:opendaylight:netconf-node-topology">false</tcp-only>
  <!-- non-mandatory fields with default values, you can safely remove these if you␣
↪do not wish to override any of these values-->
  <reconnect-on-changed-schema xmlns="urn:opendaylight:netconf-node-topology">false</
↪reconnect-on-changed-schema>
  <connection-timeout-millis xmlns="urn:opendaylight:netconf-node-topology">20000</
↪connection-timeout-millis>
  <max-connection-attempts xmlns="urn:opendaylight:netconf-node-topology">0</max-
↪connection-attempts>
  <between-attempts-timeout-millis xmlns="urn:opendaylight:netconf-node-topology">2000
↪</between-attempts-timeout-millis>
  <sleep-factor xmlns="urn:opendaylight:netconf-node-topology">1.5</sleep-factor>
  <!-- keepalive-delay set to 0 turns off keepalives-->
  <keepalive-delay xmlns="urn:opendaylight:netconf-node-topology">120</keepalive-
↪delay>
</node>
```

Note that the device name in <node-id> element must match the last element of the restconf URL.

Reconfiguring an existing connector

The steps to reconfigure an existing connector are exactly the same as when spawning a new connector. The old connection will be disconnected and a new connector with the new configuration will be created.

Deleting an existing connector

To remove an already configured NETCONF connector you need to send the following:

```
DELETE http://localhost:8181/restconf/config/network-topology:network-topology/
↪topology/topology-netconf/node/new-netconf-device
```

Connecting to a device supporting only NETCONF 1.0

OpenDaylight is schema-based distribution and heavily depends on YANG models. However some legacy NETCONF devices are not schema-based and implement just RFC 4741. This type of device does not utilize YANG models internally and OpenDaylight does not know how to communicate with such devices, how to validate data, or what the semantics of data are.

NETCONF connector can communicate also with these devices, but the trade-offs are worsened possibilities in utilization of NETCONF mountpoints. Using RESTCONF with such devices is not suported. Also communicating with schemaless devices from application code is slightly different.

To connect to schemaless device, there is a optional configuration option in netconf-node-topology model called schemaless. You have to set this option to true.

Clustered NETCONF connector

To spawn NETCONF connectors that are cluster-aware you need to install the `odl-netconf-clustered-topology` karaf feature.

> **Warning:** The `odl-netconf-topology` and `odl-netconf-clustered-topology` features are considered **INCOMPATIBLE**. They both manage the same space in the datastore and would issue conflicting writes if installed together.

Configuration of clustered NETCONF connectors works the same as the configuration through the topology model in the previous section.

When a new clustered connector is configured the configuration gets distributed among the member nodes and a NETCONF connector is spawned on each node. From these nodes a master is chosen which handles the schema download from the device and all the communication with the device. You will be able to read/write to/from the device from all slave nodes due to the proxy data brokers implemented.

You can use the `odl-netconf-clustered-topology` feature in a single node scenario as well but the code that uses akka will be used, so for a scenario where only a single node is used, `odl-netconf-topology` might be preferred.

Netconf-connector utilization

Once the connector is up and running, users can utilize the new mount point instance. By using RESTCONF or from their application code. This chapter deals with using RESTCONF and more information for app developers can be found in the developers guide or in the official tutorial application **ncmount** that can be found in the coretutorials project:

- https://github.com/opendaylight/coretutorials/tree/stable/beryllum/ncmount

Reading data from the device

Just invoke (no body needed):

GET http://localhost:8080/restconf/operational/network-topology:network-topology/topology/topology-netconf/node/new-netconf-device/yang-ext:mount/

This will return the entire content of operation datastore from the device. To view just the configuration datastore, change **operational** in this URL to **config**.

Writing configuration data to the device

In general, you cannot simply write any data you want to the device. The data have to conform to the YANG models implemented by the device. In this example we are adding a new interface-configuration to the mounted device (assuming the device supports Cisco-IOS-XR-ifmgr-cfg YANG model). In fact this request comes from the tutorial dedicated to the **ncmount** tutorial app.

POST http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/new-netconf-device/yang-ext:mount/Cisco-IOS-XR-ifmgr-cfg:interface-configurations

```
<interface-configuration xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
    <active>act</active>
    <interface-name>mpls</interface-name>
    <description>Interface description</description>
    <bandwidth>32</bandwidth>
    <link-status></link-status>
</interface-configuration>
```

Should return 200 response code with no body.

Tip: This call is transformed into a couple of NETCONF RPCs. Resulting NETCONF RPCs that go directly to the device can be found in the OpenDaylight logs after invoking `log:set TRACE org.opendaylight.controller.sal.connect.netconf` in the Karaf shell. Seeing the NETCONF RPCs might help with debugging.

This request is very similar to the one where we spawned a new netconf device. That's because we used the loopback netconf-connector to write configuration data into config-subsystem datastore and config-subsystem picked it up from there.

Invoking custom RPC

Devices can implement any additional RPC and as long as it provides YANG models for it, it can be invoked from OpenDaylight. Following example shows how to invoke the get-schema RPC (get-schema is quite common among netconf devices). Invoke:

POST http://localhost:8181/restconf/operations/network-topology:network-topology/topology/topology-netconf/node/new-netconf-device/yang-ext:mount/ietf-netconf-monitoring:get-schema

```
<input xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
  <identifier>ietf-yang-types</identifier>
  <version>2013-07-15</version>
</input>
```

This call should fetch the source for ietf-yang-types YANG model from the mounted device.

Netconf-connector + Netopeer

Netopeer (an open-source NETCONF server) can be used for testing/exploring NETCONF southbound in OpenDaylight.

Netopeer installation

A Docker container with netopeer will be used in this guide. To install Docker and start the netopeer image perform following steps:

1. Install docker http://docs.docker.com/linux/step_one/
2. Start the netopeer image:

```
docker run –rm –t –p 1831:830 dockeruser/netopeer
```

3. Verify netopeer is running by invoking (netopeer should send its HELLO message right away:

```
ssh root@localhost –p 1831 –s netconf
(password root)
```

Mounting netopeer NETCONF server

Preconditions:

- OpenDaylight is started with features `odl-restconf-all` and `odl-netconf-connector-all`.
- Netopeer is up and running in docker

Now just follow the chapter: *Spawning netconf-connector*. In the payload change the:

- name, e.g., to netopeer
- username/password to your system credentials
- ip to localhost
- port to 1831.

After netopeer is mounted successfully, its configuration can be read using RESTCONF by invoking:

GET http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/netopeer/yang-ext:mount/

Northbound (NETCONF servers)

OpenDaylight provides 2 types of NETCONF servers:

- **NETCONF server for config-subsystem (listening by default on port 1830)**
  - Serves as a default interface for config-subsystem and allows users to spawn/reconfigure/destroy modules (or applications) in OpenDaylight
- **NETCONF server for MD-SAL (listening by default on port 2830)**
  - Serves as an alternative interface for MD-SAL (besides RESTCONF) and allows users to read/write data from MD-SAL's datastore and to invoke its rpcs (NETCONF notifications are not available in the Boron release of OpenDaylight)

Note: The reason for having 2 NETCONF servers is that config-subsystem and MD-SAL are 2 different components of OpenDaylight and require different approach for NETCONF message handling and data translation. These 2 components will probably merge in the future.

Note: Since Nitrogen release, there is performance regression in NETCONF servers accepting SSH connections. While opening a connection takes less than 10 seconds on Carbon, on Nitrogen time can increase up to 60 seconds. Please see https://bugs.opendaylight.org/show_bug.cgi?id=9020

NETCONF server for config-subsystem

This NETCONF server is the primary interface for config-subsystem. It allows the users to interact with config-subsystem in a standardized NETCONF manner.

In terms of RFCs, these are supported:

- RFC-6241
- RFC-5277
- RFC-6470
  - (partially, only the schema-change notification is available in Boron release)
- RFC-6022

For regular users it is recommended to use RESTCONF + the controller-config loopback mountpoint instead of using pure NETCONF. How to do that is spesific for each component/module/application in OpenDaylight and can be found in their dedicated user guides.

NETCONF server for MD-SAL

This NETCONF server is just a generic interface to MD-SAL in OpenDaylight. It uses the stadard MD-SAL APIs and serves as an alternative to RESTCONF. It is fully model driven and supports any data and rpcs that are supported by MD-SAL.

In terms of RFCs, these are supported:

- RFC-6241
- RFC-6022
- draft-ietf-netconf-yang-library-06

Notifications over NETCONF are not supported in the Boron release.

Tip: Install NETCONF northbound for MD-SAL by installing feature: `odl-netconf-mdsal` in karaf. Default binding port is **2830**.

Configuration

The default configuration can be found in file: *08-netconf-mdsal.xml*. The file contains the configuration for all necessary dependencies and a single SSH endpoint starting on port 2830. There is also a (by default disabled) TCP

endpoint. It is possible to start multiple endpoints at the same time either in the initial configuration file or while OpenDaylight is running.

The credentials for SSH endpoint can also be configured here, the defaults are admin/admin. Credentials in the SSH endpoint are not yet managed by the centralized AAA component and have to be configured separately.

Verifying MD-SAL's NETCONF server

After the NETCONF server is available it can be examined by a command line ssh tool:

```
ssh admin@localhost -p 2830 -s netconf
```

The server will respond by sending its HELLO message and can be used as a regular NETCONF server from then on.

Mounting the MD-SAL's NETCONF server

To perform this operation, just spawn a new netconf-connector as described in *Spawning netconf-connector*. Just change the ip to "127.0.0.1" port to "2830" and its name to "controller-mdsal".

Now the MD-SAL's datastore can be read over RESTCONF via NETCONF by invoking:

GET http://localhost:8181/restconf/operational/network-topology:network-topology/topology/topology-netconf/node/controller-mdsal/yang-ext:mount

---

Note: This might not seem very useful, since MD-SAL can be accessed directly from RESTCONF or from Application code, but the same method can be used to mount and control other OpenDaylight instances by the "master OpenDaylight".

NETCONF testtool

NETCONF testtool is a set of standalone runnable jars that can:

- Simulate NETCONF devices (suitable for scale testing)
- Stress/Performance test NETCONF devices
- Stress/Performance test RESTCONF devices

These jars are part of OpenDaylight's controller project and are built from the NETCONF codebase in OpenDaylight.

Tip: Download testtool from OpenDaylight Nexus at: https://nexus.opendaylight.org/content/repositories/public/org/opendaylight/netconf/netconf-testtool/1.1.0-Boron/

Nexus contains 3 executable tools:

- executable.jar - device simulator
- stress.client.tar.gz - NETCONF stress/performance measuring tool
- perf-client.jar - RESTCONF stress/performance measuring tool

---

Tip: Each executable tool provides help. Just invoke `java -jar <name-of-the-tool.jar> --help`

NETCONF device simulator

NETCONF testtool (or NETCONF device simulator) is a tool that

- Simulates 1 or more NETCONF devices
- Is suitable for scale, performance or crud testing
- Uses core implementation of NETCONF server from OpenDaylight
- Generates configuration files for controller so that the OpenDaylight distribution (Karaf) can easily connect to all simulated devices
- Provides broad configuration options
- Can start a fully fledged MD-SAL datastore
- Supports notifications

Building testtool

1. Check out latest NETCONF repository from git
2. Move into the `opendaylight/netconf/tools/netconf-testtool/` folder
3. Build testtool using the `mvn clean install` command

Downloading testtool

Netconf-testtool is now part of default maven build profile for controller and can be also downloaded from nexus. The executable jar for testtool can be found at: nexus-artifacts

Running testtool

1. After successfully building or downloading, move into the `opendaylight/netconf/tools/netconf-testtool/target/` folder and there is file `netconf-testtool-1.1.0-SNAPSHOT-executable.jar` (or if downloaded from nexus just take that jar file)
2. Execute this file using, e.g.:

```
java -jar netconf-testtool-1.1.0-SNAPSHOT-executable.jar
```

This execution runs the testtool with default for all parameters and you should see this log output from the testtool :

```
10:31:08.206 [main] INFO  o.o.c.n.t.t.NetconfDeviceSimulator - Starting 1, SSH␣
↪simulated devices starting on port 17830
10:31:08.675 [main] INFO  o.o.c.n.t.t.NetconfDeviceSimulator - All simulated␣
↪devices started successfully from port 17830 to 17830
```

Default Parameters

The default parameters for testtool are:

- Use SSH

- Run 1 simulated device
- Device port is 17830
- YANG modules used by device are only: ietf-netconf-monitoring, ietf-yang-types, ietf-inet-types (these modules are required for device in order to support NETCONF monitoring and are included in the netconf-testtool)
- Connection timeout is set to 30 minutes (quite high, but when testing with 10000 devices it might take some time for all of them to fully establish a connection)
- Debug level is set to false
- No distribution is modified to connect automatically to the NETCONF testtool

Verifying testtool

To verify that the simulated device is up and running, we can try to connect to it using command line ssh tool. Execute this command to connect to the device:

```
ssh admin@localhost -p 17830 -s netconf
```

Just accept the server with yes (if required) and provide any password (testtool accepts all users with all passwords). You should see the hello message sent by simulated device.

Testtool help

```
usage: netconf testool [-h] [--device-count DEVICES-COUNT] [--devices-per-port␣
↪DEVICES-PER-PORT] [--schemas-dir SCHEMAS-DIR] [--notification-file NOTIFICATION-
↪FILE]
                       [--initial-config-xml-file INITIAL-CONFIG-XML-FILE] [--
↪starting-port STARTING-PORT] [--generate-config-connection-timeout GENERATE-CONFIG-
↪CONNECTION-TIMEOUT]
                       [--generate-config-address GENERATE-CONFIG-ADDRESS] [--
↪generate-configs-batch-size GENERATE-CONFIGS-BATCH-SIZE] [--distribution-folder␣
↪DISTRO-FOLDER] [--ssh SSH] [--exi EXI]
                       [--debug DEBUG] [--md-sal MD-SAL]

NETCONF device simulator. Detailed info can be found at https://wiki.opendaylight.org/
↪view/OpenDaylight_Controller:Netconf:Testtool#Building_testtool

optional arguments:
  -h, --help             show this help message and exit
  --device-count DEVICES-COUNT
                         Number of simulated netconf devices to spin. This is the␣
↪number of actual ports open for the devices.
  --devices-per-port DEVICES-PER-PORT
                         Amount of config files generated per port to spoof more␣
↪devices then are actually running
  --schemas-dir SCHEMAS-DIR
                         Directory containing yang schemas to describe simulated␣
↪devices. Some schemas e.g. netconf monitoring and inet types are included by default
  --notification-file NOTIFICATION-FILE
                         Xml file containing notifications that should be sent to␣
↪clients after create subscription is called
  --initial-config-xml-file INITIAL-CONFIG-XML-FILE
                         Xml file containing initial simulatted configuration to be␣
↪returned via get-config rpc
```

```
  --starting-port STARTING-PORT
                        First port for simulated device. Each other device will have␣
↪previous+1 port number
  --generate-config-connection-timeout GENERATE-CONFIG-CONNECTION-TIMEOUT
                        Timeout to be generated in initial config files
  --generate-config-address GENERATE-CONFIG-ADDRESS
                        Address to be placed in generated configs
  --generate-configs-batch-size GENERATE-CONFIGS-BATCH-SIZE
                        Number of connector configs per generated file
  --distribution-folder DISTRO-FOLDER
                        Directory where the karaf distribution for controller is␣
↪located
  --ssh SSH             Whether to use ssh for transport or just pure tcp
  --exi EXI             Whether to use exi to transport xml content
  --debug DEBUG         Whether to use debug log level instead of INFO
  --md-sal MD-SAL       Whether to use md-sal datastore instead of default simulated␣
↪datastore.
```

Supported operations

Testtool default simple datastore supported operations:

get-schema returns YANG schemas loaded from user specified directory,

edit-config always returns OK and stores the XML from the input in a local variable available for get-config and get RPC. Every edit-config replaces the previous data,

commit always returns OK, but does not actually commit the data,

get-config returns local XML stored by edit-config,

get returns local XML stored by edit-config with netconf-state subtree, but also supports filtering.

(un)lock returns always OK with no lock guarantee

create-subscription returns always OK and after the operation is triggered, provided NETCONF notifications (if any) are fed to the client. No filtering or stream recognition is supported.

Note: when operation="delete" is present in the payload for edit-config, it will wipe its local store to simulate the removal of data.

When using the MD-SAL datastore testtool behaves more like normal NETCONF server and is suitable for crud testing. create-subscription is not supported when testtool is running with the MD-SAL datastore.

Notification support

Testtool supports notifications via the –notification-file switch. To trigger the notification feed, create-subscription operation has to be invoked. The XML file provided should look like this example file:

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<notifications>

<!-- Notifications are processed in the order they are defined in XML -->

<!-- Notification that is sent only once right after create-subscription is called -->
<notification>
    <!-- Content of each notification entry must contain the entire notification with␣
↪event time. Event time can be hardcoded, or generated by testtool if XXXX is set as␣
↪eventtime in this XML -->
```

```
    <content><![CDATA[
        <notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
            <eventTime>2011-01-04T12:30:46</eventTime>
            <random-notification xmlns="http://www.opendaylight.org/netconf/event:1.0
↪">
                <random-content>single no delay</random-content>
            </random-notification>
        </notification>
    ]]></content>
</notification>

<!-- Repeated Notification that is sent 5 times with 2 second delay inbetween -->
<notification>
    <!-- Delay in seconds from previous notification -->
    <delay>2</delay>
    <!-- Number of times this notification should be repeated -->
    <times>5</times>
    <content><![CDATA[
        <notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
            <eventTime>XXXX</eventTime>
            <random-notification xmlns="http://www.opendaylight.org/netconf/event:1.0
↪">
                <random-content>scheduled 5 times 10 seconds each</random-content>
            </random-notification>
        </notification>
    ]]></content>
</notification>

<!-- Single notification that is sent only once right after the previous notification␣
↪-->
<notification>
    <delay>2</delay>
    <content><![CDATA[
        <notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
            <eventTime>XXXX</eventTime>
            <random-notification xmlns="http://www.opendaylight.org/netconf/event:1.0
↪">
                <random-content>single with delay</random-content>
            </random-notification>
        </notification>
    ]]></content>
</notification>

</notifications>
```

Connecting testtool with controller Karaf distribution

Auto connect to OpenDaylight

It is possible to make OpenDaylight auto connect to the simulated devices spawned by testtool (so user does not have to post a configuration for every NETCONF connector via RESTCONF). The testtool is able to modify the OpenDaylight distribution to auto connect to the simulated devices after feature `odl-netconf-connector-all` is installed. When running testtool, issue this command (just point the testool to the distribution:

```
java -jar netconf-testtool-1.1.0-SNAPSHOT-executable.jar --device-count 10 --
↪distribution-folder ~/distribution-karaf-0.4.0-SNAPSHOT/ --debug true
```

With the distribution-folder parameter, the testtool will modify the distribution to include configuration for netconf-connector to connect to all simulated devices. So there is no need to spawn netconf-connectors via RESTCONF.

Running testtool and OpenDaylight on different machines

The testtool binds by default to 0.0.0.0 so it should be accessible from remote machines. However you need to set the parameter "generate-config-address" (when using autoconnect) to the address of machine where testtool will be run so OpenDaylight can connect. The default value is localhost.

Executing operations via RESTCONF on a mounted simulated device

Simulated devices support basic RPCs for editing their config. This part shows how to edit data for simulated device via RESTCONF.

Test YANG schema

The controller and RESTCONF assume that the data that can be manipulated for mounted device is described by a YANG schema. For demonstration, we will define a simple YANG model:

```
module test {
    yang-version 1;
    namespace "urn:opendaylight:test";
    prefix "tt";

    revision "2014-10-17";


   container cont {

        leaf l {
            type string;
        }
   }
}
```

Save this schema in file called test@2014-10-17.yang and store it a directory called test-schemas/, e.g., your home folder.

Editing data for simulated device

- Start the device with following command:

```
java -jar netconf-testtool-1.1.0-SNAPSHOT-executable.jar --device-count 10 --
↪distribution-folder ~/distribution-karaf-0.4.0-SNAPSHOT/ --debug true --schemas-
↪dir ~/test-schemas/
```

- Start OpenDaylight
- Install odl-netconf-connector-all feature

- Install odl-restconf feature
- Check that you can see config data for simulated device by executing GET request to

```
http://localhost:8181/restconf/config/network-topology:network-topology/topology/
↪topology-netconf/node/17830-sim-device/yang-ext:mount/
```

- The data should be just and empty data container
- Now execute edit-config request by executing a POST request to:

```
http://localhost:8181/restconf/config/network-topology:network-topology/topology/
↪topology-netconf/node/17830-sim-device/yang-ext:mount
```

with headers:

```
Accept application/xml
Content-Type application/xml
```

and payload:

```
<cont xmlns="urn:opendaylight:test">
  <l>Content</l>
</cont>
```

- Check that you can see modified config data for simulated device by executing GET request to

```
http://localhost:8181/restconf/config/network-topology:network-topology/topology/
↪topology-netconf/node/17830-sim-device/yang-ext:mount/
```

- Check that you can see the same modified data in operational for simulated device by executing GET request to

```
http://localhost:8181/restconf/operational/network-topology:network-topology/
↪topology/topology-netconf/node/17830-sim-device/yang-ext:mount/
```

Warning: Data will be mirrored in operational datastore only when using the default simple datastore.

Known problems

Slow creation of devices on virtual machines

When testtool seems to take unusually long time to create the devices use this flag when running it:

```
-Dorg.apache.sshd.registerBouncyCastle=false
```

Too many files open

When testtool or OpenDaylight starts to fail with TooManyFilesOpen exception, you need to increase the limit of open files in your OS. To find out the limit in linux execute:

```
ulimit -a
```

Example sufficient configuration in linux:

```
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 63338
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 500000
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 63338
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

To set these limits edit file: /etc/security/limits.conf, for example:

```
*         hard    nofile      500000
*         soft    nofile      500000
root      hard    nofile      500000
root      soft    nofile      500000
```

"Killed"

The testtool might end unexpectedly with a simple message: "Killed". This means that the OS killed the tool due to too much memory consumed or too many threads spawned. To find out the reason on linux you can use following command:

```
dmesg | egrep -i -B100 'killed process'
```

Also take a look at this file: /proc/sys/kernel/threads-max. It limits the number of threads spawned by a process. Sufficient (but probably much more than enough) value is, e.g., 126676

NETCONF stress/performance measuring tool

This is basically a NETCONF client that puts NETCONF servers under heavy load of NETCONF RPCs and measures the time until a configurable amount of them is processed.

RESTCONF stress-performance measuring tool

Very similar to NETCONF stress tool with the difference of using RESTCONF protocol instead of NETCONF.

YANGLIB remote repository

There are scenarios in NETCONF deployment, that require for a centralized YANG models repository. YANGLIB plugin provides such remote repository.

To start this plugin, you have to install odl-yanglib feature. Then you have to configure YANGLIB either through RESTCONF or NETCONF. We will show how to configure YANGLIB through RESTCONF.

YANGLIB configuration through RESTCONF

You have to specify what local YANG modules directory you want to provide. Then you have to specify address and port whre you want to provide YANG sources. For example, we want to serve yang sources from folder /sources on localhost:5000 adress. The configuration for this scenario will be as follows:

```
PUT  http://localhost:8181/restconf/config/network-topology:network-topology/topology/
↪topology-netconf/node/controller-config/yang-ext:mount/config:modules/module/
↪yanglib:yanglib/example
```

Headers:

- Accept: application/xml
- Content-Type: application/xml

Payload:

```
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
  <name>example</name>
  <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:yanglib:impl">
↪prefix:yanglib</type>
  <broker xmlns="urn:opendaylight:params:xml:ns:yang:controller:yanglib:impl">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding
↪">prefix:binding-broker-osgi-registry</type>
    <name>binding-osgi-broker</name>
  </broker>
  <cache-folder xmlns="urn:opendaylight:params:xml:ns:yang:controller:yanglib:impl">/
↪sources</cache-folder>
  <binding-addr xmlns="urn:opendaylight:params:xml:ns:yang:controller:yanglib:impl">
↪localhost</binding-addr>
  <binding-port xmlns="urn:opendaylight:params:xml:ns:yang:controller:yanglib:impl">
↪5000</binding-port>
</module>
```

This should result in a 2xx response and new YANGLIB instance should be created. This YANGLIB takes all YANG sources from /sources folder and for each generates URL in form:

```
http://localhost:5000/schemas/{modelName}/{revision}
```

On this URL will be hosted YANG source for particular module.

YANGLIB instance also write this URL along with source identifier to ietf-netconf-yang-library/modules-state/module list.

Netconf-connector with YANG library as fallback

There is an optional configuration in netconf-connector called yang-library. You can specify YANG library to be plugged as additional source provider into the mount's schema repository. Since YANGLIB plugin is advertising provided modules through yang-library model, we can use it in mount point's configuration as YANG library. To do this, we need to modify the configuration of netconf-connector by adding this XML

```
<yang-library xmlns="urn:opendaylight:netconf-node-topology">
  <yang-library-url xmlns="urn:opendaylight:netconf-node-topology">http://
↪localhost:8181/restconf/operational/ietf-yang-library:modules-state</yang-library-
↪url>
  <username xmlns="urn:opendaylight:netconf-node-topology">admin</username>
```

```
  <password xmlns="urn:opendaylight:netconf-node-topology">admin</password>
</yang-library>
```

This will register YANGLIB provided sources as a fallback schemas for particular mount point.

NETCONF Call Home

Important: The call home feature is experimental and will change in a future release. In particular, the Yang models will change to those specified in the RFC 8071

Call Home Installation

ODL Call-Home server is installed in Karaf by installing karaf feature `odl-netconf-callhome-ssh`. RESTCONF feature is recommended for configuring Call Home & testing its functionality.

```
feature:install odl-netconf-callhome-ssh
```

Note: In order to test Call Home functionality we recommend Netopeer. See Netopeer Call Home to learn how to enable call-home on Netopeer.

Northbound Call-Home API

The northbound Call Home API is used for administering the Call-Home Server. The following describes this configuration.

Global Configuration

Configuring global credentials

ODL Call-Home server allows user to configure global credentials, which will be used for devices which does not have device-specific credentials configured.

This is done by creating `/odl-netconf-callhome-server:netconf-callhome-server/global/credentials` with username and passwords specified.

Configuring global username & passwords to try

```
PUT
/restconf/config/odl-netconf-callhome-server:netconf-callhome-server/global/
↪credentials HTTP/1.1
Content-Type: application/json
Accept: application/json
```

```
{
  "credentials":
  {
    "username": "example",
```

```
    "passwords": [ "first-password-to-try", "second-password-to-try" ]
  }
}
```

Configuring to accept any ssh server key using global credentials

By default Netconf Call-Home Server accepts only incoming connections from allowed devices `/odl-netconf-callhome-server:netconf-callhome-server/allowed-devices`, if user desire to allow all incoming connections, it is possible to set `accept-all-ssh-keys` to `true` in `/odl-netconf-callhome-server:netconf-callhome-server/global`.

The name of this devices in `netconf-topology` will be in format `ip-address:port`. For naming devices see Device-Specific Configuration.

Allowing unknown devices to connect

This is a debug feature and should not be used in production. Besides being an obvious security issue, this also causes the Call-Home Server to drastically increase its output to the log.

```
POST
/restconf/config/odl-netconf-callhome-server:netconf-callhome-server/global HTTP/1.1
Content-Type: application/json
Accept: application/json
```

```
{
  "global": {
    "accept-all-ssh-keys": "true"
  }
}
```

Device-Specific Configuration

Allowing Device & Configuring Name

Netconf Call Home Server uses device provided SSH server key (host key) to identify device. The pairing of name and server key is configured in `/odl-netconf-callhome-server:netconf-callhome-server/allowed-devices`. This list is colloquially called a whitelist.

If the Call-Home Server finds the SSH host key in the whitelist, it continues to negotiate a NETCONF connection over an SSH session. If the SSH host key is not found, the connection between the Call Home server and the device is dropped immediately. In either case, the device that connects to the Call home server leaves a record of its presence in the operational store.

Example of configuring device

```
PUT
/restconf/config/odl-netconf-callhome-server:netconf-callhome-server/allowed-devices/
↪device/example HTTP/1.1
Content-Type: application/json
Accept: application/json
```

```
{
  "device": {
    "unique-id": "example",
```

```
    "ssh-host-key":
↪"AAAAB3NzaC1yc2EAAAADAQABAAABAQDHoH1jMjltOJnCt999uaSfc48ySutaD3ISJ9fSECe1Spdq9o9mxj0kBTTTq+2V8hPspu
↪rgJeoUewWwCAasRx9X4eTcRrJrwOQKzb5Fk+UKgQmenZ5uhLAefi2qXX/
↪agFCtZi99vw+jHXZStfHm9TZCAf2zi+HIBzoVksSNJD0VvPo66EAvLn5qKWQD4AdpQQbKqXRf5/
↪W8diPySbYdvOP2/7HFhDukW8yV/
↪7ZtcywFUIu3gdXsrzwMnTqnATSLPPuckoi0V2jd8dQvEcu1DY+rRqmqu0tEkFBurlRZDf1yhNzq5xWY3OXcjgDGN+RxwuWQK3cF
↪"
  }
}
```

Configuring Device with Device-specific Credentials

Call Home Server also allows to configure credentials per device basis, this is done by introducing `credentials` container into device-specific configuration. Format is same as in global credentials.

Configuring Device with Credentials

```
PUT
/restconf/config/odl-netconf-callhome-server:netconf-callhome-server/allowed-devices/
↪device/example HTTP/1.1
Content-Type: application/json
Accept: application/json
```

```
{
  "device": {
    "unique-id": "example",
    "credentials": {
      "username": "example",
      "passwords": [ "password" ]
    },
    "ssh-host-key":
↪"AAAAB3NzaC1yc2EAAAADAQABAAABAQDHoH1jMjltOJnCt999uaSfc48ySutaD3ISJ9fSECe1Spdq9o9mxj0kBTTTq+2V8hPspu
↪rgJeoUewWwCAasRx9X4eTcRrJrwOQKzb5Fk+UKgQmenZ5uhLAefi2qXX/
↪agFCtZi99vw+jHXZStfHm9TZCAf2zi+HIBzoVksSNJD0VvPo66EAvLn5qKWQD4AdpQQbKqXRf5/
↪W8diPySbYdvOP2/7HFhDukW8yV/
↪7ZtcywFUIu3gdXsrzwMnTqnATSLPPuckoi0V2jd8dQvEcu1DY+rRqmqu0tEkFBurlRZDf1yhNzq5xWY3OXcjgDGN+RxwuWQK3cF
↪"
  }
}
```

Operational Status

Once an entry is made into the config side of "allowed-devices", the Call-Home Server will populate an corresponding operational device that is the same as the config device but has an additional status. By default, this status is *DISCONNECTED*. Once a device calls home, this status will change to one of:

CONNECTED — The device is currently connected and the NETCONF mount is available for network management.

FAILED_AUTH_FAILURE — The last attempted connection was unsuccessful because the Call-Home Server was unable to provide the acceptable credentials of the device. The device is also disconnected and not available for network management.

FAILED_NOT_ALLOWED — The last attempted connection was unsuccessful because the device was not recognized as an acceptable device. The device is also disconnected and not available for network management.

FAILED — The last attempted connection was unsuccessful for a reason other than not allowed to connect or incorrect client credentials. The device is also disconnected and not available for network management.

DISCONNECTED — The device is currently disconnected.

Rogue Devices

Devices which are not on the whitelist might try to connect to the Call-Home Server. In these cases, the server will keep a record by instantiating an operational device. There will be no corresponding config device for these rogues. They can be identified readily because their device id, rather than being user-supplied, will be of the form "address:port". Note that if a device calls back multiple times, there will only be a single operatinal entry (even if the port changes); these devices are recognized by their unique host key.

Southbound Call-Home API

The Call-Home Server listens for incoming TCP connections and assumes that the other side of the connection is a device calling home via a NETCONF connection with SSH for management. The server uses port 6666 by default and this can be configured via a blueprint configuration file.

The device **must** initiate the connection and the server will not try to re-establish the connection in case of a drop. By requirement, the server cannot assume it has connectivity to the device due to NAT or firewalls among others.

NetIDE User Guide

Overview

OpenDaylight's NetIDE project allows users to run SDN applications written for different SDN controllers, e.g., Floodlight or Ryu, on top of OpenDaylight managed infrastructure. The NetIDE Network Engine integrates a client controller layer that executes the modules that compose a Network Application and interfaces with a server SDN controller layer that drives the underlying infrastructure. In addition, it provides a uniform interface to common tools that are intended to allow the inspection/debug of the control channel and the management of the network resources.

The Network Engine provides a compatibility layer capable of translating calls of the network applications running on top of the client controllers, into calls for the server controller framework. The communication between the client and the server layers is achieved through the NetIDE intermediate protocol, which is an application-layer protocol on top of TCP that transmits the network control/management messages from the client to the server controller and vice-versa. Between client and server controller sits the Core Layer which also speaks the intermediate protocol.

NetIDE API

Architecture and Design

The NetIDE engine follows the ONF's proposed Client/Server SDN Application architecture.

Core

The NetIDE Core is a message-based system that allows for the exchange of messages between OpenDaylight and subscribed Client SDN Controllers

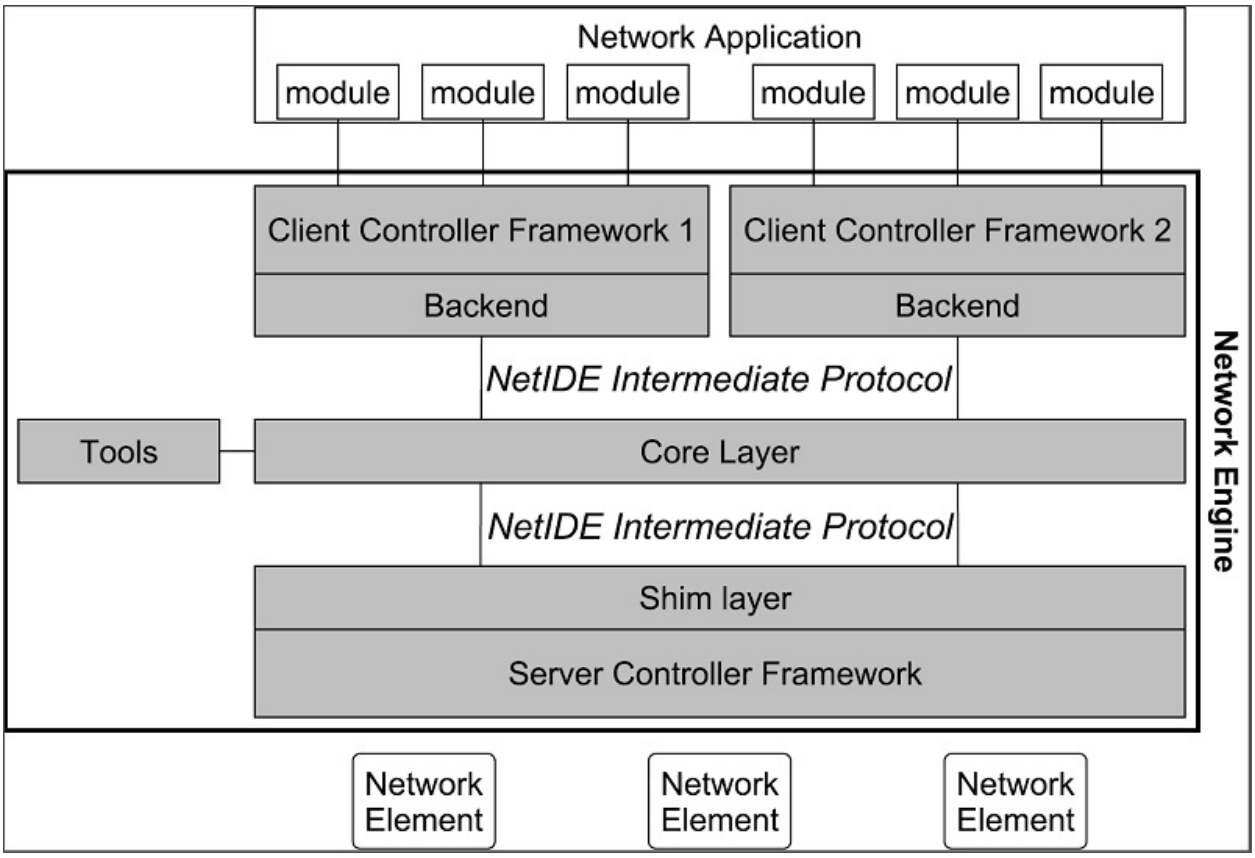


Fig. 1.69: NetIDE Network Engine Architecture

Handling reply messages correctly

When an application module sends a request to the network (e.g. flow statistics, features, etc.), the Network Engine must be able to correctly drive the corresponding reply to such a module. This is not a trivial task, as many modules may compose the network application running on top of the Network Engine, and there is no way for the Core to pair replies and requests. The transaction IDs (xid) in the OpenFlow header are unusable in this case, as it may happen that different modules use the same values.

In the proposed approach, represented in the figure below, the task of pairing replies with requests is performed by the Shim Layer which replaces the original xid of the OpenFlow requests coming from the core with new unique xid values. The Shim also saves the original OpenFlow xid value and the module id it finds in the NetIDE header. As the network elements must use the same xid values in the replies, the Shim layer can easily pair a reply with the correct request as it is using unique xid values.

The below figure shows how the Network Engine should handle the controller-to-switch OpenFlow messages. The diagram shows the case of a request message sent by an application module to a network element where the Backend inserts the module id of the module in the NetIDE header (X in the Figure). For other messages generated by the client controller platform (e.g. echo requests) or by the Backend, the module id of the Backend is used (Y in the Figure).

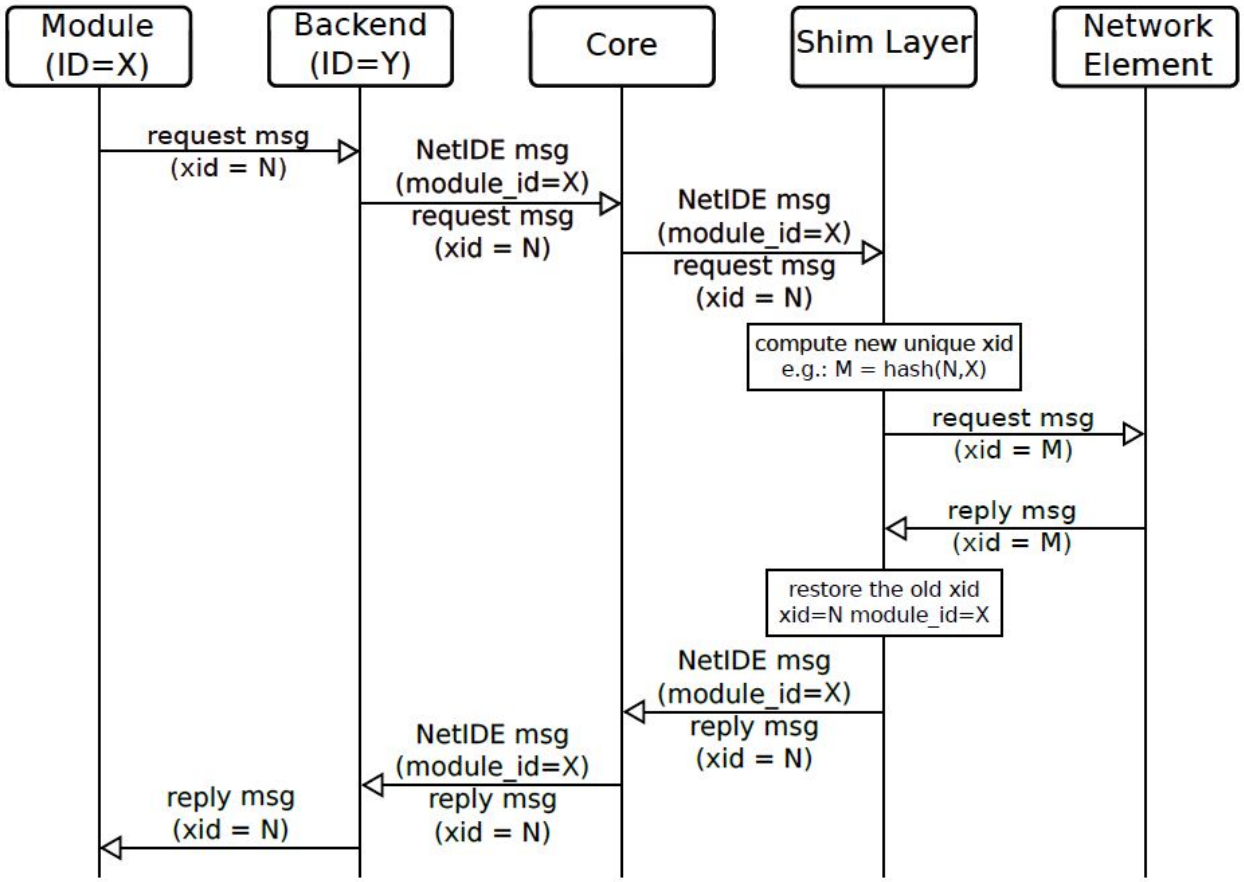


Fig. 1.70: NetIDE Communication Flow

Configuration

Below are the configuration items which can be edited, including their default values.

- core-address: This is the ip address of the NetIDE Core, default is 127.0.0.1
- core-port: The port of on which the NetIDE core is listening on
- address: IP address where the controller listens for switch connections, default is 127.0.0.1
- port: Port where controller listens for switch connections, default: 6644
- transport-protocol: default is TCP
- switch-idle-timeout: default is 15000ms

NetVirt User Guide

L3VPN Service: User Guide

Overview

L3VPN Service in OpenDaylight provides a framework to create L3VPN based on BGP-MP. It also helps to create Network Virtualization for DC Cloud environment.

Modules & Interfaces

L3VPN service can be realized using the following modules -

VPN Service Modules

1. **VPN Manager** : Creates and manages VPNs and VPN Interfaces
2. **BGP Manager** : Configures BGP routing stack and provides interface to routing services
3. **FIB Manager** : Provides interface to FIB, creates and manages forwarding rules in Dataplane
4. **Nexthop Manager** : Creates and manages nexthop egress pointer, creates egress rules in Dataplane
5. **Interface Manager** : Creates and manages different type of network interfaces, e.g., VLAN, l3tunnel etc.,
6. **Id Manager** : Provides cluster-wide unique ID for a given key. Used by different modules to get unique IDs for different entities.
7. **MD-SAL Util** : Provides interface to MD-SAL. Used by service modules to access MD-SAL Datastore and services.

All the above modules can function independently and can be utilized by other services as well.

Configuration Interfaces

The following modules expose configuration interfaces through which user can configure L3VPN Service.

1. BGP Manager
2. VPN Manager
3. Interface Manager
4. FIB Manager

Configuration Interface Details

1. Data Node Path : */config/bgp:bgp-router/*
   (a) Fields :
      i. local-as-identifier
      ii. local-as-number
   (b) REST Methods : GET, PUT, DELETE, POST
2. Data Node Path : */config/bgp:bgp-neighbors/*
   (a) Fields :
      i. List of bgp-neighbor
   (b) REST Methods : GET, PUT, DELETE, POST
3. Data Node Path : */config/bgp:bgp-neighbors/bgp-neighbor/‘‘{as-number}‘‘/*
   (a) Fields :
      i. as-number
      ii. ip-address
   (b) REST Methods : GET, PUT, DELETE, POST

1. Data Node Path : */config/l3vpn:vpn-instances/*
   (a) Fields :
      i. List of vpn-instance
   (b) REST Methods : GET, PUT, DELETE, POST
2. Data Node Path : */config/l3vpn:vpn-interfaces/vpn-instance*
   (a) Fields :
      i. name
      ii. route-distinguisher
      iii. import-route-policy
      iv. export-route-policy
   (b) REST Methods : GET, PUT, DELETE, POST
3. Data Node Path : */config/l3vpn:vpn-interfaces/*
   (a) Fields :
      i. List of vpn-interface
   (b) REST Methods : GET, PUT, DELETE, POST
4. Data Node Path : */config/l3vpn:vpn-interfaces/vpn-interface*
   (a) Fields :
      i. name
      ii. vpn-instance-name
   (b) REST Methods : GET, PUT, DELETE, POST
5. Data Node Path : */config/l3vpn:vpn-interfaces/vpn-interface/‘‘{name}‘‘/adjacency*

(a) Fields :

i. ip-address

ii. mac-address

(b) REST Methods : GET, PUT, DELETE, POST

1. Data Node Path : */config/if:interfaces/interface*

(a) Fields :

i. name

ii. type

iii. enabled

iv. of-port-id

v. tenant-id

vi. base-interface

(b) type specific fields

i. when type = *l2vlan*

A. vlan-id

ii. when type = *stacked_vlan*

A. stacked-vlan-id

iii. when type = *l3tunnel*

A. tunnel-type

B. local-ip

C. remote-ip

D. gateway-ip

iv. when type = *mpls*

A. list labelStack

B. num-labels

(c) REST Methods : GET, PUT, DELETE, POST

1. Data Node Path : */config/odl-fib:fibEntries/vrfTables*

(a) Fields :

i. List of vrfTables

(b) REST Methods : GET, PUT, DELETE, POST

2. Data Node Path : */config/odl-fib:fibEntries/vrfTables/‘‘{routeDistinguisher}‘‘/*

(a) Fields :

i. route-distinguisher

ii. list vrfEntries

A. destPrefix

B. label

C. nexthopAddress

(b) REST Methods : GET, PUT, DELETE, POST

3. Data Node Path : */config/odl-fib:fibEntries/ipv4Table*

(a) Fields :

i. list ipv4Entry

A. destPrefix

B. nexthopAddress

(b) REST Methods : GET, PUT, DELETE, POST

Provisioning Sequence & Sample Configurations

Installation

1. Edit *etc/custom.properties* and set the following property: *vpnservice.bgpspeaker.host.name = <bgpserver-ip>* *<bgpserver-ip>* here refers to the IP address of the host where BGP is running.
2. Run ODL and install VPN Service *feature:install odl-vpnservice-core*

Use REST interface to configure L3VPN service

Pre-requisites:

1. BGP stack with VRF support needs to installed and configured
   (a) *Configure BGP as specified in Step 1 below.*
2. Create pairs of GRE/VxLAN Tunnels (using ovsdb/ovs-vsctl) between each switch and between each switch to the Gateway node
   (a) *Create *l3tunnel* interfaces corresponding to each tunnel in interfaces DS as specified in Step 2 below.**

Step 1 : Configure BGP

1. Configure BGP Router

REST API : *PUT /config/bgp:bgp-router/*

Sample JSON Data

```
{
    "bgp-router": {
        "local-as-identifier": "10.10.10.10",
        "local-as-number": 108
    }
}
```

2. Configure BGP Neighbors

REST API : *PUT /config/bgp:bgp-neighbors/*

Sample JSON Data

```
{
   "bgp-neighbor" : [
          {
              "as-number": 105,
              "ip-address": "169.144.42.168"
          }
     ]
 }
```

Step 2 : Create Tunnel Interfaces

Create l3tunnel interfaces corresponding to all GRE/VxLAN tunnels created with ovsdb (*refer Prerequisites*). Use following REST Interface -

REST API : *PUT /config/if:interfaces/if:interfacce*

Sample JSON Data

```
{
    "interface": [
        {
            "name" : "GRE_192.168.57.101_192.168.57.102",
            "type" : "odl-interface:l3tunnel",
            "odl-interface:tunnel-type": "odl-interface:tunnel-type-gre",
            "odl-interface:local-ip" : "192.168.57.101",
            "odl-interface:remote-ip" : "192.168.57.102",
            "odl-interface:portId" : "openflow:1:3",
            "enabled" : "true"
        }
    ]
}
```

Following is expected as a result of these configurations

1. Unique If-index is generated
2. *Interface-state* operational DS is updated
3. Corresponding Nexthop Group Entry is created

Step 3 : OS Create Neutron Ports and attach VMs

At this step user creates VMs.

Step 4 : Create VM Interfaces

Create l2vlan interfaces corresponding to VM created in step 3

REST API : *PUT /config/if:interfaces/if:interface*

Sample JSON Data

```
{
    "interface": [
        {
            "name" : "dpn1-dp1.2",
            "type" : "l2vlan",
            "odl-interface:of-port-id" : "openflow:1:2",
            "odl-interface:vlan-id" : "1",
            "enabled" : "true"
        }
    ]
}
```

Step 5: Create VPN Instance

REST API : *PUT /config/l3vpn:vpn-instances/l3vpn:vpn-instance/*

Sample JSON Data

```
{
  "vpn-instance": [
    {
        "description": "Test VPN Instance 1",
        "vpn-instance-name": "testVpn1",
        "ipv4-family": {
            "route-distinguisher": "4000:1",
            "export-route-policy": "4000:1,5000:1",
            "import-route-policy": "4000:1,5000:1",
        }
    }
  ]
}
```

Following is expected as a result of these configurations

1. VPN ID is allocated and updated in data-store
2. Corresponding VRF is created in BGP
3. If there are vpn-interface configurations for this VPN, corresponding action is taken as defined in step 5

Step 5 : Create VPN-Interface and Local Adjacency

this can be done in two steps as well

1. Create vpn-interface

REST API : *PUT /config/l3vpn:vpn-interfaces/l3vpn:vpn-interface/*

Sample JSON Data

```
{
  "vpn-interface": [
    {
      "vpn-instance-name": "testVpn1",
      "name": "dpn1-dp1.2",
    }
  ]
}
```

Note: name here is the name of VM interface created in step 3, 4

2. Add Adjacencies on vpn-interafce

REST API : *PUT /config/l3vpn:vpn-interfaces/l3vpn:vpn-interface/dpn1-dp1.3/adjacency*

Sample JSON Data

```
  {
     "adjacency" : [
            {
                "ip-address" : "169.144.42.168",
                "mac-address" : "11:22:33:44:55:66"
            }
        ]
   }

its a list, user can define more than one adjacency on a
vpn\_interface
```

Above steps can be carried out in a single step as following

```
{
    "vpn-interface": [
        {
            "vpn-instance-name": "testVpn1",
            "name": "dpn1-dp1.3",
            "odl-l3vpn:adjacency": [
                {
                    "odl-l3vpn:mac_address": "11:22:33:44:55:66",
                    "odl-l3vpn:ip_address": "11.11.11.2",
                }
            ]
        }
    ]
}
```

Following is expected as a result of these configurations

1. Prefix label is generated and stored in DS
2. Ingress table is programmed with flow corresponding to interface
3. Local Egress Group is created

4. Prefix is added to BGP for advertisement
5. BGP pushes route update to FIB YANG Interface
6. FIB Entry flow is added to FIB Table in OF pipeline

Support

Table of Contents

Verified Combinations

This section describes which versions of OpenStack and Open vSwitch are expected to work with with OpenDaylight. Using combinations outside this list may work but have not been verified.

Note: Verified is defined as combinations that are actively tested and maintained. OpenDaylight, OpenStack and Open vSwitch are very active and quickly adding new features that makes it difficult to verify all the different release combinations. Different combinations are likely to work but support will be limited.

The following table details the expected supported combinations.

Table 1.5: Supported Version Matrix

OpenDaylight	OpenStack	Open vSwitch	Sync	Notes
Boron	Newton	2.6	S	
Carbon	Ocata	2.7		Combination drops when Pike releases
Carbon	Pike	2.7	S	
Nitrogen	Ocata	2.7		Combination drops when Pike releases
Nitrogen	Pike	2.7		Combination drops when Queens releases
Nitrogen	Queens	2.8/2.9	S	
Oxygen	Pike	2.7		Combination drops when Queens releases
Oxygen	Queens	2.8/2.9		Combination drops when OpenStack R releases
Oxygen	R	2.9	S	

- (S): in the Sync column indicates the final supported combination for that OpenDaylight release.
- Differing release schedules will lead to short-lived combinations that will drop as the releases line up. An example is with Carbon that releases before Pike so for a period of time Carbon is supported with Ocata.
- The current OpenDaylight version and the previous will be supported. Boron support will drop when Nitrogen releases; Carbon support will drop when Oxygen releases.

Open vSwitch Kernel and DPDK Modes

The table below lists the Open vSwitch requirements for the Carbon release.

Table 1.6: Kernel and DPDK Modes

Feature	OVS 2.6 kernel mode	OVS 2.6 dpdk mode
Conntrack - security groups	yes	yes
Conntrack - NAT	yes	no (target 2.8*)
Security groups stateful	yes (conntrack)	yes(conntrack)
Security groups learn	yes (but not needed)	yes (but not needed)
IPV4 NAT (without pkt punt to controller)	yes (conntrack)	no (target 2.8*)
IPV4 NAT (with pkt punt to controller)	not needed	yes (until 2.8*)

(*) support is tentatively scheduled for Open vSwitch 2.8

Bridge Configuration

Table of Contents

The following describes OVS bridge configurations supported by OpenDaylight.

The "br-int" Bridge

If the br-int bridge is not configured prior to the ovsdb manager connection with ODL, ODL will create it. If br-int exists prior to the ovsdb manager connection, ODL will retain any existing configurations on br-int. Note that if you choose to create br-int prior to connecting to ODL, `disable-in-band` MUST be set to true and any flows configured may interfere with the flows ODL will create. ODL will add the following configuration items to br-int:

1. ODL will set itself as br-int's controller
2. Any provider network configuration (see section "Provider Networks" below)

It is important to note that once the ovsdb manager connection is established with ODL, ODL "owns" br-int and other applications should not modify its settings.

Provider Networks

Provider networks should be configured prior to OVSDB connecting to ODL. These are configured in the Open_vSwitch table's other_Config column and have the format `<physnet>:<connector>` where `<physnet>` is the name of the provider network and `<connector>` is one of the following three options:

1. The name of a local interface (ODL will add this port to br-int)
2. The name of a bridge on OpenVSwitch (ODL will create patch ports between br-int and this bridge)
3. The name of a port already present on br-int (ODL will use that port)

For example, assume your provider network is called extnet and it is attached to the eth0 interface on your host you can set this in OVSDB using the following command:

```
sudo ovs-vsctl set Open_vSwitch . Other_Config:provider_mappings=extnet:eth0
```

If instead of `eth0` the provider network is accesable via on OVS bridge called `br-int`, `eth0` in the above command would be substituted with `br-int`.

Neutron Service User Guide

Overview

This Karaf feature (`odl-neutron-service`) provides integration support for OpenStack Neutron via the OpenDaylight ML2 mechanism driver. The Neutron Service is only one of the components necessary for OpenStack integration. For those related components please refer to documentations of each component:

- https://wiki.openstack.org/wiki/Neutron
- https://launchpad.net/networking-odl
- http://git.openstack.org/cgit/openstack/networking-odl/
- https://wiki.opendaylight.org/view/NeutronNorthbound:Main

Use cases and who will use the feature

If you want OpenStack integration with OpenDaylight, you will need this feature with an OpenDaylight provider feature like ovsdb/netvirt, group based policy, VTN, and lisp mapper. For provider configuration, please refer to each individual provider's documentation. Since the Neutron service only provides the northbound API for the OpenStack Neutron ML2 mechanism driver. Without those provider features, the Neutron service itself isn't useful.

Neutron Service feature Architecture

The Neutron service provides northbound API for OpenStack Neutron via RESTCONF and also its dedicated REST API. It communicates through its YANG model with providers.

Configuring Neutron Service feature

As the Karaf feature includes everything necessary for communicating northbound, no special configuration is needed. Usually this feature is used with an OpenDaylight southbound plugin that implements actual network virtualization functionality and OpenStack Neutron. The user wants to setup those configurations. Refer to each related documentations for each configurations.

Administering or Managing `odl-neutron-service`

There is no specific configuration regarding to Neutron service itself. For related configuration, please refer to OpenStack Neutron configuration and OpenDaylight related services which are providers for OpenStack.

installing `odl-neutron-service` while the controller running

1. While OpenDaylight is running, in Karaf prompt, type: `feature:install odl-neutron-service`.
2. Wait a while until the initialization is done and the controller stabilizes.

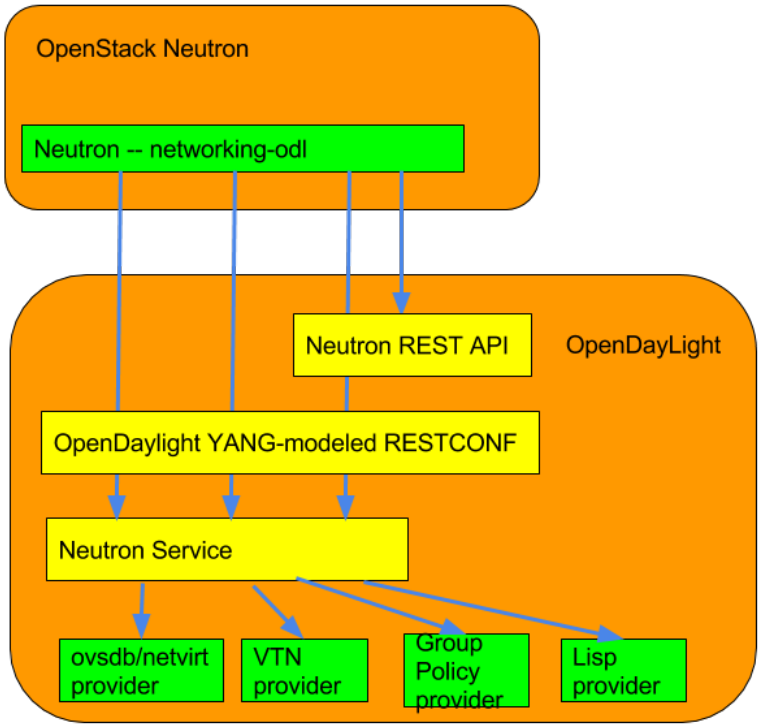


Fig. 1.71: Neutron Service Architecture

`odl-neutron-service` provides only a unified interface for OpenStack Neutron. It doesn't provide actual functionality for network virtualization. Refer to each OpenDaylight project documentation for actual configuration with OpenStack Neutron.

Neutron Logger

Another service, the Neutron Logger, is provided for debugging/logging purposes. It logs changes on Neutron YANG models.

```
feature:install odl-neutron-logger
```

Network Intent Composition (NIC) User Guide

Overview

Network Intent Composition (NIC) is an interface that allows clients to express a desired state in an implementation-neutral form that will be enforced via modification of available resources under the control of the OpenDaylight system.

This description is purposely abstract as an intent interface might encompass network services, virtual devices, storage, etc.

The intent interface is meant to be a controller-agnostic interface so that "intents" are portable across implementations, such as OpenDaylight and ONOS. Thus an intent specification should not contain implementation or technology specifics.

The intent specification will be implemented by decomposing the intent and augmenting it with implementation specifics that are driven by local implementation rules, policies, and/or settings.

Network Intent Composition (NIC) Architecture

The core of the NIC architecture is the intent model, which specifies the details of the desired state. It is the responsibility of the NIC implementation transforms this desired state to the resources under the control of OpenDaylight. The component that transforms the intent to the implementation is typically referred to as a renderer.

For the Boron release, multiple, simultaneous renderers will not be supported. Instead either the VTN or GBP renderer feature can be installed, but not both.

For the Boron release, the only actions supported are "ALLOW" and "BLOCK". The "ALLOW" action indicates that traffic can flow between the source and destination end points, while "BLOCK" prevents that flow; although it is possible that an given implementation may augment the available actions with additional actions.

Besides transforming a desired state to an actual state it is the responsibility of a renderer to update the operational state tree for the NIC data model in OpenDaylight to reflect the intent which the renderer implemented.

Configuring Network Intent Composition (NIC)

For the Boron release there is no default implementation of a renderer, thus without an additional module installed the NIC will not function.

Administering or Managing Network Intent Composition (NIC)

There is no additional administration of management capabilities related to the Network Intent Composition features.

Interactions

A user can interact with the Network Intent Composition (NIC) either through the RESTful interface using standard RESTCONF operations and syntax or via the Karaf console CLI.

REST

Configuration

The Network Intent Composition (NIC) feature supports the following REST operations against the configuration data store.

- POST - creates a new instance of an intent in the configuration store, which will trigger the realization of that intent. An ID *must* be specified as part of this request as an attribute of the intent.
- GET - fetches a list of all configured intents or a specific configured intent.
- DELETE - removes a configured intent from the configuration store, which triggers the removal of the intent from the network.

Operational

The Network Intent Composition (NIC) feature supports the following REST operations against the operational data store.

- GET - fetches a list of all operational intents or a specific operational intent.

Karaf Console CLI

This feature provides karaf console CLI command to manipulate the intent data model. The CLI essentailly invokes the equivalent data operations.

intent:add

Creates a new intent in the configuration data tree

```
DESCRIPTION
        intent:add

    Adds an intent to the controller.

Examples: --actions [ALLOW] --from <subject> --to <subject>
          --actions [BLOCK] --from <subject>

SYNTAX
        intent:add [options]
```

```
OPTIONS
        -a, --actions
                Action to be performed.
                -a / --actions BLOCK/ALLOW
                (defaults to [BLOCK])
        --help
                Display this help message
        -t, --to
                Second Subject.
                -t / --to <subject>
                (defaults to any)
        -f, --from
                First subject.
                -f / --from <subject>
                (defaults to any)
```

intent:delete

Removes an existing intent from the system

```
DESCRIPTION
        intent:remove

    Removes an intent from the controller.

SYNTAX
        intent:remove id

ARGUMENTS
        id  Intent Id
```

intent:list

Lists all the intents in the system

```
DESCRIPTION
        intent:list

    Lists all intents in the controller.

SYNTAX
        intent:list [options]

OPTIONS
        -c, --config
                List Configuration Data (optional).
                -c / --config <ENTER>
        --help
                Display this help message
```

intent:show

Displayes the details of a single intent

```
DESCRIPTION
        intent:show

    Shows detailed information about an intent.

SYNTAX
        intent:show id

ARGUMENTS
        id  Intent Id
```

intent:map

List/Add/Delete current state from/to the mapping service.

```
DESCRIPTION
        intent:map

        List/Add/Delete current state from/to the mapping service.

SYNTAX
        intent:map [options]

         Examples: --list, -l [ENTER], to retrieve all keys.
                   --add-key <key> [ENTER], to add a new key with empty contents.
                   --del-key <key> [ENTER], to remove a key with it's values."
                   --add-key <key> --value [<value 1>, <value 2>, ...] [ENTER],
                     to add a new key with some values (json format).
OPTIONS
       --help
           Display this help message
       -l, --list
           List values associated with a particular key.
       -l / --filter <regular expression> [ENTER]
       --add-key
           Adds a new key to the mapping service.
       --add-key <key name> [ENTER]
       --value
           Specifies which value should be added/delete from the mapping service.
       --value "key=>value"... --value "key=>value" [ENTER]
           (defaults to [])
       --del-key
           Deletes a key from the mapping service.
       --del-key <key name> [ENTER]
```

NIC Usage Examples

Default Requirements

Start mininet, and create three switches (s1, s2, and s3) and four hosts (h1, h2, h3, and h4) in it.

Replace <Controller IP> based on your environment.

```
$  sudo mn --mac --topo single,2 --controller=remote,ip=<Controller IP>
```

```
mininet> net
h1 h1-eth0:s2-eth1
h2 h2-eth0:s2-eth2
h3 h3-eth0:s3-eth1
h4 h4-eth0:s3-eth2
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s3-eth3
s2 lo:  s2-eth1:h1-eth0 s2-eth2:h2-eth0 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h3-eth0 s3-eth2:h4-eth0 s3-eth3:s1-eth2
```

Downloading and deploy Karaf distribution

- Get the Boron distribution.
- Unzip the downloaded zip distribution.
- To run the Karaf.

```
./bin/karaf
```

- Once the console is up, type as below to install feature.

```
feature:install odl-nic-core-mdsal odl-nic-console odl-nic-listeners
```

Simple Mininet topology

```
!/usr/bin/python

from mininet.topo import Topo

class SimpleTopology( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."


    Topo.__init__( self )


        Switch1 = self.addSwitch( 's1' )
        Switch2 = self.addSwitch( 's2' )
        Switch3 = self.addSwitch( 's3' )
        Switch4 = self.addSwitch( 's4' )
        Host11 = self.addHost( 'h1' )
        Host12 = self.addHost( 'h2' )
        Host21 = self.addHost( 'h3' )
        Host22 = self.addHost( 'h4' )
        Host23 = self.addHost( 'h5' )
        Service1 = self.addHost( 'srvc1' )


        self.addLink( Host11, Switch1 )
```

```
        self.addLink( Host12, Switch1 )
        self.addLink( Host21, Switch2 )
        self.addLink( Host22, Switch2 )
        self.addLink( Host23, Switch2 )
        self.addLink( Switch1, Switch2 )
        self.addLink( Switch2, Switch4 )
        self.addLink( Switch4, Switch3 )
        self.addLink( Switch3, Switch1 )
        self.addLink( Switch3, Service1 )
        self.addLink( Switch4, Service1 )


topos = { 'simpletopology': ( lambda: SimpleTopology() ) }
```

- Initialize topology
- Add hosts and switches
- Host used to represent the service
- Add links

  Source: https://gist.github.com/vinothgithub15/315d0a427d5afc39f2d7

How to configure VTN Renderer

The section demonstrates allow or block packets of the traffic within a VTN Renderer, according to the specified flow conditions.

The table below lists the actions to be applied when a packet matches the condition:

Action	Function
Allow	Permits the packet to be forwarded normally.
Block	Discards the packet preventing it from being forwarded.

Requirement

- Before execute the follow steps, please, use default requirements. See section *Default Requirements*.

Configuration

Please execute the following curl commands to test network intent using mininet:

Create Intent

To provision the network for the two hosts(h1 and h2) and demonstrates the action allow.

```
curl -v --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X PUT http://localhost:8181/restconf/config/intent:intents/
↪intent/b9a13232-525e-4d8c-be21-cd65e3436034 -d '{ "intent:intent" : { "intent:id":
↪"b9a13232-525e-4d8c-be21-cd65e3436034", "intent:actions" : [ { "order" : 2, "allow"␣
↪: {} } ], "intent:subjects" : [ { "order":1 , "end-point-group" : {"name":"10.0.0.1
↪"} }, { "order":2 , "end-point-group" : {"name":"10.0.0.2"}} ] } }'
```

To provision the network for the two hosts(h2 and h3) and demonstrates the action allow.

```
curl -v --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X PUT http://localhost:8181/restconf/config/intent:intents/
↪intent/b9a13232-525e-4d8c-be21-cd65e3436035 -d '{ "intent:intent" : { "intent:id":
↪"b9a13232-525e-4d8c-be21-cd65e3436035", "intent:actions" : [ { "order" : 2, "allow"␣
↪: {} } ], "intent:subjects" : [ { "order":1 , "end-point-group" : {"name":"10.0.0.2
↪"} }, { "order":2 , "end-point-group" : {"name":"10.0.0.3"}} ] } }'
```

Verification

As we have applied action type allow now ping should happen between hosts (h1 and h2) and (h2 and h3).

```
mininet> pingall
Ping: testing ping reachability
h1 -> h2 X X
h2 -> h1 h3 X
h3 -> X h2 X
h4 -> X X X
```

Update the intent

To provision block action that indicates traffic is not allowed between h1 and h2.

```
curl -v --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X PUT http://localhost:8181/restconf/config/intent:intents/
↪intent/b9a13232-525e-4d8c-be21-cd65e3436034 -d '{ "intent:intent" : { "intent:id":
↪"b9a13232-525e-4d8c-be21-cd65e3436034", "intent:actions" : [ { "order" : 2, "block"␣
↪: {} } ], "intent:subjects" : [ { "order":1 , "end-point-group" : {"name":"10.0.0.1
↪"} }, { "order":2 , "end-point-group" : {"name":"10.0.0.2"}} ] } }'
```

Verification

As we have applied action type block now ping should not happen between hosts (h1 and h2).

```
mininet> pingall
Ping: testing ping reachability
h1 -> X X X
h2 -> X h3 X
h3 -> X h2 X
h4 -> X X X
```

Note: Old actions and hosts are replaced by the new action and hosts.

Delete the intent

Respective intent and the traffics will be deleted.

```
curl -v --user "admin":"admin" -H "Accept: application/json" -H     "Content-type:␣
↪application/json" -X DELETE http://localhost:8181/restconf/config/intent:intents/
↪intent/b9a13232-525e-4d8c-be21-cd65e3436035
```

Verification

Deletion of intent and flow.

```
mininet> pingall
Ping: testing ping reachability
h1 -> X X X
h2 -> X X X
h3 -> X X X
h4 -> X X X
```

Note: Ping between two hosts can also be done using MAC Address

To provision the network for the two hosts(h1 MAC address and h2 MAC address).

```
curl -v --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X PUT http://localhost:8181/restconf/config/intent:intents/
↪intent/b9a13232-525e-4d8c-be21-cd65e3436035 -d '{ "intent:intent" : { "intent:id":
↪"b9a13232-525e-4d8c-be21-cd65e3436035", "intent:actions" : [ { "order" : 2, "allow"␣
↪: {} } ], "intent:subjects" : [ { "order":1 , "end-point-group" : {"name":
↪"6e:4f:f7:27:15:c9"} }, { "order":2 , "end-point-group" : {"name":"aa:7d:1f:4a:70:81
↪"}} ] } }'
```

How to configure Redirect Action

The section explains the redirect action supported in NIC. The redirect functionality supports forwarding (to redirect) the traffic to a service configured in SFC before forwarding it to the destination.

Following steps explain Redirect action function:

- Configure the service in SFC using the SFC APIs.
- Configure the intent with redirect action and the service information where the traffic needs to be redirected.
- The flows are computed as below
  1. First flow entry between the source host connected node and the ingress node of the configured service.
  2. Second flow entry between the egress Node id the configured service and the ID and destination host connected host.
  3. Third flow entry between the destination host node and the source host node.

Requirement

- Save the mininet *Simple Mininet topology* script as redirect_test.py
- Start mininet, and create switches in it.

Replace <Controller IP> based on your environment.

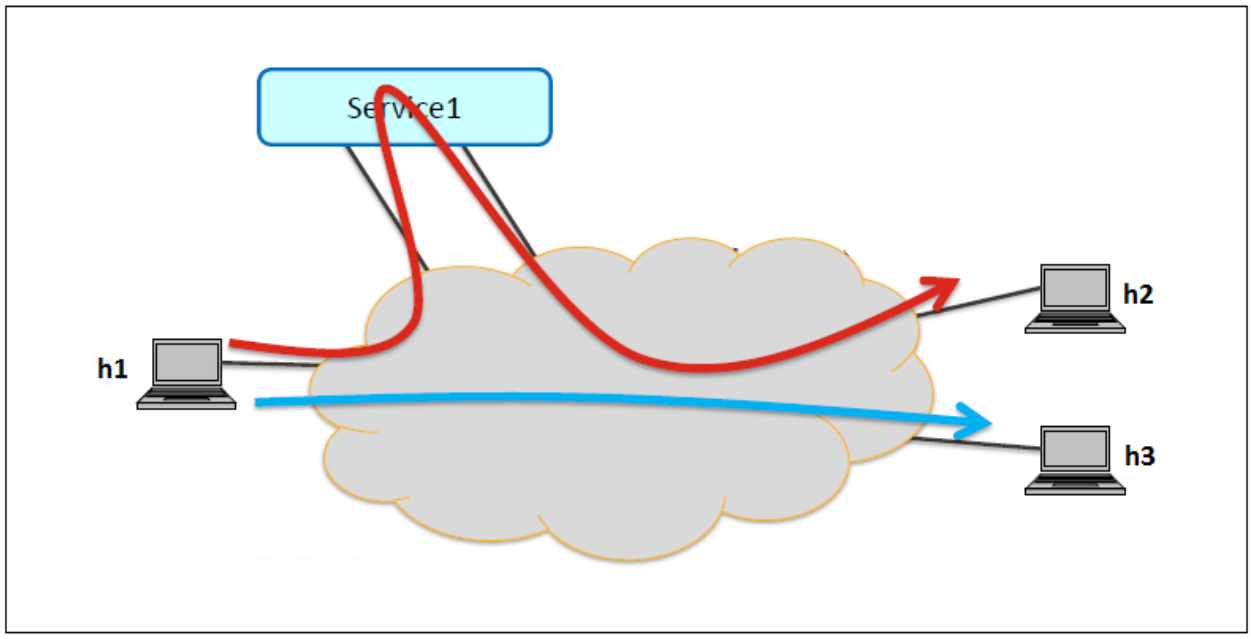


Fig. 1.72: REDIRECT SERVICE

```
sudo mn --controller=remote,ip=<Controller IP>--custom redirect_test.py --topo mytopo2
```

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s2-eth1
h4 h4-eth0:s2-eth2
h5 h5-eth0:s2-eth3
srvc1 srvc1-eth0:s3-eth3 srvc1-eth1:s4-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:s2-eth4 s1-eth4:s3-eth2
s2 lo:  s2-eth1:h3-eth0 s2-eth2:h4-eth0 s2-eth3:h5-eth0 s2-eth4:s1-eth3 s2-eth5:s4-
↪eth1
s3 lo:  s3-eth1:s4-eth2 s3-eth2:s1-eth4 s3-eth3:srvc1-eth0
s4 lo:  s4-eth1:s2-eth5 s4-eth2:s3-eth1 s4-eth3:srvc1-eth1
c0
```

Starting the Karaf

- Before execute the following steps, please, use the default requirements. See section *Downloading and deploy Karaf distribution*.

Configuration

Mininet

- Configure srvc1 as service node in the mininet environment.

Please execute the following commands in the mininet console (where mininet script is executed).

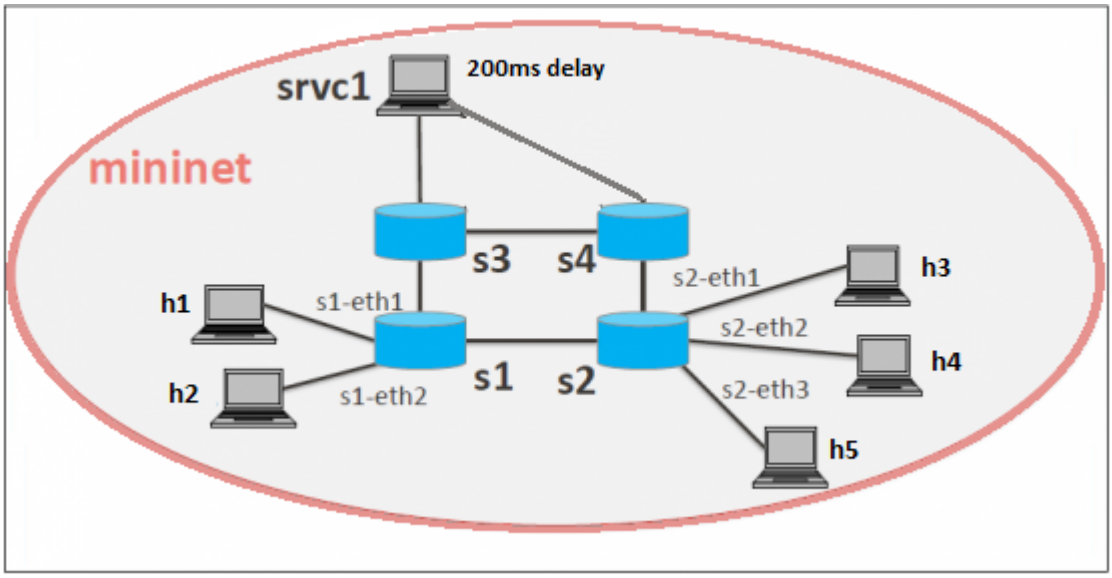


Fig. 1.73: CONFIGURATION THE NETWORK IN MININET

```
srvc1 ip addr del 10.0.0.6/8 dev srvc1-eth0
srvc1 brctl addbr br0
srvc1 brctl addif br0 srvc1-eth0
srvc1 brctl addif br0 srvc1-eth1
srvc1 ifconfig br0 up
srvc1 tc qdisc add dev srvc1-eth1 root netem delay 200ms
```

Configure service in SFC

The service (srvc1) is configured using SFC REST API. As part of the configuration the ingress and egress node connected the service is configured.

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '{
  "service-functions": {
    "service-function": [
      {
        "name": "srvc1",
        "sf-data-plane-locator": [
          {
            "name": "Egress",
            "service-function-forwarder": "openflow:4"
          },
          {
            "name": "Ingress",
            "service-function-forwarder": "openflow:3"
          }
        ],
        "nsh-aware": false,
        "type": "delay"
      }
    ]
  }
}' -X PUT --user admin:admin http://localhost:8181/restconf/config/service-
↪function:service-functions/
```

SFF RESTCONF Request

Configuring switch and port information for the service functions.

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '{
  "service-function-forwarders": {
    "service-function-forwarder": [
      {
        "name": "openflow:3",
        "service-node": "OVSDB2",
        "sff-data-plane-locator": [
          {
            "name": "Ingress",
            "data-plane-locator":
            {
                "vlan-id": 100,
                "mac": "11:11:11:11:11:11",
                "transport": "service-locator:mac"
            },
            "service-function-forwarder-ofs:ofs-port":
            {
                "port-id" : "3"
            }
          }
        ],
        "service-function-dictionary": [
          {
            "name": "srvc1",
            "sff-sf-data-plane-locator":
            {
                "sf-dpl-name" : "openflow:3",
                "sff-dpl-name" : "Ingress"
            }
          }
        ]
      },
      {
        "name": "openflow:4",
        "service-node": "OVSDB3",
        "sff-data-plane-locator": [
          {
            "name": "Egress",
            "data-plane-locator":
            {
                "vlan-id": 200,
                "mac": "44:44:44:44:44:44",
                "transport": "service-locator:mac"
            },
            "service-function-forwarder-ofs:ofs-port":
            {
                "port-id" : "3"
            }
          }
        ],
        "service-function-dictionary": [
          {
            "name": "srvc1",
```

```
            "sff-sf-data-plane-locator":
            {
                "sf-dpl-name" : "openflow:4",
                "sff-dpl-name" : "Egress"
            }
          }
        ]
      }
    ]
  }
}' -X PUT --user admin:admin http://localhost:8181/restconf/config/service-function-
↪forwarder:service-function-forwarders/
```

CLI Command

To provision the network for the two hosts (h1 and h5).

Demonstrates the redirect action with service name srvc1.

```
intent:add -f <SOURCE_MAC> -t <DESTINATION_MAC> -a REDIRECT -s <SERVICE_NAME>
```

Example:

```
intent:add -f 32:bc:ec:65:a7:d1 -t c2:80:1f:77:41:ed -a REDIRECT -s srvc1
```

Verification

- As we have applied action type redirect now ping should happen between hosts h1 and h5.

```
mininet> h1 ping h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=201 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=200 ms
64 bytes from 10.0.0.5: icmp_seq=4 ttl=64 time=200 ms
```

The redirect functionality can be verified by the time taken by the ping operation (200ms). The service srvc1 configured using SFC introduces 200ms delay. As the traffic from h1 to h5 is redirected via the srvc1, the time taken by the traffic from h1 to h5 will take about 200ms.

- Flow entries added to nodes for the redirect action.

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=9.406s, table=0, n_packets=6, n_bytes=588, idle_age=3,␣
↪priority=9000,in_port=1,dl_src=32:bc:ec:65:a7:d1, dl_dst=c2:80:1f:77:41:ed␣
↪actions=output:4
cookie=0x0, duration=9.475s, table=0, n_packets=6, n_bytes=588, idle_age=3,␣
↪priority=9000,in_port=3,dl_src=c2:80:1f:77:41:ed, dl_dst=32:bc:ec:65:a7:d1␣
↪actions=output:1
cookie=0x1, duration=362.315s, table=0, n_packets=144, n_bytes=12240, idle_age=4,␣
↪priority=9500,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x1, duration=362.324s, table=0, n_packets=4, n_bytes=168, idle_age=3,␣
↪priority=10000,arp actions=CONTROLLER:65535,NORMAL
*** s2 ------------------------------------------------------------------------
```

```
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=9.503s, table=0, n_packets=6, n_bytes=588, idle_age=3,␣
↪priority=9000,in_port=3,dl_src=c2:80:1f:77:41:ed, dl_dst=32:bc:ec:65:a7:d1␣
↪actions=output:4
cookie=0x0, duration=9.437s, table=0, n_packets=6, n_bytes=588, idle_age=3,␣
↪priority=9000,in_port=5,dl_src=32:bc:ec:65:a7:d1, dl_dst=c2:80:1f:77:41:ed␣
↪actions=output:3
cookie=0x3, duration=362.317s, table=0, n_packets=144, n_bytes=12240, idle_age=4,␣
↪priority=9500,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x3, duration=362.32s, table=0, n_packets=4, n_bytes=168, idle_age=3,␣
↪priority=10000,arp actions=CONTROLLER:65535,NORMAL
*** s3 ----------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=9.41s, table=0, n_packets=6, n_bytes=588, idle_age=3,␣
↪priority=9000,in_port=2,dl_src=32:bc:ec:65:a7:d1, dl_dst=c2:80:1f:77:41:ed␣
↪actions=output:3
*** s4 ----------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=9.486s, table=0, n_packets=6, n_bytes=588, idle_age=3,␣
↪priority=9000,in_port=3,dl_src=32:bc:ec:65:a7:d1, dl_dst=c2:80:1f:77:41:ed␣
↪actions=output:1
```

How to configure QoS Attribute Mapping

This section explains how to provision QoS attribute mapping constraint using NIC OF-Renderer.

The QoS attribute mapping currently supports DiffServ. It uses a 6-bit differentiated services code point (DSCP) in the 8-bit differentiated services field (DS field) in the IP header.

Action	Function
Allow	Permits the packet to be forwarded normally, but allows for packet header fields, e.g., DSCP, to be modified.

The following steps explain QoS Attribute Mapping function:

- Initially configure the QoS profile which contains profile name and DSCP value.
- When a packet is transferred from a source to destination, the flow builder evaluates whether the transferred packet matches the condition such as action, endpoints in the flow.
- If the packet matches the endpoints, the flow builder applies the flow matching action and DSCP value.

Requirement

- Before execute the following steps, please, use the default requirements. See section *Default Requirements*.

Configuration

Please execute the following CLI commands to test network intent using mininet:

- To apply the QoS constraint, configure the QoS profile.

```
intent:qosConfig -p <qos_profile_name> -d <valid_dscp_value>
```

Example:

```
intent:qosConfig -p High_Quality -d 46
```

Note: Valid DSCP value ranges from 0-63.

- To provision the network for the two hosts (h1 and h3), add intents that allows traffic in both directions by execute the following CLI command.

Demonstrates the ALLOW action with constraint QoS and QoS profile name.

```
intent:add -a ALLOW -t <DESTINATION_MAC> -f <SOURCE_MAC> -q QOS -p <qos_profile_name>
```

Example:

```
intent:add -a ALLOW -t 00:00:00:00:00:03 -f 00:00:00:00:00:01 -q QOS -p High_Quality
intent:add -a ALLOW -t 00:00:00:00:00:01 -f 00:00:00:00:00:03 -q QOS -p High_Quality
```

Verification

- As we have applied action type ALLOW now ping should happen between hosts h1 and h3.

```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=0.984 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=0.110 ms
64 bytes from 10.0.0.3: icmp_req=3 ttl=64 time=0.098 ms
```

- Verification of the flow entry and ensuring the mod_nw_tos is part of actions.

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=21.873s, table=0, n_packets=3, n_bytes=294, idle_age=21,␣
↪priority=9000,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01 actions=NORMAL,mod_
↪nw_tos:184
cookie=0x0, duration=41.252s, table=0, n_packets=3, n_bytes=294, idle_age=41,␣
↪priority=9000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03 actions=NORMAL,mod_
↪nw_tos:184
```

Requirement

- Before execute the follow steps, please, use default requirements. See section *Default Requirements*.

How to configure Log Action

This section demonstrates log action in OF Renderer. This demonstration aims at enabling communication between two hosts and logging the flow statistics details of the particular traffic.

Configuration

Please execute the following CLI commands to test network intent using mininet:

- To provision the network for the two hosts (h1 and h3), add intents that allows traffic in both directions by execute the following CLI command.

```
intent:add -a ALLOW -t <DESTINATION_MAC> -f <SOURCE_MAC>
```

Example:

```
intent:add -a ALLOW -t 00:00:00:00:00:03 -f 00:00:00:00:00:01
intent:add -a ALLOW -t 00:00:00:00:00:01 -f 00:00:00:00:00:03
```

- To log the flow statistics details of the particular traffic.

```
intent:add -a LOG -t <DESTINATION_MAC> -f <SOURCE_MAC>
```

Example:

```
intent:add -a LOG -t 00:00:00:00:00:03 -f 00:00:00:00:00:01
```

Verification

- As we have applied action type ALLOW now ping should happen between hosts h1 and h3.

```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=0.984 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=0.110 ms
64 bytes from 10.0.0.3: icmp_req=3 ttl=64 time=0.098 ms
```

- To view the flow statistics log details such as, byte count, packet count and duration, check the karaf.log.

```
2015-12-15 22:56:20,256 | INFO | lt-dispatcher-23 | IntentFlowManager | 264 - org.
↪opendaylight.nic.of-renderer - 1.1.0.SNAPSHOT | Creating block intent for␣
↪endpoints: source00:00:00:00:00:01 destination 00:00:00:00:00:03
2015-12-15 22:56:20,252 | INFO | lt-dispatcher-29 | FlowStatisticsListener | 264 -␣
↪org.opendaylight.nic.of-renderer - 1.1.0.SNAPSHOT | Flow Statistics gathering for␣
↪Byte Count:Counter64 [_value=238]
2015-12-15 22:56:20,252 | INFO | lt-dispatcher-29 | FlowStatisticsListener | 264 -␣
↪org.opendaylight.nic.of-renderer - 1.1.0.SNAPSHOT | Flow Statistics gathering for␣
↪Packet Count:Counter64 [_value=3]
2015-12-15 22:56:20,252 | INFO | lt-dispatcher-29 | FlowStatisticsListener | 264 -␣
↪org.opendaylight.nic.of-renderer - 1.1.0.SNAPSHOT | Flow Statistics gathering for␣
↪Duration in Nano second:Counter32 [_value=678000000]
2015-12-15 22:56:20,252 | INFO | lt-dispatcher-29 | FlowStatisticsListener | 264 -␣
↪org.opendaylight.nic.of-renderer - 1.1.0.SNAPSHOT | Flow Statistics gathering for␣
↪Duration in Second:Counter32 [_value=49]
```

OCP Plugin User Guide

This document describes how to use the ORI Control & Management Protocol (OCP) feature in OpenDaylight. This document contains overview, scope, architecture and design, installation, configuration and tutorial sections for the feature.

Overview

OCP is an ETSI standard protocol for control and management of Remote Radio Head (RRH) equipment. The OCP Project addresses the need for a southbound plugin that allows applications and controller services to interact with RRHs using OCP. The OCP southbound plugin will allow applications acting as a Radio Equipment Control (REC) to interact with RRHs that support an OCP agent.

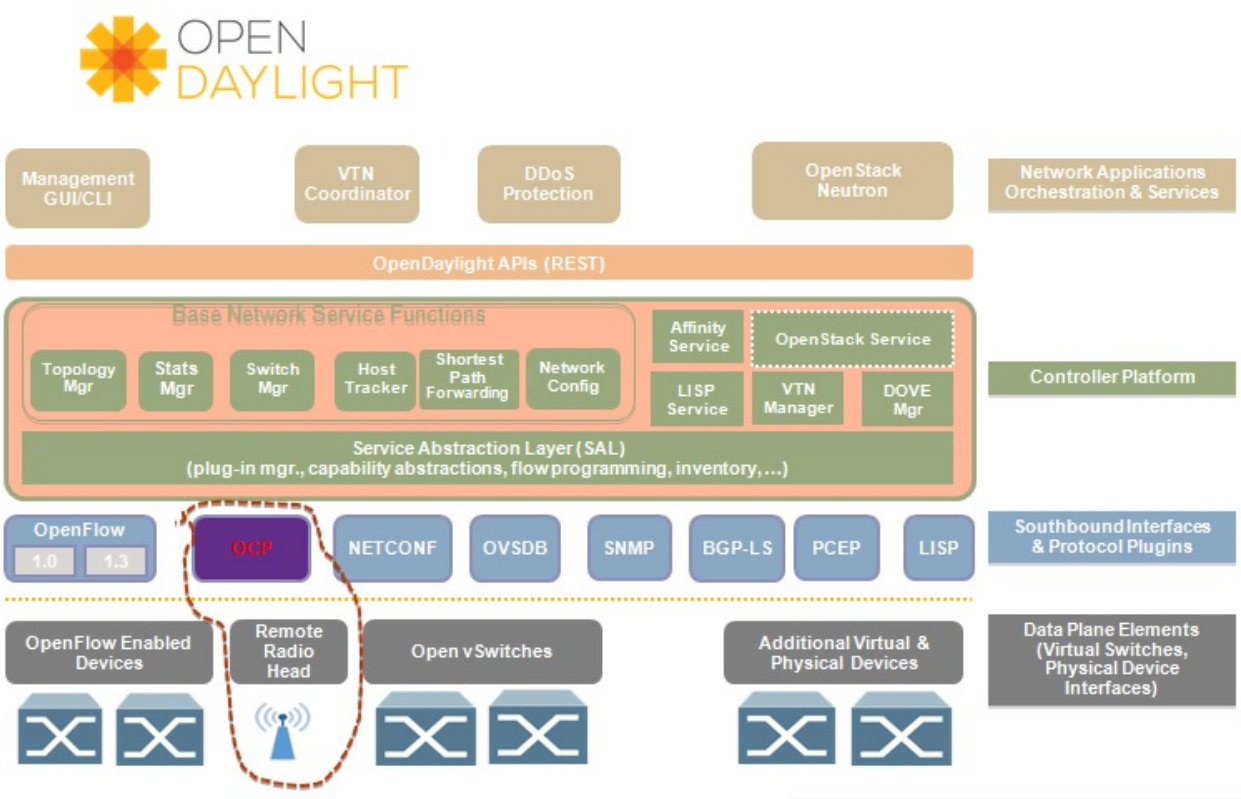


Fig. 1.74: OCP southbound plugin

It is foreseen that, in 5G, C-RAN will use the packet-based Transport-SDN (T-SDN) as the fronthaul network to transport both control plane and user plane data between RRHs and BBUs. As a result, the addition of the OCP plugin to OpenDaylight will make it possible to build an RRH controller on top of OpenDaylight to centrally manage deployed RRHs, as well as integrating the RRH controller with T-SDN on one single platform, achieving the joint RRH and fronthaul network provisioning in C-RAN.

Scope

The OCP Plugin project includes:

- OCP v4.1.1 support
- Integration of OCP protocol library
- Simple API invoked as a RPC
- Simple API that allows applications to perform elementary functions of the following categories:
  - Device management

  - Config management
  - Object lifecycle
  - Object state management
  - Fault management
  - Software management (not yet implemented)
- Indication processing
- Logging (not yet implemented)
- AISG/Iuant interface message tunnelling (not yet implemented)
- ALD connection management (not yet implemented)

Architecture and Design

OCP is a vendor-neutral standard communications interface defined to enable control and management between RE and REC of an ORI architecture. The OCP Plugin supports the implementation of the OCP specification; it is based on the Model Driven Service Abstraction Layer (MD-SAL) architecture.

OCP Plugin will support the following functionality:

- Connection handling
- Session management
- State management
- Error handling
- Connection establishment will be handled by OCP library using opensource netty.io library
- Message handling
- Event/indication handling and propagation to upper layers

Activities in OCP plugin module

- Integration with OCP protocol library
- Integration with corresponding MD-SAL infrastructure

OCP protocol library is a component in OpenDaylight that mediates communication between OpenDaylight controller and RRHs supporting OCP protocol. Its primary goal is to provide the OCP Plugin with communication channel that can be used for managing RRHs.

Key objectives:

- Immutable transfer objects generation (transformation of OCP protocol library's POJO objects into OpenDaylight DTO objects)
- Scalable non-blocking implementation
- Pipeline processing
- Scatter buffer
- TLS support

OCP Service addresses the need for a northbound interface that allows applications and other controller services to interact with RRHs using OCP, by providing API for abstracting OCP operations.

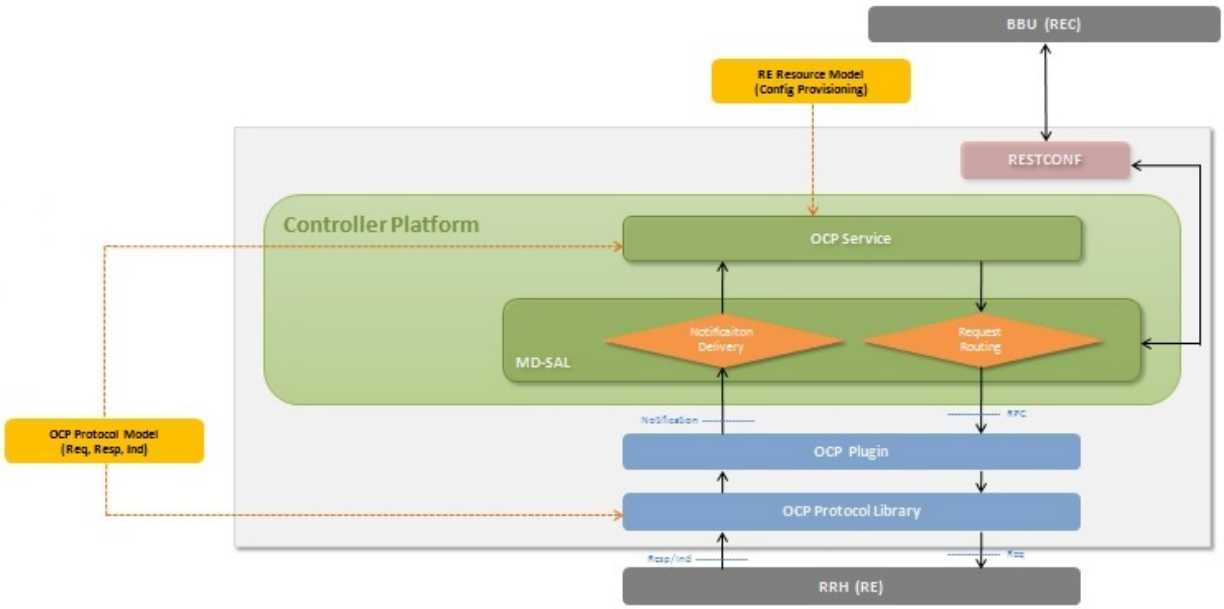


Fig. 1.75: Overall architecture

Message Flow

Installation

The OCP Plugin project has two top level Karaf features, odl-ocpplugin-all and odl-ocpjava-all, which contain the following sub-features:

- odl-ocpplugin-southbound
- odl-ocpplugin-app-ocp-service
- odl-ocpjava-protocol

The OCP service (odl-ocpplugin-app-ocp-service), together with the OCP southbound (odl-ocpplugin-southbound) and OCP protocol library (odl-ocpjava-protocol), provides OpenDaylight with basic OCP v4.1.1 functionality.

There are two ways to interact with OCP service: one is via RESTCONF (programmatic) and the other is using DLUX web interface (manual), so you have to install the following features to enable RESTCONF and DLUX.

```
karaf#>feature:install odl-restconf odl-l2switch-switch odl-mdsal-apidocs odl-dlux-
↪core odl-dluxapps-applications
```

Then install the odl-ocpplugin-all feature which includes the odl-ocpplugin-southbound and odl-ocpplugin-app-ocp-service features. Note that the odl-ocpjava-all feature will be installed automatically as the odl-ocpplugin-southbound feature is dependent on the odl-ocpjava-protocol feature.

```
karaf#>feature:install odl-ocpplugin-all
```

After all required features are installed, use following command from karaf console to check and make sure features are correctly installed and initialized.

```
karaf#>feature:list | grep ocp
```

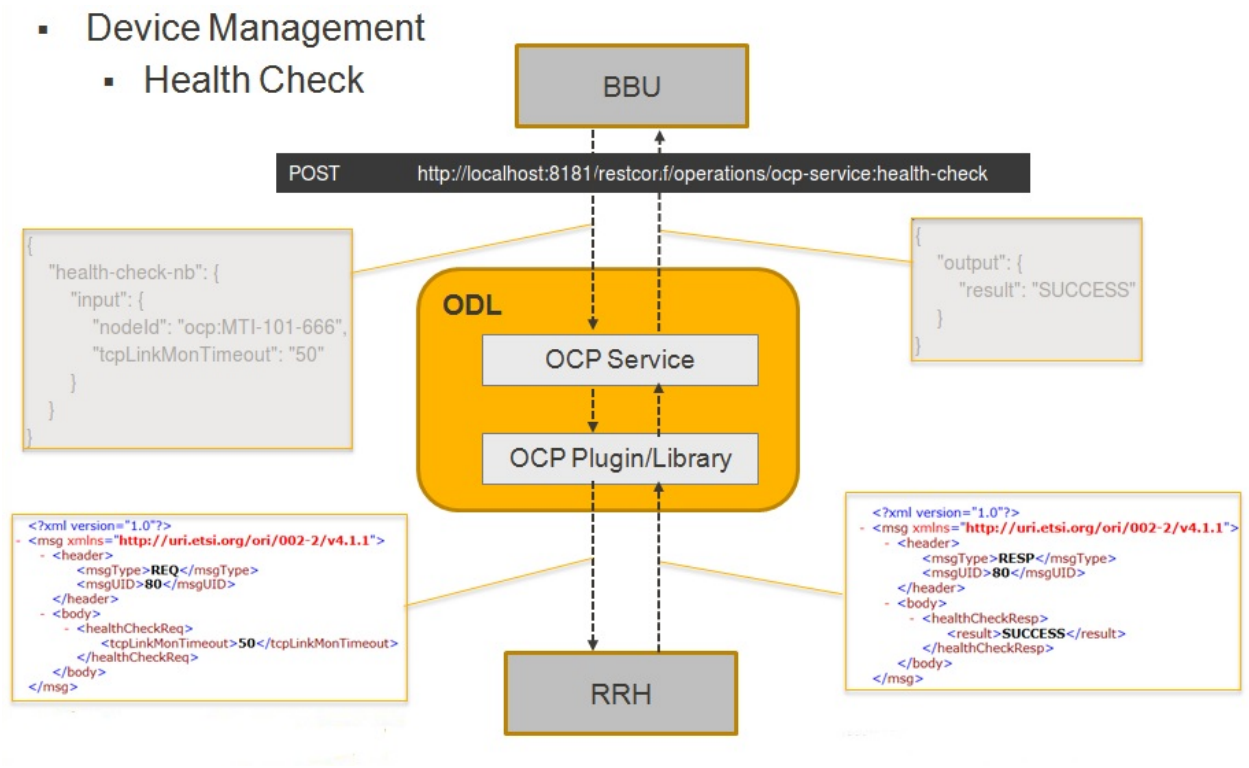


Fig. 1.76: Message flow example

Configuration

Configuring the OCP plugin can be done via its configuration file, 62-ocpplugin.xml, which can be found in the <odl-install-dir>/etc/opendaylight/karaf/ directory.

There are the following settings that are configurable:

1. **port** specifies the port number on which the OCP plugin listens for connection requests
2. **radioHead-idle-timeout** determines the time duration (unit: milliseconds) for which a radio head has been idle before the idle event is triggered to perform health check
3. **ocp-version** specifies the OCP protocol version supported by the OCP plugin
4. **rpc-requests-quota** sets the maximum number of concurrent rpc requests allowed
5. **global-notification-quota** sets the maximum number of concurrent notifications allowed

Test Environment

The OCP Plugin project contains a simple OCP agent for testing purposes; the agent has been designed specifically to act as a fake radio head device, giving you an idea of what it would look like during the OCP handshake taking place between the OCP agent and OpenDaylight (OCP plugin).

To run the simple OCP agent, you have to first download its JAR file from OpenDaylight Nexus Repository.

```
wget https://nexus.opendaylight.org/content/repositories/opendaylight.release/org/
↪opendaylight/ocpplugin/simple-agent/${ocp-version}/simple-agent-${ocp-version}.jar
```

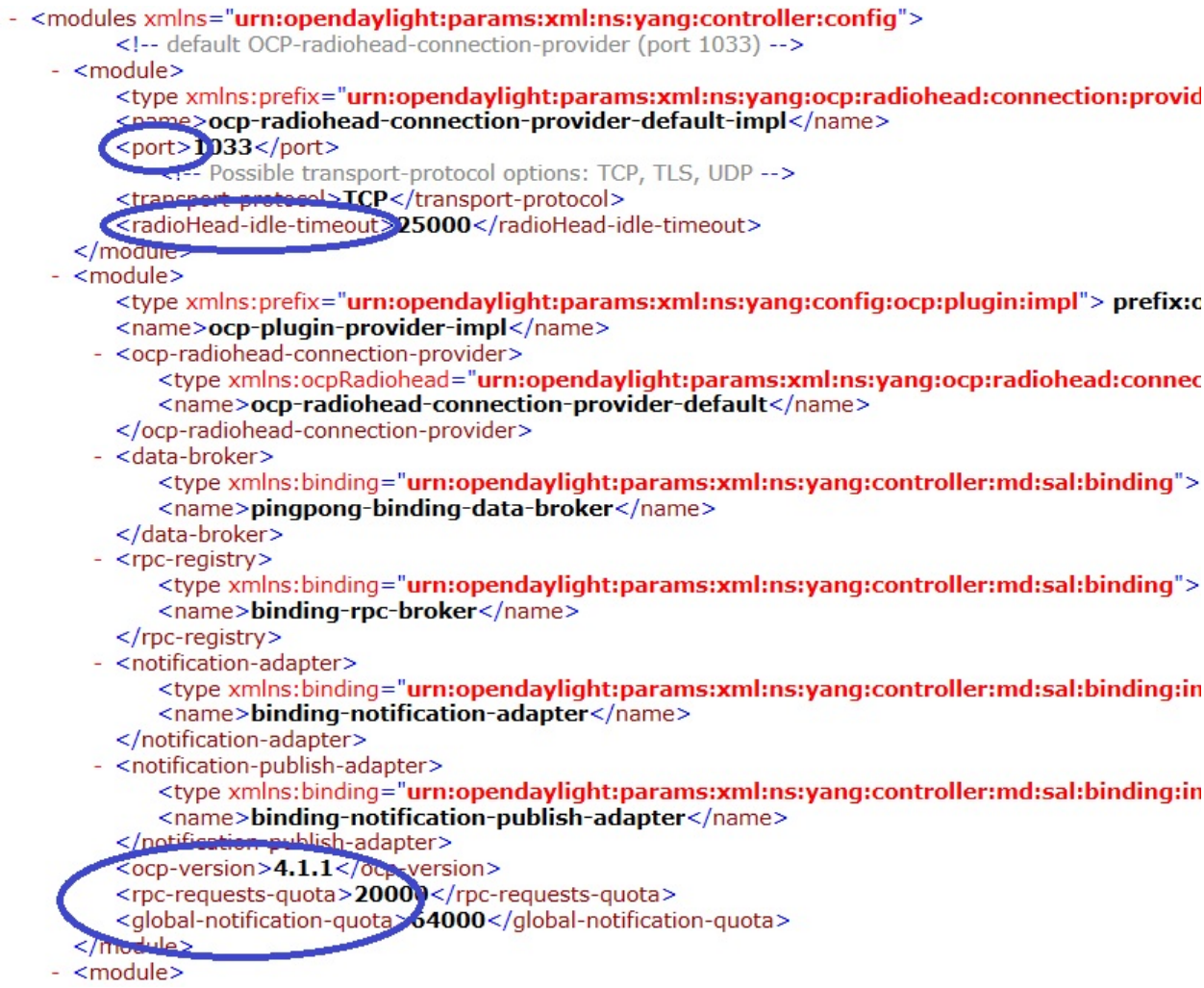


Fig. 1.77: OCP plugin configuration

Then run the agent with no arguments (assuming you already have JDK 1.8 or above installed) and it should display the usage that lists the expected arguments.

```
java -classpath simple-agent-${ocp-version}.jar org.opendaylight.ocpplugin.OcpAgent

Usage: java org.opendaylight.ocpplugin.OcpAgent <controller's ip address> <port␣
↪number> <vendor id> <serial number>
```

Here is an example:

```
java -classpath simple-agent-${ocp-version}.jar org.opendaylight.ocpplugin.OcpAgent␣
↪127.0.0.1 1033 XYZ 123
```

Web / Graphical Interface

Once you enable the DLUX feature, you can access the Controller GUI using following URL.

```
http://<controller-ip>:8080/index.html
```

Expand Nodes. You should see all the radio head devices that are connected to the controller running at <controller-ip>.

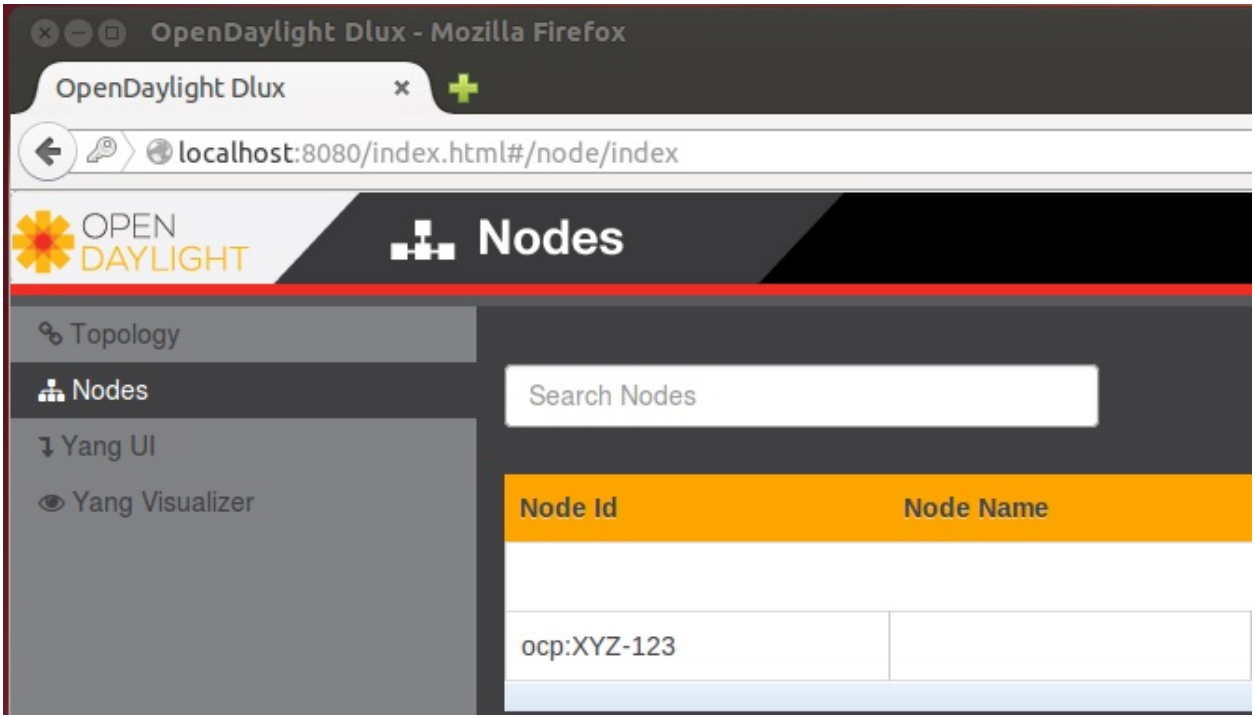


Fig. 1.78: DLUX Nodes

And expand Yang UI if you want to browse the various northbound APIs exposed by the OCP service.

For information on how to use these northbound APIs, please refer to the OCP Plugin Developer Guide.

Fig. 1.79: DLUX Yang UI

Programmatic Interface

The OCP Plugin project has implemented a complete set of the C&M operations (elementary functions) defined in the OCP specification, in the form of both northbound and southbound APIs, including:

- health-check
- set-time
- re-reset
- get-param
- modify-param
- create-obj
- delete-obj
- get-state
- modify-state
- get-fault

The API is documented in the OCP Plugin Developer Guide under the section Southbound API and Northbound API, respectively.

ODL-SDNi User Guide

Introduction

This user guide will help to setup the ODL-SDNi application.

Components

SDNiAggregator, SDNi API, SDNiWrapper, and SDNiUI are the four components in ODL-SDNi App:

- SDNiAggregator: Connects with switch, topology, hosttracker managers of controller to get the topology and other related data.
- SDNi REST API: It is a part of controller northbound, which gives the required information by quering SDNi-Aggregator through RESTCONF.
- SDNiWrapper: This component uses the SDNi REST API and gathers the information required to be shared among controllers.
- SDNiUI:This component displays all the SDN controllers which are connected to each other.

Troubleshooting

To work with multiple controllers, change some of the configuration in config.ini file. For example change the listening port of one controller to 6653 and other controller to 6663 in /root/controller/opendaylight/distribution/opendaylight/target/distribution.opendaylight-osgipackage/opendaylight/configuration/config.ini (i.e., of.listenPort=6653).

OpenFlow related system parameters.

TCP port on which the controller is listening (default 6633) of.listenPort=6653

OF-CONFIG User Guide

Overview

OF-CONFIG defines an OpenFlow switch as an abstraction called an OpenFlow Logical Switch. The OF-CONFIG protocol enables configuration of essential artifacts of an OpenFlow Logical Switch so that an OpenFlow controller can communicate and control the OpenFlow Logical switch via the OpenFlow protocol. OF-CONFIG introduces an operating context for one or more OpenFlow data paths called an OpenFlow Capable Switch for one or more switches. An OpenFlow Capable Switch is intended to be equivalent to an actual physical or virtual network element (e.g. an Ethernet switch) which is hosting one or more OpenFlow data paths by partitioning a set of OpenFlow related resources such as ports and queues among the hosted OpenFlow data paths. The OF-CONFIG protocol enables dynamic association of the OpenFlow related resources of an OpenFlow Capable Switch with specific OpenFlow Logical Switches which are being hosted on the OpenFlow Capable Switch. OF-CONFIG does not specify or report how the partitioning of resources on an OpenFlow Capable Switch is achieved. OF-CONFIG assumes that resources such as ports and queues are partitioned amongst multiple OpenFlow Logical Switches such that each OpenFlow Logical Switch can assume full control over the resources that is assigned to it.

How to start

- start OF-CONFIG feature as below:

```
feature:install odl-of-config-all
```

Configuration on the OVS supporting OF-CONFIG

Note: OVS is not supported by OF-CONFIG temporarily because the OpenDaylight version of OF-CONFIG is 1.2 while the OVS version of OF-CONFIG is not standard.

The introduction of configuring the OVS can be referred to:

https://github.com/openvswitch/of-config.

Connection Establishment between the Capable/Logical Switch and OF-CONFIG

The OF-CONFIG protocol is based on NETCONF. So the switches supporting OF-CONFIG can also access OpenDaylight using the functions provided by NETCONF. This is the preparation step before connecting to OF-CONFIG. How to access the switch to OpenDaylight using the NETCONF can be referred to the *NETCONF Southbound User Guide* or NETCONF Southbound examples on the wiki.

Now the switches supporting OF-CONFIG and they have connected to the controller using NETCONF as described in preparation phase. OF-CONFIG can check whether the switch can support OF-CONFIG by reading the capability list in NETCONF.

The OF-CONFIG will get the information of the capable switch and logical switch via the NETCONF connection, and creates separate topologies for the capable and logical switches in the OpenDaylight Topology module.

The Connection between the capable/logical switches and OF-CONFIG is finished.

Configuration On Capable Switch

Here is an example showing how to make the configuration to modify-controller-connection on the capable switch using OF-CONFIG. Other configurations can follow the same way of the example.

- Example: modify-controller-connection

Note: this configuration can execute via the NETCONF, which can be referred to the *NETCONF Southbound User Guide* or NETCONF Southbound examples on the wiki.

OpenFlow Plugin Project User Guide

Overview and Architecture

Overview and Architecture

Overview

OpenFlow is a vendor-neutral standard communications interface defined to enable interaction between the control and forwarding layers of an SDN architecture. The OpenFlow plugin project intends to develop a plugin to support implementations of the OpenFlow specification as it develops and evolves. Specifically the project has developed a plugin aiming to support OpenFlow 1.0 and 1.3.x. It can be extended to add support for subsequent OpenFlow specifications. The plugin is based on the Model Driven Service Abstraction Layer (MD-SAL) architecture (https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL). This new OpenFlow 1.0/1.3 MD-SAL based plugin is distinct from the old OpenFlow 1.0 plugin which was based on the API driven SAL (AD-SAL) architecture.

Scope

- Southbound plugin and integration of OpenFlow 1.0/1.3.x library project
- Ongoing support and integration of the OpenFlow specification
- The plugin should be implemented in an easily extensible manner
- Protocol verification activities will be performed on supported OpenFlow specifications

Architecture and Design

Functionality

OpenFlow 1.3 Plugin will support the following functionality

- Connection Handling
- Session Management
- State Management.
- Error Handling.
- Mapping function(Infrastructure to OF structures).
- Connection establishment will be handled by OpenFlow library using opensource netty.io library.

- Message handling(Ex: Packet in).
- Event handling and propagation to upper layers.
- Plugin will support both MD-SAL and Hard SAL.
- Will be backward compatible with OF 1.0.

Activities in OF plugin module

- New OF plugin bundle will support both OF 1.0 and OF 1.3.
- Integration with OpenFlow library.
- Integration with corresponding MD-SAL infrastructure.
- Hard SAL will be supported as adapter on top of MD-SAL plugin.
- OF 1.3 and OF 1.0 plugin will be integrated as single bundle.

Design

Overall Architecture

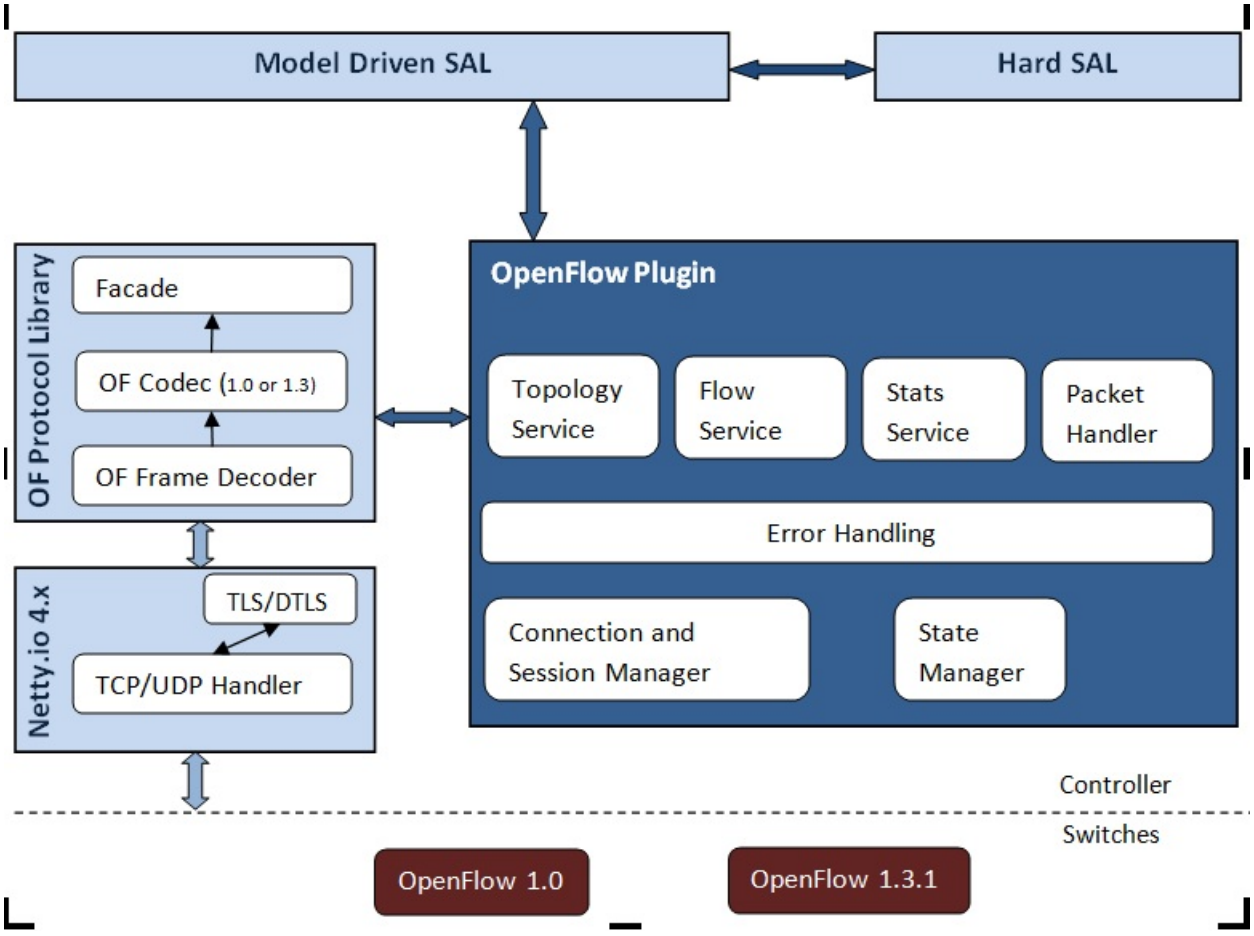


Fig. 1.80: overal architecture

Coverage

Intro

This page is to catalog the things that have been tested and confirmed to work:

Coverage

Coverage has been moved to a GoogleDoc Spreadsheet

OF 1.3 Considerations

The baseline model is a OF 1.3 model, and the coverage tables primarily deal with OF 1.3. However for OF 1.0, we have a column to indicate either N/A if it doesn't apply, or whether its been confirmed working.

OF 1.0 Considerations

OF 1.0 is being considered as a switch with: * 1 Table * 0 Groups * 0 Meters * 1 Instruction (Apply Actions) * and a limited vocabulary of matches and actions.

Tutorial / How-To

Running the controller with the new OpenFlow Plugin

How to start

There are all helium features (from features-openflowplugin) duplicated into features-openflowplugin-li. The duplicates got suffix *-li* and provide Lithium codebase functionality.

These are most used:

- odl-openflowplugin-app-lldp-speaker-li
- odl-openflowplugin-flow-services-rest-li
- odl-openflowplugin-drop-test-li

In case topology is required then the first one should be installed.

```
feature:install odl-openflowplugin-app-lldp-speaker-li
```

The Li-southbound currently provides:

- flow management
- group management
- meter management
- statistics polling

What to log

In order to see really low level messages enter these in karaf console:

```
log:set TRACE org.opendaylight.openflowplugin.openflow.md.core
log:set TRACE org.opendaylight.openflowplugin.impl
```

How enable topology

In order for topology to work (fill dataStore/operational with links) there must be LLDP responses delivered back to controller. This requires table-miss-entries. Table-miss-entry is a flow in table.id=0 with low priority, empty match and one output action = send to controller. Having this flow installed on every node will enable for gathering and exporting links between nodes into dataStore/operational. This is done if you use for example l2 switch application.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
   <barrier>false</barrier>
   <cookie>54</cookie>
   <flags>SEND_FLOW_REM</flags>
   <flow-name>FooXf54</flow-name>
   <hard-timeout>0</hard-timeout>
   <id>4242</id>
   <idle-timeout>0</idle-timeout>
   <installHw>false</installHw>
   <instructions>
       <instruction>
           <apply-actions>
               <action>
                   <output-action>
                       <max-length>65535</max-length>
                       <output-node-connector>CONTROLLER</output-node-connector>
                   </output-action>
                   <order>0</order>
               </action>
           </apply-actions>
           <order>0</order>
       </instruction>
   </instructions>
   <match/>
   <priority>0</priority>
   <strict>false</strict>
   <table_id>0</table_id>
</flow>
```

Enable RESTCONF and Controller GUI

If you want to use RESTCONF with openflowplugin project, you have to install *odl-restconf* feature to enable that. To install *odl-restconf* feature run the following command

```
karaf#>feature:install odl-restconf
```

If you want to access the Controller GUI, you have to install *odl-dlux-core* feature to enable that. Run following command to install it

```
karaf#>feature:install odl-dlux-core
```

Once you enable the feature, access the Controller GUI using following URL

```
http://<controller-ip>:8181/dlux/index.html
```

OpenFlow 1.3 Enabled Software Switches / Environment

Getting Mininet with OF 1.3

Download Mininet VM Upgraded to OF 1.3 (or the newer mininet-2.1.0 with OVS-2.0 that works with VMware Player. For using this on VirtualBox, import this to VMware Player and then export the .vmdk) or you could build one yourself Openflow Protocol Library:OpenVirtualSwitch[Instructions for setting up Mininet with OF 1.3].

Installing under VirtualBox

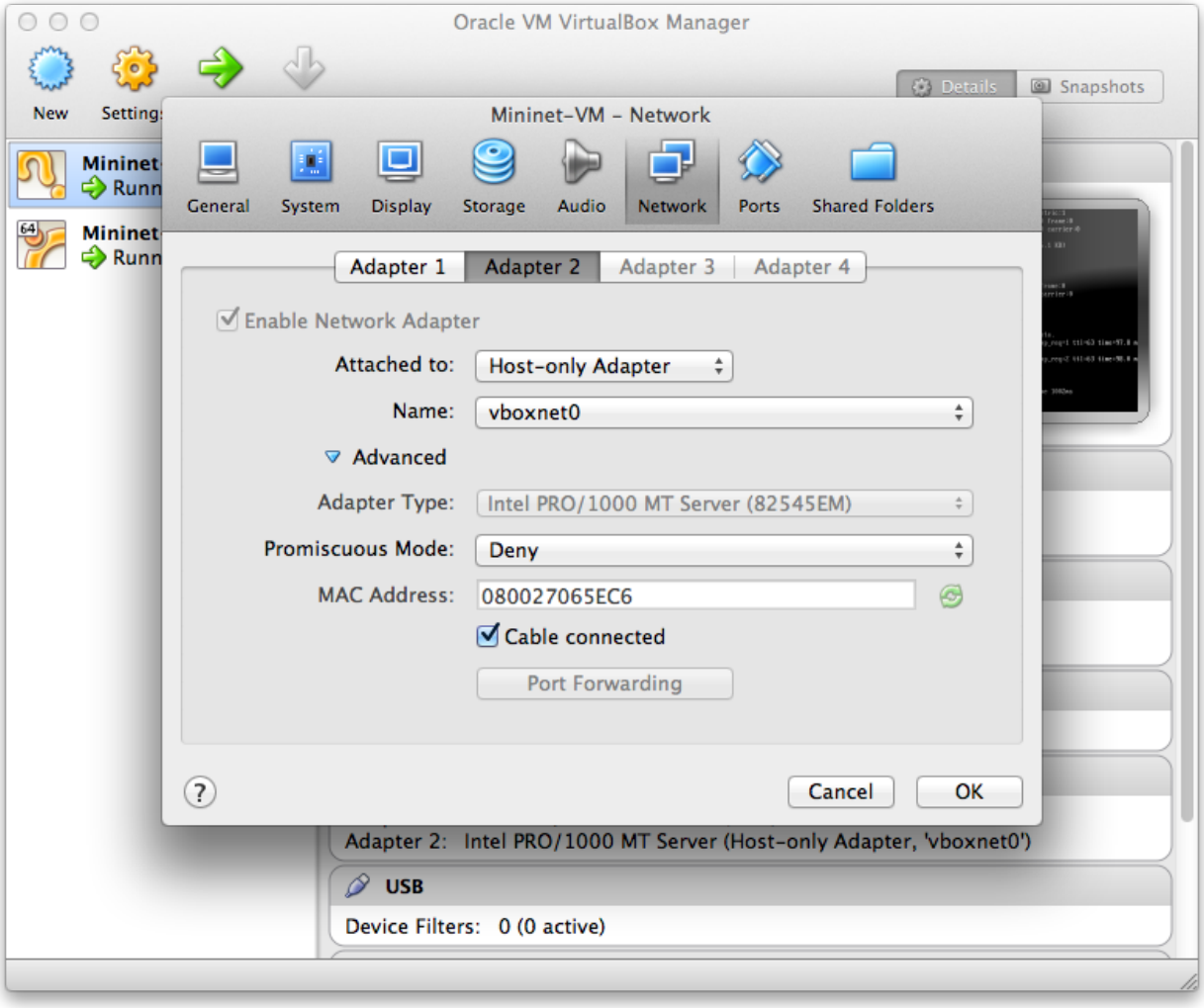


Fig. 1.81: configuring a host-only adapter

For whatever reason, at least on the Mac, NATed interfaces in VirtualBox don't actually seem to allow for connections from the host to the VM. Instead, you need to configure a host-only network and set it up. Do this by:

- Go to the VM's settings in VirtualBox then to network and add a second adapter attached to "Host-only Adapter" (see the screenshot to the right)
- Edit the /etc/network/interfaces file to configure the adapter properly by adding these two lines

```
auto eth1
iface eth1 inet dhcp
```

- Reboot the VM

At this point you should have two interfaces one which gives you NATed access to the internet and another that gives you access between your mac and the VMs. At least for me, the NATed interface gets a 10.0.2.x address and the the host-only interface gets a 192.168.56.x address.

Your simplest choice: Use Vagrant

Download Virtual Box and install it Download Vagrant and install it

```
cd openflowplugin/vagrant/mininet-2.1.0-of-1.3/
vagrant up
vagrant ssh
```

This will leave you sshed into a fully provisioned Ubuntu Trusty box with mininet-2.1.0 and OVS 2.0 patches to work with OF 1.3.

Setup CPqD Openflow 1.3 Soft Switch

Latest version of Openvswitch (v2.0.0) doesn't support all the openflow 1.3 features, e.g group multipart statistics request. Alternate options is CPqD Openflow 1.3 soft switch, It supports most of the openflow 1.3 features.

- You can setup the switch as per the instructions given on the following URL

https://github.com/CPqD/ofsoftswitch13

- Fire following command to start the switch

Start the datapath :

```
$ sudo udatapath/ofdatapath --datapath-id=<dpid> --interfaces=<if-list> ptcp:<port>
 e.g $ sudo udatapath/ofdatapath --datapath-id=000000000001 --interfaces=ethX␣
↪ptcp:6680
```

ethX should not be associated with ip address and ipv6 should be disabled on it. If you are installing the switch on your local machine, you can use following command (for Ubuntu) to create virtual interface.

```
ip link add link ethX address 00:19:d1:29:d2:58 macvlan0 type macvlan
```

ethX - Any existing interface.

Or if you are using mininet VM for installing this switch, you can simply add one more adaptor to your VM.

Start Openflow protocol agent:

```
$secchan/ofprotocol tcp:<switch-host>:<switch-port> tcp:<ctrl-host>:<ctrl-port>
 e.g $secchan/ofprotocol tcp:127.0.0.1:6680 tcp:127.0.0.1:6653
```

Commands to add entries to various tables of the switch

- Add meter

```
$utilities/dpctl tcp:<switch-host>:<switch-port> meter-mod cmd=add,meter=1␣
↪drop:rate=50
```

- Add Groups

```
$utilities/dpctl tcp:127.0.0.1:6680 group-mod cmd=add,type=all,group=1
```

```
$utilities/dpctl tcp:127.0.0.1:6680 group-mod cmd=add,type=sel,group=2 weight=10␣
↪output:1
```

- Create queue

```
$utilities/dpctl tcp:<ip>:<switch port> queue-mod <port-number> <queue-number>
↪<minimum-bandwidth>
  e.g - $utilities/dpctl tcp:127.0.0.1:6680 queue-mod 1 1 23
```

"dpctl" –help is not very intuitive, so please keep adding any new command you figured out while your experiment with the switch.

Using the built-in Wireshark

Mininet comes with pre-installed Wireshark, but for some reason it does not include the Openflow protocol dissector. You may want to get and install it in the */.wireshark/plugins/* directory.

First login to your mininet VM

```
ssh mininet@<your mininet vm ip> -X
```

The -X option in ssh will enable x-session over ssh so that the wireshark window can be shown on your host machine's display. when prompted, enter the password (mininet).

From the mininet vm shell, set the wireshark capture privileges (http://wiki.wireshark.org/CaptureSetup/CapturePrivileges):

```
sudo chgrp mininet /usr/bin/dumpcap
sudo chmod 754 /usr/bin/dumpcap
sudo setcap 'CAP_NET_RAW+eip CAP_NET_ADMIN+eip' /usr/bin/dumpcap
```

Finally, start wireshark:

```
wireshark
```

The wireshark window should show up.

To see only Openflow packets, you may want to apply the following filter in the Filter window:

```
tcp.port == 6633 and tcp.flags.push == 1
```

Start the capture on *any* port.

Running Mininet with OF 1.3

From within the Mininet VM, run:

```
sudo mn --topo single,3  --controller 'remote,ip=<your controller ip>,port=6653' --
↪switch ovsk,protocols=OpenFlow13
```

End to End Inventory

Introduction

The purpose of this page is to walk you through how to see the Inventory Manager working end to end with the openflowplugin using OpenFlow 1.3.

Basically, you will learn how to:

1. Run the Base/Virtualization/Service provider Edition with the new openflowplugin: OpenDaylight_OpenFlow_Plugin::Running_controller_with_the_new_OF_plugin[Running the controller with the new OpenFlow Plugin]
2. Start mininet to use OF 1.3: OpenDaylight_OpenFlow_Plugin::Test_Environment[OpenFlow 1.3 Enabled Software Switches / Environment]
3. Use RESTCONF to see the nodes appear in inventory.

Restconf for Inventory

The REST url for listing all the nodes is:

```
http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/
```

You will need to set the Accept header:

```
Accept: application/xml
```

You will also need to use HTTP Basic Auth with username: admin password: admin.

Alternately, if you have a node's id you can address it as

```
http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/node/<id>
```

for example

```
http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/node/
↪openflow:1
```

How to hit RestConf with Postman

Install Postman for Chrome

In the chrome browser bar enter

```
chrome://apps/
```

And click on Postman.

Enter the URL. Click on the Headers button on the far right. Enter the Accept: header. Click on the Basic Auth Tab at the top and setup the username and password. Send.

Known Bug

If you have not had any switches come up, and though no children for http://localhost:8080/restconf/datastore/opendaylight-inventory:nodes/ and exception will be thrown. I'm pretty sure I know how to fix this bug, just need to get to it :)

End to End Flows

Instructions

Learn End to End for Inventory

See *End to End Inventory*

Check inventory

- Run mininet with support for OF 1.3 as described in *End to End Inventory*
- Make sure you see the openflow:1 node come up as described in *End to End Inventory*

Flow Strategy

Current way to flush a flow to switch looks like this:

1. Create MD-SAL modeled flow and commit it into dataStore using two phase commit MD-SAL FAQ
2. FRM gets notified and invokes corresponding rpc (addFlow) on particular service provider (if suitable provider for given node registered)
3. The provider (plugin in this case) transforms MD-SAL modeled flow into OF-API modeled flow
4. OF-API modeled flow is then flushed into OFLibrary
5. OFLibrary encodes flow into particular version of wire protocol and sends it to particular switch
6. Check on mininet side if flow is set

Push your flow

- With PostMan:
  - Set headers:
    - Content-Type: application/xml
    - Accept: application/xml
    - Authentication
  - Use URL: "http://<controller IP>:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1"
  - PUT
  - Use Body:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <priority>2</priority>
    <flow-name>Foo</flow-name>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
        <ipv4-destination>10.0.10.2/24</ipv4-destination>
    </match>
    <id>1</id>
    <table_id>0</table_id>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                   <order>0</order>
                   <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
</flow>
```

***Note**: If you want to try a different flow id or a different table, make sure the URL and the body stay in sync. For example, if you wanted to try: table 2 flow 20 you'd change the URL to:

"http://<controller IP>:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/2/flow/20"

but you would also need to update the 20 and 2 in the body of the XML.

Other caveat, we have a known bug with updates, so please only write to a given flow id and table id on a given node once at this time until we resolve it. Or you can use the DELETE method with the same URL in PostMan to delete the flow information on switch and controller cache.

Check for your flow on the switch

- See your flow on your mininet:

```
mininet@mininet-vm:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=7.325s, table=0, n_packets=0, n_bytes=0, idle_timeout=300, hard_
↪timeout=600, send_flow_rem priority=2,ip,nw_dst=10.0.10.0/24 actions=dec_ttl
```

If you want to see the above information from the mininet prompt - use "sh" instead of "sudo" i.e. use "sh ovs-ofctl -O OpenFlow13 dump-flows s1".

Check for your flow in the controller config via RESTCONF

- See your configured flow in POSTMAN with
  - URL http://<controller IP>:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/table/0/
  - GET

- You no longer need to set Accept header

Return Response:


```
{
  "flow-node-inventory:table": [
    {
      "flow-node-inventory:id": 0,
      "flow-node-inventory:flow": [
        {
          "flow-node-inventory:priority": 1,
          "flow-node-inventory:id": "10b1a23c-5299-4f7b-83d6-563bab472754",
          "flow-node-inventory:table_id": 0,
          "flow-node-inventory:hard-timeout": 0,
          "flow-node-inventory:idle-timeout": 0,
          "flow-node-inventory:instructions": {
            "flow-node-inventory:instruction": [
              {
                "flow-node-inventory:apply-actions": {
                  "flow-node-inventory:action": [
                    {
                      "flow-node-inventory:output-action": {
                        "flow-node-inventory:output-node-connector": "openflow:1:1"
                      },
                      "flow-node-inventory:order": 0
                    }
                  ]
                },
                "flow-node-inventory:order": 0
              }
            ]
          },
          "flow-node-inventory:match": {
            "flow-node-inventory:ethernet-match": {
              "flow-node-inventory:ethernet-type": {
                "flow-node-inventory:type": 2048
              }
            },
            "flow-node-inventory:ipv4-destination": "10.0.0.2"
          },
          "flow-node-inventory:cookie": 0
        },
        {
          "flow-node-inventory:priority": 1,
          "flow-node-inventory:id": "020bf359-1299-4da6-b4f7-368bd83b5841",
          "flow-node-inventory:table_id": 0,
          "flow-node-inventory:hard-timeout": 0,
          "flow-node-inventory:idle-timeout": 0,
          "flow-node-inventory:instructions": {
            "flow-node-inventory:instruction": [
              {
                "flow-node-inventory:apply-actions": {
                  "flow-node-inventory:action": [
                    {
                      "flow-node-inventory:output-action": {
                        "flow-node-inventory:output-node-connector": "openflow:1:1"
                      },
                      "flow-node-inventory:order": 0
                    }
```

```
                ]
              },
              "flow-node-inventory:order": 0
            }
          ]
        },
        "flow-node-inventory:match": {
          "flow-node-inventory:ethernet-match": {
            "flow-node-inventory:ethernet-type": {
              "flow-node-inventory:type": 2048
            }
          },
          "flow-node-inventory:ipv4-destination": "10.0.0.1"
        },
        "flow-node-inventory:cookie": 0
      },
      {
        "flow-node-inventory:priority": 1,
        "flow-node-inventory:id": "42172bfc-9142-4a92-9e90-ee62529b1e85",
        "flow-node-inventory:table_id": 0,
        "flow-node-inventory:hard-timeout": 0,
        "flow-node-inventory:idle-timeout": 0,
        "flow-node-inventory:instructions": {
          "flow-node-inventory:instruction": [
            {
              "flow-node-inventory:apply-actions": {
                "flow-node-inventory:action": [
                  {
                    "flow-node-inventory:output-action": {
                      "flow-node-inventory:output-node-connector": "openflow:1:1"
                    },
                    "flow-node-inventory:order": 0
                  }
                ]
              },
              "flow-node-inventory:order": 0
            }
          ]
        },
        "flow-node-inventory:match": {
          "flow-node-inventory:ethernet-match": {
            "flow-node-inventory:ethernet-type": {
              "flow-node-inventory:type": 2048
            }
          },
          "flow-node-inventory:ipv4-destination": "10.0.0.3"
        },
        "flow-node-inventory:cookie": 0
      },
      {
        "flow-node-inventory:priority": 1,
        "flow-node-inventory:id": "99bf566e-89f3-4c6f-ae9e-e26012ceb1e4",
        "flow-node-inventory:table_id": 0,
        "flow-node-inventory:hard-timeout": 0,
        "flow-node-inventory:idle-timeout": 0,
        "flow-node-inventory:instructions": {
          "flow-node-inventory:instruction": [
            {
```

```
            "flow-node-inventory:apply-actions": {
              "flow-node-inventory:action": [
                {
                  "flow-node-inventory:output-action": {
                    "flow-node-inventory:output-node-connector": "openflow:1:1"
                  },
                  "flow-node-inventory:order": 0
                }
              ]
            },
            "flow-node-inventory:order": 0
          }
        ]
      },
      "flow-node-inventory:match": {
        "flow-node-inventory:ethernet-match": {
          "flow-node-inventory:ethernet-type": {
            "flow-node-inventory:type": 2048
          }
        },
        "flow-node-inventory:ipv4-destination": "10.0.0.4"
      },
      "flow-node-inventory:cookie": 0
    },
    {
      "flow-node-inventory:priority": 1,
      "flow-node-inventory:id": "019dcc2e-5b4f-44f0-90cc-de490294b862",
      "flow-node-inventory:table_id": 0,
      "flow-node-inventory:hard-timeout": 0,
      "flow-node-inventory:idle-timeout": 0,
      "flow-node-inventory:instructions": {
        "flow-node-inventory:instruction": [
          {
            "flow-node-inventory:apply-actions": {
              "flow-node-inventory:action": [
                {
                  "flow-node-inventory:output-action": {
                    "flow-node-inventory:output-node-connector": "openflow:1:2"
                  },
                  "flow-node-inventory:order": 0
                }
              ]
            },
            "flow-node-inventory:order": 0
          }
        ]
      },
      "flow-node-inventory:match": {
        "flow-node-inventory:ethernet-match": {
          "flow-node-inventory:ethernet-type": {
            "flow-node-inventory:type": 2048
          }
        },
        "flow-node-inventory:ipv4-destination": "10.0.0.5"
      },
      "flow-node-inventory:cookie": 0
    },
    {
```

```
        "flow-node-inventory:priority": 1,
        "flow-node-inventory:id": "968cf81e-3f16-42f1-8b16-d01ff719c63c",
        "flow-node-inventory:table_id": 0,
        "flow-node-inventory:hard-timeout": 0,
        "flow-node-inventory:idle-timeout": 0,
        "flow-node-inventory:instructions": {
          "flow-node-inventory:instruction": [
            {
              "flow-node-inventory:apply-actions": {
                "flow-node-inventory:action": [
                  {
                    "flow-node-inventory:output-action": {
                      "flow-node-inventory:output-node-connector": "openflow:1:2"
                    },
                    "flow-node-inventory:order": 0
                  }
                ]
              },
              "flow-node-inventory:order": 0
            }
          ]
        },
        "flow-node-inventory:match": {
          "flow-node-inventory:ethernet-match": {
            "flow-node-inventory:ethernet-type": {
              "flow-node-inventory:type": 2048
            }
          },
          "flow-node-inventory:ipv4-destination": "10.0.0.8"
        },
        "flow-node-inventory:cookie": 0
      },
      {
        "flow-node-inventory:priority": 1,
        "flow-node-inventory:id": "1c14ea3c-9dcc-4434-b566-7e99033ea252",
        "flow-node-inventory:table_id": 0,
        "flow-node-inventory:hard-timeout": 0,
        "flow-node-inventory:idle-timeout": 0,
        "flow-node-inventory:instructions": {
          "flow-node-inventory:instruction": [
            {
              "flow-node-inventory:apply-actions": {
                "flow-node-inventory:action": [
                  {
                    "flow-node-inventory:output-action": {
                      "flow-node-inventory:output-node-connector": "openflow:1:2"
                    },
                    "flow-node-inventory:order": 0
                  }
                ]
              },
              "flow-node-inventory:order": 0
            }
          ]
        },
        "flow-node-inventory:match": {
          "flow-node-inventory:ethernet-match": {
            "flow-node-inventory:ethernet-type": {
```

```
                "flow-node-inventory:type": 2048
              }
            },
            "flow-node-inventory:ipv4-destination": "10.0.0.6"
          },
          "flow-node-inventory:cookie": 0
        },
        {
          "flow-node-inventory:priority": 1,
          "flow-node-inventory:id": "ed9deeb2-be8f-4b84-bcd8-9d12049383d6",
          "flow-node-inventory:table_id": 0,
          "flow-node-inventory:hard-timeout": 0,
          "flow-node-inventory:idle-timeout": 0,
          "flow-node-inventory:instructions": {
            "flow-node-inventory:instruction": [
              {
                "flow-node-inventory:apply-actions": {
                  "flow-node-inventory:action": [
                    {
                      "flow-node-inventory:output-action": {
                        "flow-node-inventory:output-node-connector": "openflow:1:2"
                      },
                      "flow-node-inventory:order": 0
                    }
                  ]
                },
                "flow-node-inventory:order": 0
              }
            ]
          },
          "flow-node-inventory:match": {
            "flow-node-inventory:ethernet-match": {
              "flow-node-inventory:ethernet-type": {
                "flow-node-inventory:type": 2048
              }
            },
            "flow-node-inventory:ipv4-destination": "10.0.0.7"
          },
          "flow-node-inventory:cookie": 0
        }
      ]
    }
  ]
}
```


Look for your flow stats in the controller operational data via

RESTCONF

- See your operational flow stats in POSTMAN with
  - URL "http://<controller IP>:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/table/0/"
  - GET

Return Response:

```
{
  "flow-node-inventory:table": [
    {
      "flow-node-inventory:id": 0,
      "flow-node-inventory:flow": [
        {
          "flow-node-inventory:id": "10b1a23c-5299-4f7b-83d6-563bab472754",
          "opendaylight-flow-statistics:flow-statistics": {
            "opendaylight-flow-statistics:cookie": 0,
            "opendaylight-flow-statistics:duration": {
              "opendaylight-flow-statistics:nanosecond": 886000000,
              "opendaylight-flow-statistics:second": 2707
            },
            "opendaylight-flow-statistics:hard-timeout": 0,
            "opendaylight-flow-statistics:byte-count": 784,
            "opendaylight-flow-statistics:match": {
              "opendaylight-flow-statistics:ethernet-match": {
                "opendaylight-flow-statistics:ethernet-type": {
                  "opendaylight-flow-statistics:type": 2048
                }
              },
              "opendaylight-flow-statistics:ipv4-destination": "10.0.0.2/32"
            },
            "opendaylight-flow-statistics:priority": 1,
            "opendaylight-flow-statistics:packet-count": 8,
            "opendaylight-flow-statistics:table_id": 0,
            "opendaylight-flow-statistics:idle-timeout": 0,
            "opendaylight-flow-statistics:instructions": {
              "opendaylight-flow-statistics:instruction": [
                {
                  "opendaylight-flow-statistics:order": 0,
                  "opendaylight-flow-statistics:apply-actions": {
                    "opendaylight-flow-statistics:action": [
                      {
                        "opendaylight-flow-statistics:order": 0,
                        "opendaylight-flow-statistics:output-action": {
                          "opendaylight-flow-statistics:output-node-connector": "1",
                          "opendaylight-flow-statistics:max-length": 0
                        }
                      }
                    ]
                  }
                }
              ]
            }
          }
        },
        {
          "flow-node-inventory:id": "020bf359-1299-4da6-b4f7-368bd83b5841",
          "opendaylight-flow-statistics:flow-statistics": {
            "opendaylight-flow-statistics:cookie": 0,
            "opendaylight-flow-statistics:duration": {
              "opendaylight-flow-statistics:nanosecond": 826000000,
              "opendaylight-flow-statistics:second": 2711
            },
            "opendaylight-flow-statistics:hard-timeout": 0,
            "opendaylight-flow-statistics:byte-count": 1568,
            "opendaylight-flow-statistics:match": {
```

```
            "opendaylight-flow-statistics:ethernet-match": {
              "opendaylight-flow-statistics:ethernet-type": {
                "opendaylight-flow-statistics:type": 2048
              }
            },
            "opendaylight-flow-statistics:ipv4-destination": "10.0.0.1/32"
          },
          "opendaylight-flow-statistics:priority": 1,
          "opendaylight-flow-statistics:packet-count": 16,
          "opendaylight-flow-statistics:table_id": 0,
          "opendaylight-flow-statistics:idle-timeout": 0,
          "opendaylight-flow-statistics:instructions": {
            "opendaylight-flow-statistics:instruction": [
              {
                "opendaylight-flow-statistics:order": 0,
                "opendaylight-flow-statistics:apply-actions": {
                  "opendaylight-flow-statistics:action": [
                    {
                      "opendaylight-flow-statistics:order": 0,
                      "opendaylight-flow-statistics:output-action": {
                        "opendaylight-flow-statistics:output-node-connector": "1",
                        "opendaylight-flow-statistics:max-length": 0
                      }
                    }
                  ]
                }
              }
            ]
          }
        }
      },
      {
        "flow-node-inventory:id": "42172bfc-9142-4a92-9e90-ee62529b1e85",
        "opendaylight-flow-statistics:flow-statistics": {
          "opendaylight-flow-statistics:cookie": 0,
          "opendaylight-flow-statistics:duration": {
            "opendaylight-flow-statistics:nanosecond": 548000000,
            "opendaylight-flow-statistics:second": 2708
          },
          "opendaylight-flow-statistics:hard-timeout": 0,
          "opendaylight-flow-statistics:byte-count": 784,
          "opendaylight-flow-statistics:match": {
            "opendaylight-flow-statistics:ethernet-match": {
              "opendaylight-flow-statistics:ethernet-type": {
                "opendaylight-flow-statistics:type": 2048
              }
            },
            "opendaylight-flow-statistics:ipv4-destination": "10.0.0.3/32"
          },
          "opendaylight-flow-statistics:priority": 1,
          "opendaylight-flow-statistics:packet-count": 8,
          "opendaylight-flow-statistics:table_id": 0,
          "opendaylight-flow-statistics:idle-timeout": 0,
          "opendaylight-flow-statistics:instructions": {
            "opendaylight-flow-statistics:instruction": [
              {
                "opendaylight-flow-statistics:order": 0,
                "opendaylight-flow-statistics:apply-actions": {
```

```
                    "opendaylight-flow-statistics:action": [
                      {
                        "opendaylight-flow-statistics:order": 0,
                        "opendaylight-flow-statistics:output-action": {
                          "opendaylight-flow-statistics:output-node-connector": "1",
                          "opendaylight-flow-statistics:max-length": 0
                        }
                      }
                    ]
                  }
                }
              ]
            }
          }
        },
        {
          "flow-node-inventory:id": "99bf566e-89f3-4c6f-ae9e-e26012ceb1e4",
          "opendaylight-flow-statistics:flow-statistics": {
            "opendaylight-flow-statistics:cookie": 0,
            "opendaylight-flow-statistics:duration": {
              "opendaylight-flow-statistics:nanosecond": 296000000,
              "opendaylight-flow-statistics:second": 2710
            },
            "opendaylight-flow-statistics:hard-timeout": 0,
            "opendaylight-flow-statistics:byte-count": 1274,
            "opendaylight-flow-statistics:match": {
              "opendaylight-flow-statistics:ethernet-match": {
                "opendaylight-flow-statistics:ethernet-type": {
                  "opendaylight-flow-statistics:type": 2048
                }
              },
              "opendaylight-flow-statistics:ipv4-destination": "10.0.0.4/32"
            },
            "opendaylight-flow-statistics:priority": 1,
            "opendaylight-flow-statistics:packet-count": 13,
            "opendaylight-flow-statistics:table_id": 0,
            "opendaylight-flow-statistics:idle-timeout": 0,
            "opendaylight-flow-statistics:instructions": {
              "opendaylight-flow-statistics:instruction": [
                {
                  "opendaylight-flow-statistics:order": 0,
                  "opendaylight-flow-statistics:apply-actions": {
                    "opendaylight-flow-statistics:action": [
                      {
                        "opendaylight-flow-statistics:order": 0,
                        "opendaylight-flow-statistics:output-action": {
                          "opendaylight-flow-statistics:output-node-connector": "1",
                          "opendaylight-flow-statistics:max-length": 0
                        }
                      }
                    ]
                  }
                }
              ]
            }
          }
        },
        {
```

```
        "flow-node-inventory:id": "019dcc2e-5b4f-44f0-90cc-de490294b862",
        "opendaylight-flow-statistics:flow-statistics": {
          "opendaylight-flow-statistics:cookie": 0,
          "opendaylight-flow-statistics:duration": {
            "opendaylight-flow-statistics:nanosecond": 392000000,
            "opendaylight-flow-statistics:second": 2711
          },
          "opendaylight-flow-statistics:hard-timeout": 0,
          "opendaylight-flow-statistics:byte-count": 1470,
          "opendaylight-flow-statistics:match": {
            "opendaylight-flow-statistics:ethernet-match": {
              "opendaylight-flow-statistics:ethernet-type": {
                "opendaylight-flow-statistics:type": 2048
              }
            },
            "opendaylight-flow-statistics:ipv4-destination": "10.0.0.5/32"
          },
          "opendaylight-flow-statistics:priority": 1,
          "opendaylight-flow-statistics:packet-count": 15,
          "opendaylight-flow-statistics:table_id": 0,
          "opendaylight-flow-statistics:idle-timeout": 0,
          "opendaylight-flow-statistics:instructions": {
            "opendaylight-flow-statistics:instruction": [
              {
                "opendaylight-flow-statistics:order": 0,
                "opendaylight-flow-statistics:apply-actions": {
                  "opendaylight-flow-statistics:action": [
                    {
                      "opendaylight-flow-statistics:order": 0,
                      "opendaylight-flow-statistics:output-action": {
                        "opendaylight-flow-statistics:output-node-connector": "2",
                        "opendaylight-flow-statistics:max-length": 0
                      }
                    }
                  ]
                }
              }
            ]
          }
        }
      },
      {
        "flow-node-inventory:id": "968cf81e-3f16-42f1-8b16-d01ff719c63c",
        "opendaylight-flow-statistics:flow-statistics": {
          "opendaylight-flow-statistics:cookie": 0,
          "opendaylight-flow-statistics:duration": {
            "opendaylight-flow-statistics:nanosecond": 344000000,
            "opendaylight-flow-statistics:second": 2707
          },
          "opendaylight-flow-statistics:hard-timeout": 0,
          "opendaylight-flow-statistics:byte-count": 784,
          "opendaylight-flow-statistics:match": {
            "opendaylight-flow-statistics:ethernet-match": {
              "opendaylight-flow-statistics:ethernet-type": {
                "opendaylight-flow-statistics:type": 2048
              }
            },
            "opendaylight-flow-statistics:ipv4-destination": "10.0.0.8/32"
```

```
            },
            "opendaylight-flow-statistics:priority": 1,
            "opendaylight-flow-statistics:packet-count": 8,
            "opendaylight-flow-statistics:table_id": 0,
            "opendaylight-flow-statistics:idle-timeout": 0,
            "opendaylight-flow-statistics:instructions": {
              "opendaylight-flow-statistics:instruction": [
                {
                  "opendaylight-flow-statistics:order": 0,
                  "opendaylight-flow-statistics:apply-actions": {
                    "opendaylight-flow-statistics:action": [
                      {
                        "opendaylight-flow-statistics:order": 0,
                        "opendaylight-flow-statistics:output-action": {
                          "opendaylight-flow-statistics:output-node-connector": "2",
                          "opendaylight-flow-statistics:max-length": 0
                        }
                      }
                    ]
                  }
                }
              ]
            }
          }
        },
        {
          "flow-node-inventory:id": "ed9deeb2-be8f-4b84-bcd8-9d12049383d6",
          "opendaylight-flow-statistics:flow-statistics": {
            "opendaylight-flow-statistics:cookie": 0,
            "opendaylight-flow-statistics:duration": {
              "opendaylight-flow-statistics:nanosecond": 577000000,
              "opendaylight-flow-statistics:second": 2706
            },
            "opendaylight-flow-statistics:hard-timeout": 0,
            "opendaylight-flow-statistics:byte-count": 784,
            "opendaylight-flow-statistics:match": {
              "opendaylight-flow-statistics:ethernet-match": {
                "opendaylight-flow-statistics:ethernet-type": {
                  "opendaylight-flow-statistics:type": 2048
                }
              },
              "opendaylight-flow-statistics:ipv4-destination": "10.0.0.7/32"
            },
            "opendaylight-flow-statistics:priority": 1,
            "opendaylight-flow-statistics:packet-count": 8,
            "opendaylight-flow-statistics:table_id": 0,
            "opendaylight-flow-statistics:idle-timeout": 0,
            "opendaylight-flow-statistics:instructions": {
              "opendaylight-flow-statistics:instruction": [
                {
                  "opendaylight-flow-statistics:order": 0,
                  "opendaylight-flow-statistics:apply-actions": {
                    "opendaylight-flow-statistics:action": [
                      {
                        "opendaylight-flow-statistics:order": 0,
                        "opendaylight-flow-statistics:output-action": {
                          "opendaylight-flow-statistics:output-node-connector": "2",
                          "opendaylight-flow-statistics:max-length": 0
```

```
                        }
                      }
                    ]
                  }
                }
              ]
            }
          }
        },
        {
          "flow-node-inventory:id": "1c14ea3c-9dcc-4434-b566-7e99033ea252",
          "opendaylight-flow-statistics:flow-statistics": {
            "opendaylight-flow-statistics:cookie": 0,
            "opendaylight-flow-statistics:duration": {
              "opendaylight-flow-statistics:nanosecond": 659000000,
              "opendaylight-flow-statistics:second": 2705
            },
            "opendaylight-flow-statistics:hard-timeout": 0,
            "opendaylight-flow-statistics:byte-count": 784,
            "opendaylight-flow-statistics:match": {
              "opendaylight-flow-statistics:ethernet-match": {
                "opendaylight-flow-statistics:ethernet-type": {
                  "opendaylight-flow-statistics:type": 2048
                }
              },
              "opendaylight-flow-statistics:ipv4-destination": "10.0.0.6/32"
            },
            "opendaylight-flow-statistics:priority": 1,
            "opendaylight-flow-statistics:packet-count": 8,
            "opendaylight-flow-statistics:table_id": 0,
            "opendaylight-flow-statistics:idle-timeout": 0,
            "opendaylight-flow-statistics:instructions": {
              "opendaylight-flow-statistics:instruction": [
                {
                  "opendaylight-flow-statistics:order": 0,
                  "opendaylight-flow-statistics:apply-actions": {
                    "opendaylight-flow-statistics:action": [
                      {
                        "opendaylight-flow-statistics:order": 0,
                        "opendaylight-flow-statistics:output-action": {
                          "opendaylight-flow-statistics:output-node-connector": "2",
                          "opendaylight-flow-statistics:max-length": 0
                        }
                      }
                    ]
                  }
                }
              ]
            }
          }
        }
      ],
      "opendaylight-flow-table-statistics:flow-table-statistics": {
        "opendaylight-flow-table-statistics:active-flows": 8,
        "opendaylight-flow-table-statistics:packets-matched": 97683,
        "opendaylight-flow-table-statistics:packets-looked-up": 101772
      }
    }
```

```
  ]
}
```

Discovering and testing new Flow Types

Currently, the openflowplugin has a test-provider that allows you to push various flows through the system from the OSGI command line. Once those flows have been pushed through, you can see them as examples and then use them to see in the config what a particular flow example looks like.

Using addMDFlow

From the

```
cd openflowplugin/distribution/base/target/distributions-openflowplugin-base-0.0.1-
↪SNAPSHOT-osgipackage/opendaylight
./run.sh
```

Point your mininet at the controller as described above.

once you can see your node (probably openflow:1 if you've been following along) in the inventory, at the OSGI command line try running:

```
addMDFlow openflow:1 f#
```

Where # is a number between 1 and 80. This will create one of 80 possible flows. You can go confirm they were created on the switch.

Once you've done that, use

- GET
- Accept: application/xml
- URL: "http://192.168.195.157:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/2/"

To see a full listing of the flows in table 2 (where they will be put). If you want to see a particular flow, look at

- URL: "http://192.168.195.157:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/2/flow/#"

Where # is 123 + the f# you used. So for example, for f22, your url would be

- URL: "http://192.168.195.157:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/2/flow/145"

Note: You may have to trim out some of the sections like that contain bitfields and binary types that are not correctly modeled.

Note: Before attempting to PUT a flow you have created via addMDFlow, please change its URL and body to, for example, use table 1 instead of table 2 or another Flow Id, so you don't collide.

Note: There are several test command providers and the one handling flows is **OpenflowpluginTestCommandProvider**. Methods, which can be use as **commands in OSGI-console** have prefix _.

Example Flows

Examples for XML for various flow matches, instructions & actions can be found in following section *here*.

End to End Topology

Introduction

The purpose of this page is to walk you through how to see the Topology Manager working end to end with the openflowplugin using OpenFlow 1.3.

Basically, you will learn how to:

1. Run the Base/Virtualization/Service provider Edition with the new openflowplugin: *Running the controller with the new OpenFlow Plugin*
2. Start mininet to use OF 1.3: OpenFlow 1.3 Enabled Software Switches / Environment
3. Use RESTCONF to see the topology information.

Restconf for Topology

The REST url for listing all the nodes is:

```
http://localhost:8080/restconf/operational/network-topology:network-topology/
```

You will need to set the Accept header:

```
Accept: application/xml
```

You will also need to use HTTP Basic Auth with username: admin password: admin.

Alternately, if you have a node's id you can address it as

```
http://localhost:8080/restconf/operational/network-topology:network-topology/topology/
↪<id>
```

for example

```
http://localhost:8080/restconf/operational/network-topology:network-topology/topology/
↪flow:1/
```

How to hit RestConf with Postman

Install postman for Chrome

In the chrome browser bar enter

```
chrome://apps/
```

And click on Postman.

Enter the URL. Click on the Headers button on the far right. Enter the Accept: header. Click on the Basic Auth Tab at the top and setup the username and password. Send.

End to End Groups

NOTE

Groups are NOT SUPPORTED in current (2.0.0) version of openvswitch. See

- http://openvswitch.org/releases/NEWS-2.0.0
- http://comments.gmane.org/gmane.linux.network.openvswitch.general/3251

For testing group feature please use for example CPQD virtual switch in the *End to End Inventory* section.

Instructions

Learn End to End for Inventory

End to End Inventory

Check inventory

Run CPqD with support for OF 1.3 as described in *End to End Inventory*

Make sure you see the openflow:1 node come up as described in *End to End Inventory*

Group Strategy

Current way to flush a group to switch looks like this:

1. create MD-SAL modeled group and commit it into dataStore using two phase commit
2. FRM gets notified and invokes corresponding rpc (addGroup) on particular service provider (if suitable provider for given node registered)
3. the provider (plugin in this case) transforms MD-SAL modeled group into OF-API modeled group
4. OF-API modeled group is then flushed into OFLibrary
5. OFLibrary encodes group into particular version of wire protocol and sends it to particular switch
6. check on CPqD if group is installed

Push your Group

- With PostMan:
  - Set
    - Content-Type: application/xml
    - Accept: application/xml
  - Use URL: "http://<ip-address>:8080/restconf/config/opendaylight-inventory:nodes/node/openflow:1/group/1"
  - PUT
  - Use Body:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<group xmlns="urn:opendaylight:flow:inventory">
    <group-type>group-all</group-type>
    <buckets>
        <bucket>
            <action>
                <pop-vlan-action/>
                <order>0</order>
            </action>
            <bucket-id>12</bucket-id>
            <watch_group>14</watch_group>
            <watch_port>1234</watch_port>
        </bucket>
        <bucket>
            <action>
                <set-field>
                    <ipv4-source>100.1.1.1</ipv4-source>
                </set-field>
                <order>0</order>
            </action>
            <action>
                <set-field>
                    <ipv4-destination>200.71.9.5210</ipv4-destination>
                </set-field>
                <order>1</order>
            </action>
            <bucket-id>13</bucket-id>
            <watch_group>14</watch_group>
            <watch_port>1234</watch_port>
        </bucket>
    </buckets>
    <barrier>false</barrier>
    <group-name>Foo</group-name>
    <group-id>1</group-id>
</group>
```

Note: If you want to try a different group id, make sure the URL and the body stay in sync. For example, if you wanted to try: group-id 20 you'd change the URL to "http://<ip-address>:8080/restconf/config/opendaylight-inventory:nodes/node/openflow:1/group/20" but you would also need to update the <group-id>20</group-id> in the body to match.

Note: <ip-address> :Provide the IP Address of the machine on which the controller is running.

Check for your group on the switch

- See your group on your cpqd switch:

```
COMMAND: sudo dpctl tcp:127.0.0.1:6000 stats-group

SENDING:
stat_req{type="grp", flags="0x0", group="all"}
```

```
RECEIVED:
stat_repl{type="grp", flags="0x0", stats=[
{group="1", ref_cnt="0", pkt_cnt="0", byte_cnt="0", cntrs=[{pkt_cnt="0", byte_cnt="0"}
↪, {pkt_cnt="0", byte_cnt="0"}]}]}
```

Check for your group in the controller config via RESTCONF

- See your configured group in POSTMAN with
  - URL http://<ip-address>:8080/restconf/config/opendaylight-inventory:nodes/node/openflow:1/group/1
  - GET
  - You should no longer need to set Accept
  - Note: <ip-address> :Provide the IP Address of the machine on which the controller is running.

Look for your group stats in the controller operational data via RESTCONF

- See your operational group stats in POSTMAN with
  - URL http://<ip-address>:8080/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/group/1
  - GET
  - Note: <ip-address> :Provide the IP Address of the machine on which the controller is running.

Discovering and testing Group Types

Currently, the openflowplugin has a test-provider that allows you to push various groups through the system from the OSGI command line. Once those groups have been pushed through, you can see them as examples and then use them to see in the config what a particular group example looks like.

Using addGroup

From the

```
cd openflowplugin/distribution/base/target/distributions-openflowplugin-base-0.0.1-
↪SNAPSHOT-osgipackage/opendaylight
./run.sh
```

Point your CPqD at the controller as described above.

once you can see your node (probably openflow:1 if you've been following along) in the inventory, at the OSGI command line try running:

```
addGroup openflow:1
```

This will install a group in the switch. You can check whether the group is installed or not.

Once you've done that, use

- GET
- Accept: application/xml

- URL: "http://<ip-address>:8080/restconf/config/opendaylight-inventory:nodes/node/openflow:1/group/1"
  - Note: <ip-address> :Provide the IP Address of the machine on which the controller is running.

Note: Before attempting to PUT a group you have created via addGroup, please change its URL and body to, for example, use group 1 instead of group 2 or another Group Id, so that they don't collide.

Note: There are several test command providers and the one handling groups is OpenflowpluginGroupTestCommandProvider. Methods, which can be use as commands in OSGI-console have prefix _.

Example Group

Examples for XML for various Group Types can be found in the test-scripts bundle of the plugin code with names g1.xml, g2.xml and g3.xml.

End to End Meters

Instructions

Learn End to End for Inventory

- *End to End Inventory*

Check inventory

- Run mininet with support for OF 1.3 as described in *End to End Inventory*
- Make sure you see the openflow:1 node come up as described in *End to End Inventory*

Meter Strategy

Current way to flush a meter to switch looks like this:

1. create MD-SAL modeled flow and commit it into dataStore using two phase commit
2. FRM gets notified and invokes corresponding rpc (addMeter) on particular service provider (if suitable provider for given node registered)
3. the provider (plugin in this case) transforms MD-SAL modeled meter into OF-API modeled meter
4. OF-API modeled meter is then flushed into OFLibrary
5. OFLibrary encodes meter into particular version of wire protocol and sends it to particular switch
6. check on mininet side if meter is installed

Push your Meter

- Using PostMan:
  - Set Request Headers
    - Content-Type: application/xml
    - Accept: application/xml
  - Use URL: "http://:8080/restconf/config/opendaylight-inventory:nodes/node/openflow:1/meter/1"
  - Method:PUT
  - Request Body:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<meter xmlns="urn:opendaylight:flow:inventory">
    <container-name>abcd</container-name>
    <flags>meter-burst</flags>
    <meter-band-headers>
        <meter-band-header>
            <band-burst-size>444</band-burst-size>
            <band-id>0</band-id>
            <band-rate>234</band-rate>
            <dscp-remark-burst-size>5</dscp-remark-burst-size>
            <dscp-remark-rate>12</dscp-remark-rate>
            <prec_level>1</prec_level>
            <meter-band-types>
                <flags>ofpmbt-dscp-remark</flags>
            </meter-band-types>
        </meter-band-header>
    </meter-band-headers>
    <meter-id>1</meter-id>
    <meter-name>Foo</meter-name>
</meter>
```

Note: If you want to try a different meter id, make sure the URL and the body stay in sync. For example, if you wanted to try: meter-id 20 you'd change the URL to "http://:8080/restconf/config/opendaylight-inventory:nodes/node/openflow:1/meter/20" but you would also need to update the 20 in the body to match.

Note: :Provide the IP Address of the machine on which the controller is running.

Check for your meter on the switch

- See your meter on your CPqD switch:

```
COMMAND: $ sudo dpctl tcp:127.0.0.1:6000 meter-config

SENDING:
stat_req{type="mconf", flags="0x0"{meter_id= ffffffff"}


RECEIVED:
stat_repl{type="mconf", flags="0x0", stats=[{meter= c"", flags="4", bands=[{type =␣
↪dscp_remark, rate="12", burst_size="5", prec_level="1"}]}]}
```

Check for your meter in the controller config via RESTCONF

- See your configured flow in POSTMAN with
  - URL "http://:8080/restconf/config/opendaylight-inventory:nodes/node/openflow:1/meter/1"
  - Method: GET
  - You should no longer need to set Request Headers for Accept
  - Note: :Provide the IP Address of the machine on which the controller is running.

Look for your meter stats in the controller operational data via RESTCONF

- See your operational meter stats in POSTMAN with
  - URL "http://:8080/restconfig/operational/opendaylight-inventory:nodes/node/openflow:1/meter/1"
  - Method: GET
  - Note: :Provide the IP Address of the machine on which the controller is running.

Discovering and testing Meter Types

Currently, the openflowplugin has a test-provider that allows you to push various meters through the system from the OSGI command line. Once those meters have been pushed through, you can see them as examples and then use them to see in the config what a particular meter example looks like.

Using addMeter

From the

```
cd openflowplugin/distribution/base/target/distributions-openflowplugin-base-0.0.1-
↪SNAPSHOT-osgipackage/opendaylight
./run.sh
```

Point your CPqD at the controller as described above.

Once you can see your CPqD connected to the controller, at the OSGI command line try running:

```
addMeter openflow:1
```

Once you've done that, use

- GET
- Accept: application/xml
- URL: "http://:8080/restconf/config/opendaylight-inventory:nodes/node/openflow:1/meter/12"
  - Note: :Provide the IP Address of the machine on which the controller is running.

Note: Before attempting to PUT a meter you have created via addMeter, please change its URL and body to, for example, use meter 1 instead of meter 2 or another Meter Id, so you don't collide.

Note: There are several test command providers and the one handling Meter is **OpenflowpluginMeterTestCommandProvider**. Methods, which can be used as **commands in OSGI-console** have prefix _. Examples: addMeter, modifyMeter and removeMeter.

Example Meter

Examples for XML for various Meter Types can be found in the test-scripts bundle of the plugin code with names m1.xml, m2.xml and m3.xml.

Statistics

Overview

This page contains high level detail about the statistics collection mechanism in new OpenFlow plugin.

Statistics collection in new OpenFlow plugin

New OpenFlow plugin collects following statistics from OpenFlow enabled node(switch):

1. Individual Flow Statistics
2. Aggregate Flow Statistics
3. Flow Table Statistics
4. Port Statistics
5. Group Description
6. Group Statistics
7. Meter Configuration
8. Meter Statistics
9. Queue Statistics
10. Node Description
11. Flow Table Features
12. Port Description
13. Group Features
14. Meter Features

At a high level statistics collection mechanism is divided into following three parts

1. Statistics related YANG models, service APIs and notification interfaces are defined in the MD-SAL.
2. Service APIs (RPCs) defined in yang models are implemented by OpenFlow plugin. Notification interfaces are wired up by OpenFlow plugin to MD-SAL.

3. Statistics Manager Module: This module use service APIs implemented by OpenFlow plugin to send statistics requests to all the connected OpenFlow enabled nodes. Module also implements notification interfaces to receive statistics response from nodes. Once it receives statistics response, it augment all the statistics data to the relevant element of the node (like node-connector, flow, table,group, meter) and store it in MD-SAL operational data store.

Details of statistics collection

- Current implementation collects above mentioned statistics (except 10-14) at a periodic interval of 15 seconds.
- Statistics mentioned in 10 to 14 are only fetched when any node connects to the controller because these statistics are just static details about the respective elements.
- Whenever any new element is added to node (like flow, group, meter, queue) it sends statistics request immediately to fetch the latest statistics and store it in the operational data store.
- Whenever any element is deleted from the node, it immediately remove the relevant statistics from operational data store.
- Statistics data are augmented to their respective element stored in the configuration data store. E.g Controller installed flows are stored in configuration data store. Whenever Statistics Manager receive statistics data related to these flow, it search the corresponding flow in the configuration data store and augment statistics in the corresponding location in operational data store. Similar approach is used for other elements of the node.
- Statistics Manager stores flow statistics as an unaccounted flow statistics in operational data store if there is no corresponding flow exist in configuration data store. ID format of unaccounted flow statistics is as follows - [#UF$TABLE**Unaccounted-flow-count - e.g #UF$TABLE*2*1].
- All the unaccounted flows will be cleaned up periodically after every two cycle of flow statistics collection, given that there is no update for these flows in the last two cycles.
- Statistics Manager only entertains statistics response for the request sent by itself. User can write its own statistics collector using the statistics service APIs and notification defined in yang models, it won't effect the functioning of Statistics Manager.
- OpenFlow 1.0 don't have concept of Meter and Group, so Statistics Manager don't send any group & meter related statistics request to OpenFlow 1.0 enabled switch.

RESTCONF Uris to access statistics of various node elements

- Aggregate Flow Statistics & Flow Table Statistics

```
GET  http://<controller-ip>:8080/restconf/operational/opendaylight-inventory:nodes/
↪node/{node-id}/table/{table-id}
```

- Individual Flow Statistics from specific table

```
GET  http://<controller-ip>:8080/restconf/operational/opendaylight-inventory:nodes/
↪node/{node-id}/table/{table-id}/flow/{flow-id}
```

- Group Features & Meter Features Statistics

```
GET  http://<controller-ip>:8080/restconf/operational/opendaylight-inventory:nodes/
↪node/{node-id}
```

- Group Description & Group Statistics

```
GET  http://<controller-ip>:8080/restconf/operational/opendaylight-inventory:nodes/
↪node/{node-id}/group/{group-id}
```

- Meter Configuration & Meter Statistics

```
GET  http://<controller-ip>:8080/restconf/operational/opendaylight-inventory:nodes/
↪node/{node-id}/meter/{meter-id}
```

- Node Connector Statistics

```
GET  http://<controller-ip>:8080/restconf/operational/opendaylight-inventory:nodes/
↪node/{node-id}/node-connector/{node-connector-id}
```

- Queue Statistics

```
GET  http://<controller-ip>:8080/restconf/operational/opendaylight-inventory:nodes/
↪node/{node-id}/node-connector/{node-connector-id}/queue/{queue-id}
```

Bugs

For more details and queuries, please send mail to openflowplugin-dev@lists.opendaylight.org or avishnoi@in.ibm.com If you want to report any bug in statistics collection, please use bugzilla.

Web / Graphical Interface

In the Hydrogen & Helium release, the current Web UI does not support the new OpenFlow 1.3 constructs such as groups, meters, new fields in the flows, multiple flow tables, etc.

Command Line Interface

The following is not exactly CLI - just a set of test commands which can be executed on the OSGI console testing various features in OpenFlow 1.3 spec.

- *OSGI Console Test Provider Commands: Flows*
- *OSGI Console Test Provider Commands: Groups*
- *OSGI Console Test Provider Commands: Meters*
- *OSGI Console Test Provider Commands: Topology Events*

Flows : Test Provider

Currently, the openflowplugin has a test-provider that allows you to push various flows through the system from the OSGI command line. Once those flows have been pushed through, you can see them as examples and then use them to see in the config what a particular flow example looks like.

AddFlow : addMDFlow

Run the controller by executing:

```
cd openflowplugin/distribution/base/target/distributions-openflowplugin-base-0.0.1-
↪SNAPSHOT-osgipackage/opendaylight
./run.sh
```

Point your mininet to the controller by giving the parameters –controller=remote,ip=.

Once you see your node (probably openflow:1 if you've been following along) in the inventory, at the OSGI command line try running:

```
addMDFlow openflow:1 f#
```

Where # is a number between 1 and 80 and openflow:1 is the of the switch. This will create one of 80 possible flows. You can confirm that they were created on the switch.

RemoveFlow : removeMDFlow

Similar to addMDFlow, from the controller OSGi prompt, while your switch is connected to the controller, try running:

```
removeMDFlow openflow:1 f#
```

where # is a number between 1 and 80 and openflow:1 is the of the switch. The flow to be deleted should have same flowid and Nodeid as used for flow add.

ModifyFlow : modifyMDFlow

Similar to addMDFlow, from the controller OSGi prompt, while your switch is connected to the controller, try running:

```
modifyMDFlow openflow:1 f#
```

where # is a number between 1 and 80 and openflow:1 is the of the switch. The flow to be deleted should have same flowid and Nodeid as used for flow add.

Group : Test Provider

Currently, the openflowplugin has a test-provider that allows you to push various flows through the system from the OSGI command line. Once those flows have been pushed through, you can see them as examples and then use them to see in the config what a particular flow example looks like.

AddGroup : addGroup

Run the controller by executing:

```
cd openflowplugin/distribution/base/target/distributions-openflowplugin-base-0.0.1-
↪SNAPSHOT-osgipackage/opendaylight
./run.sh
```

Point your mininet to the controller by giving the parameters –controller=remote,ip=.

Once you see your node (probably openflow:1 if you've been following along) in the inventory, at the OSGI command line try running:

```
addGroup openflow:1 a# g#
```

Where # is a number between 1 and 4 for grouptype(g#) and 1 and 28 for actiontype(a#). You can confirm that they were created on the switch.

RemoveGroup : removeGroup

Run the controller by executing:

```
cd openflowplugin/distribution/base/target/distributions-openflowplugin-base-0.0.1-
↪SNAPSHOT-osgipackage/opendaylight
./run.sh
```

Point your mininet at the controller as described above.

Once you see your node (probably openflow:1 if you've been following along) in the inventory, at the OSGI command line try running:

```
removeGroup openflow:1 a# g#
```

Where # is a number between 1 and 4 for grouptype(g#) and 1 and 28 for actiontype(a#). GroupId should be same as that used for adding the flow. You can confirm that it was removed from the switch.

ModifyGroup : modifyGroup

Run the controller by executing:

```
cd openflowplugin/distribution/base/target/distributions-openflowplugin-base-0.0.1-
↪SNAPSHOT-osgipackage/opendaylight
./run.sh
```

Point your mininet at the controller as described above.

Once you see your node (probably openflow:1 if you've been following along) in the inventory, at the OSGI command line try running:

```
modifyGroup openflow:1 a# g#
```

Where # is a number between 1 and 4 for grouptype(g#) and 1 and 28 for actiontype(a#). GroupId should be same as that used for adding the flow. You can confirm that it was modified on the switch.

Meters : Test Provider

Currently, the openflowplugin has a test-provider that allows you to push various flows through the system from the OSGI command line. Once those flows have been pushed through, you can see them as examples and then use them to see in the config what a particular flow example looks like.

AddMeter : addMeter

Run the controller by executing:

```
cd openflowplugin/distribution/base/target/distributions-openflowplugin-base-0.0.1-
↪SNAPSHOT-osgipackage/opendaylight
./run.sh
```

Point your mininet to the controller by giving the parameters –controller=remote,ip=.

Once you see your node (probably openflow:1 if you've been following along) in the inventory, at the OSGI command line try running:

```
addMeter openflow:1
```

You can now confirm that meter has been created on the switch.

RemoveMeter : removeMeter

Run the controller by executing:

```
cd openflowplugin/distribution/base/target/distributions-openflowplugin-base-0.0.1-
↪SNAPSHOT-osgipackage/opendaylight
./run.sh
```

Point your mininet to the controller by giving the parameters –controller=remote,ip=.

Once you see your node (probably openflow:1 if you've been following along) in the inventory, at the OSGI command line try running:

```
removeMeter openflow:1
```

The CLI takes care of using the same meterId and nodeId as used for meter add. You can confirm that it was removed from the switch.

ModifyMeter : modifyMeter

Run the controller by executing:

```
cd openflowplugin/distribution/base/target/distributions-openflowplugin-base-0.0.1-
↪SNAPSHOT-osgipackage/opendaylight
./run.sh
```

Point your mininet to the controller by giving the parameters –controller=remote,ip=.

Once you see your node (probably openflow:1 if you've been following along) in the inventory, at the OSGI command line try running:

```
modifyMeter openflow:1
```

The CLI takes care of using the same meterId and nodeId as used for meter add. You can confirm that it was modified on the switch.

Topology : Notification

Currently, the openflowplugin has a test-provider that allows you to get notifications for the topology related events like Link-Discovered , Link-Removed events.

Link Discovered Event : Testing

Run the controller by executing:

```
cd openflowplugin/distribution/base/target/distributions-openflowplugin-base-0.0.1-
↪SNAPSHOT-osgipackage/opendaylight
./run.sh
```

Point your mininet to the controller by giving the parameters –controller=remote,ip=. Once the controller is connected to the switch, Link-Discovered event can be tested by initially configuring the specific flows on the switch. For Link Discovered event either table-miss flow or LLDP ether-type flow can be configured.

Configuring Table-Miss flow using OpenflowpluginTestCommandProvider

```
addMDFlow Openflow:1 fTM
```

as per this OpenDaylight_OpenFlow_Plugin:Test_Provider#Flows_:_Test_Provider[link]. *fTM* is the table-miss scenario here.

Once the table-miss flow is configured through above command, we can see the Link-Discovered event in the debug logs on the controller console.

Configuring LLDP ether-type flow using OpenflowpluginTestCommandProvider

```
addMDFlow Openflow:1 0(table-id) f81
```

You can confirm that they were created on the switch.

Once the LLDP ether-type flow is configured through above command, we can see the Link-Discovered event in the debug logs on the controller console.

Link Removed Event : Testing

Having configured either table-miss or lldp ether-type flow on switch, once the switch is disconnected we see the Link-Removed event

Programmatic Interface

The API is documented in the model documentation under the section OpenFlow Services at:

- Models Documentation (OpenFlow Services Section)

Example flows

Overview

The flow examples on this page are tested to work with OVS.

Use, for example, POSTMAN with the following parameters:

```
PUT http://<ctrl-addr>:8080/restconf/config/opendaylight-inventory:nodes/node/<Node-
↪id>/table/<Table-#>/flow/<Flow-#>

- Accept: application/xml
- Content-Type: application/xml
```

For example:

```
PUT http://localhost:8080/restconf/config/opendaylight-inventory:nodes/node/
↪openflow:1/table/2/flow/127
```

Make sure that the Table-# and Flow-# in the URL and in the XML match.

The format of the flow-programming XML is determined by by the grouping *flow* in the opendaylight-flow-types yang model: MISSING LINK.

Match Examples

The format of the XML that describes OpenFlow matches is determined by the opendaylight-match-types yang model: .

IPv4 Dest Address

- Flow=124, Table=2, Priority=2, Instructions=\{Apply_Actions={dec_nw_ttl}}, match=\{ipv4_destination_address=10.0.1.1/24}
- Note that ethernet-type MUST be 2048 (0x800)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <table_id>2</table_id>
    <id>124</id>
    <cookie_mask>255</cookie_mask>
    <installHw>false</installHw>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
        <ipv4-destination>10.0.1.1/24</ipv4-destination>
    </match>
    <hard-timeout>12</hard-timeout>
    <cookie>1</cookie>
    <idle-timeout>34</idle-timeout>
    <flow-name>FooXf1</flow-name>
    <priority>2</priority>
    <barrier>false</barrier>
</flow>
```

Ethernet Src Address

- Flow=126, Table=2, Priority=2, Instructions=\{Apply_Actions={drop}}, match=\{ethernet-source=00:00:00:00:00:01}

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <drop-action/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <table_id>2</table_id>
    <id>126</id>
    <cookie_mask>255</cookie_mask>
    <installHw>false</installHw>
    <match>
        <ethernet-match>
            <ethernet-source>
                <address>00:00:00:00:00:01</address>
            </ethernet-source>
        </ethernet-match>
    </match>
    <hard-timeout>12</hard-timeout>
    <cookie>3</cookie>
    <idle-timeout>34</idle-timeout>
    <flow-name>FooXf3</flow-name>
    <priority>2</priority>
    <barrier>false</barrier>
</flow>
```

Ethernet Src & Dest Addresses, Ethernet Type

- Flow=127, Table=2, Priority=2, Instructions=\{Apply_Actions={drop}}, match=\{ethernet-source=00:00:00:00:23:ae, ethernet-destination=ff:ff:ff:ff:ff:ff, ethernet-type=45}

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-mpls-ttl/>
                </action>
            </apply-actions>
        </instruction>
```

```
    </instructions>
    <table_id>2</table_id>
    <id>127</id>
    <cookie_mask>255</cookie_mask>
    <installHw>false</installHw>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>45</type>
            </ethernet-type>
            <ethernet-destination>
                <address>ff:ff:ff:ff:ff:ff</address>
            </ethernet-destination>
            <ethernet-source>
                <address>00:00:00:00:23:ae</address>
            </ethernet-source>
        </ethernet-match>
    </match>
    <hard-timeout>12</hard-timeout>
    <cookie>4</cookie>
    <idle-timeout>34</idle-timeout>
    <flow-name>FooXf4</flow-name>
    <priority>2</priority>
    <barrier>false</barrier>
</flow>
```

Ethernet Src & Dest Addresses, IPv4 Src & Dest Addresses, Input Port

- Note that ethernet-type MUST be 34887 (0x8847)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-mpls-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <table_id>2</table_id>
    <id>128</id>
    <cookie_mask>255</cookie_mask>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>34887</type>
            </ethernet-type>
            <ethernet-destination>
                <address>ff:ff:ff:ff:ff:ff</address>
            </ethernet-destination>
            <ethernet-source>
```

```
                <address>00:00:00:00:23:ae</address>
            </ethernet-source>
        </ethernet-match>
        <ipv4-source>10.1.2.3/24</ipv4-source>
        <ipv4-destination>20.4.5.6/16</ipv4-destination>
        <in-port>0</in-port>
    </match>
    <hard-timeout>12</hard-timeout>
    <cookie>5</cookie>
    <idle-timeout>34</idle-timeout>
    <flow-name>FooXf5</flow-name>
    <priority>2</priority>
    <barrier>false</barrier>
</flow>
```

Ethernet Src & Dest Addresses, IPv4 Src & Dest Addresses, IP

Protocol #, IP DSCP, IP ECN, Input Port

- Note that ethernet-type MUST be 2048 (0x800)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <table_id>2</table_id>
    <id>130</id>
    <cookie_mask>255</cookie_mask>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
            <ethernet-destination>
                <address>ff:ff:ff:ff:ff:aa</address>
            </ethernet-destination>
            <ethernet-source>
                <address>00:00:00:11:23:ae</address>
            </ethernet-source>
        </ethernet-match>
        <ipv4-source>10.1.2.3/24</ipv4-source>
        <ipv4-destination>20.4.5.6/16</ipv4-destination>
        <ip-match>
            <ip-protocol>56</ip-protocol>
            <ip-dscp>15</ip-dscp>
            <ip-ecn>1</ip-ecn>
        </ip-match>
```

```
        <in-port>0</in-port>
    </match>
    <hard-timeout>12000</hard-timeout>
    <cookie>7</cookie>
    <idle-timeout>12000</idle-timeout>
    <flow-name>FooXf7</flow-name>
    <priority>2</priority>
    <barrier>false</barrier>
</flow>
```

Ethernet Src & Dest Addresses, IPv4 Src & Dest Addresses, TCP Src &

Dest Ports, IP DSCP, IP ECN, Input Port

- Note that ethernet-type MUST be 2048 (0x800)
- Note that IP Protocol Type MUST be 6

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <table_id>2</table_id>
    <id>131</id>
    <cookie_mask>255</cookie_mask>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
            <ethernet-destination>
                <address>ff:ff:29:01:19:61</address>
            </ethernet-destination>
            <ethernet-source>
                <address>00:00:00:11:23:ae</address>
            </ethernet-source>
        </ethernet-match>
        <ipv4-source>17.1.2.3/8</ipv4-source>
        <ipv4-destination>172.168.5.6/16</ipv4-destination>
        <ip-match>
            <ip-protocol>6</ip-protocol>
            <ip-dscp>2</ip-dscp>
            <ip-ecn>2</ip-ecn>
        </ip-match>
        <tcp-source-port>25364</tcp-source-port>
        <tcp-destination-port>8080</tcp-destination-port>
        <in-port>0</in-port>
```

```
    </match>
    <hard-timeout>1200</hard-timeout>
    <cookie>8</cookie>
    <idle-timeout>3400</idle-timeout>
    <flow-name>FooXf8</flow-name>
    <priority>2</priority>
    <barrier>false</barrier>
</flow>
```

Ethernet Src & Dest Addresses, IPv4 Src & Dest Addresses, UDP Src &

Dest Ports, IP DSCP, IP ECN, Input Port

- Note that ethernet-type MUST be 2048 (0x800)
- Note that IP Protocol Type MUST be 17

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <table_id>2</table_id>
    <id>132</id>
    <cookie_mask>255</cookie_mask>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
            <ethernet-destination>
                <address>20:14:29:01:19:61</address>
            </ethernet-destination>
            <ethernet-source>
                <address>00:00:00:11:23:ae</address>
            </ethernet-source>
        </ethernet-match>
        <ipv4-source>19.1.2.3/10</ipv4-source>
        <ipv4-destination>172.168.5.6/18</ipv4-destination>
        <ip-match>
            <ip-protocol>17</ip-protocol>
            <ip-dscp>8</ip-dscp>
            <ip-ecn>3</ip-ecn>
        </ip-match>
        <udp-source-port>25364</udp-source-port>
        <udp-destination-port>8080</udp-destination-port>
        <in-port>0</in-port>
    </match>
```

```
    <hard-timeout>1200</hard-timeout>
    <cookie>9</cookie>
    <idle-timeout>3400</idle-timeout>
    <flow-name>FooXf9</flow-name>
    <priority>2</priority>
    <barrier>false</barrier>
```

Ethernet Src & Dest Addresses, IPv4 Src & Dest Addresses, ICMPv4

Type & Code, IP DSCP, IP ECN, Input Port

- Note that ethernet-type MUST be 2048 (0x800)
- Note that IP Protocol Type MUST be 1

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <table_id>2</table_id>
    <id>134</id>
    <cookie_mask>255</cookie_mask>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
            <ethernet-destination>
                <address>ff:ff:29:01:19:61</address>
            </ethernet-destination>
            <ethernet-source>
                <address>00:00:00:11:23:ae</address>
            </ethernet-source>
        </ethernet-match>
        <ipv4-source>17.1.2.3/8</ipv4-source>
        <ipv4-destination>172.168.5.6/16</ipv4-destination>
        <ip-match>
            <ip-protocol>1</ip-protocol>
            <ip-dscp>27</ip-dscp>
            <ip-ecn>3</ip-ecn>
        </ip-match>
        <icmpv4-match>
            <icmpv4-type>6</icmpv4-type>
            <icmpv4-code>3</icmpv4-code>
        </icmpv4-match>
        <in-port>0</in-port>
    </match>
```

```
    <hard-timeout>1200</hard-timeout>
    <cookie>11</cookie>
    <idle-timeout>3400</idle-timeout>
    <flow-name>FooXf11</flow-name>
    <priority>2</priority>
</flow>
```

Ethernet Src & Dest Addresses, ARP Operation, ARP Src & Target

Transport Addresses, ARP Src & Target Hw Addresses

- Note that ethernet-type MUST be 2054 (0x806)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
                <action>
                    <order>1</order>
                    <dec-mpls-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <table_id>2</table_id>
    <id>137</id>
    <cookie_mask>255</cookie_mask>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2054</type>
            </ethernet-type>
            <ethernet-destination>
                <address>ff:ff:ff:ff:FF:ff</address>
            </ethernet-destination>
            <ethernet-source>
                <address>00:00:FC:01:23:ae</address>
            </ethernet-source>
        </ethernet-match>
        <arp-op>1</arp-op>
        <arp-source-transport-address>192.168.4.1</arp-source-transport-address>
        <arp-target-transport-address>10.21.22.23</arp-target-transport-address>
        <arp-source-hardware-address>
            <address>12:34:56:78:98:AB</address>
        </arp-source-hardware-address>
        <arp-target-hardware-address>
            <address>FE:DC:BA:98:76:54</address>
        </arp-target-hardware-address>
    </match>
    <hard-timeout>12</hard-timeout>
```

```
    <cookie>14</cookie>
    <idle-timeout>34</idle-timeout>
    <flow-name>FooXf14</flow-name>
    <priority>2</priority>
    <barrier>false</barrier>
```

Ethernet Src & Dest Addresses, Ethernet Type, VLAN ID, VLAN PCP

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <table_id>2</table_id>
    <id>138</id>
    <cookie_mask>255</cookie_mask>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
            <ethernet-destination>
                <address>ff:ff:29:01:19:61</address>
            </ethernet-destination>
            <ethernet-source>
                <address>00:00:00:11:23:ae</address>
            </ethernet-source>
        </ethernet-match>
        <vlan-match>
            <vlan-id>
                <vlan-id>78</vlan-id>
                <vlan-id-present>true</vlan-id-present>
            </vlan-id>
            <vlan-pcp>3</vlan-pcp>
      </vlan-match>
    </match>
    <hard-timeout>1200</hard-timeout>
    <cookie>15</cookie>
    <idle-timeout>3400</idle-timeout>
    <flow-name>FooXf15</flow-name>
    <priority>2</priority>
    <barrier>false</barrier>
</flow>
```

Ethernet Src & Dest Addresses, MPLS Label, MPLS TC, MPLS BoS

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <flow-name>FooXf17</flow-name>
    <id>140</id>
    <cookie_mask>255</cookie_mask>
    <cookie>17</cookie>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <priority>2</priority>
    <table_id>2</table_id>
    <strict>false</strict>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>34887</type>
            </ethernet-type>
            <ethernet-destination>
                <address>ff:ff:29:01:19:61</address>
            </ethernet-destination>
            <ethernet-source>
                <address>00:00:00:11:23:ae</address>
            </ethernet-source>
        </ethernet-match>
        <protocol-match-fields>
            <mpls-label>567</mpls-label>
            <mpls-tc>3</mpls-tc>
            <mpls-bos>1</mpls-bos>
        </protocol-match-fields>
    </match>
</flow>
```

IPv6 Src & Dest Addresses

- Note that ethernet-type MUST be 34525

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf18</flow-name>
    <id>141</id>
    <cookie_mask>255</cookie_mask>
    <cookie>18</cookie>
    <table_id>2</table_id>
```

```
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>34525</type>
            </ethernet-type>
        </ethernet-match>
        <ipv6-source>fe80::2acf:e9ff:fe21:6431/128</ipv6-source>
        <ipv6-destination>aabb:1234:2acf:e9ff::fe21:6431/64</ipv6-destination>
    </match>
</flow>
```

Metadata

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf19</flow-name>
    <id>142</id>
    <cookie_mask>255</cookie_mask>
    <cookie>19</cookie>
    <table_id>2</table_id>
    <priority>1</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <metadata>
            <metadata>12345</metadata>
        </metadata>
    </match>
```

```
</flow>
```

Metadata, Metadata Mask

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf20</flow-name>
    <id>143</id>
    <cookie_mask>255</cookie_mask>
    <cookie>20</cookie>
    <table_id>2</table_id>
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <metadata>
            <metadata>12345</metadata>
            <metadata-mask>//FF</metadata-mask>
        </metadata>
    </match>
</flow>
```

IPv6 Src & Dest Addresses, Metadata, IP DSCP, IP ECN, UDP Src & Dest Ports

- Note that ethernet-type MUST be 34525

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf21</flow-name>
    <id>144</id>
    <cookie_mask>255</cookie_mask>
    <cookie>21</cookie>
    <table_id>2</table_id>
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
```

```
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>34525</type>
            </ethernet-type>
        </ethernet-match>
        <ipv6-source>1234:5678:9ABC:DEF0:FDCD:A987:6543:210F/76</ipv6-source>
        <ipv6-destination>fe80::2acf:e9ff:fe21:6431/128</ipv6-destination>
        <metadata>
            <metadata>12345</metadata>
        </metadata>
        <ip-match>
            <ip-protocol>17</ip-protocol>
            <ip-dscp>8</ip-dscp>
            <ip-ecn>3</ip-ecn>
        </ip-match>
        <udp-source-port>25364</udp-source-port>
        <udp-destination-port>8080</udp-destination-port>
    </match>
</flow>
```

IPv6 Src & Dest Addresses, Metadata, IP DSCP, IP ECN, TCP Src & Dest Ports

- Note that ethernet-type MUST be 34525
- Note that IP Protocol MUST be 6

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf22</flow-name>
    <id>145</id>
    <cookie_mask>255</cookie_mask>
    <cookie>22</cookie>
    <table_id>2</table_id>
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
```

```
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>34525</type>
            </ethernet-type>
        </ethernet-match>
        <ipv6-source>1234:5678:9ABC:DEF0:FDCD:A987:6543:210F/76</ipv6-source>
        <ipv6-destination>fe80:2acf:e9ff:fe21::6431/94</ipv6-destination>
        <metadata>
            <metadata>12345</metadata>
        </metadata>
        <ip-match>
            <ip-protocol>6</ip-protocol>
            <ip-dscp>60</ip-dscp>
            <ip-ecn>3</ip-ecn>
        </ip-match>
        <tcp-source-port>183</tcp-source-port>
        <tcp-destination-port>8080</tcp-destination-port>
    </match>
</flow>
```

IPv6 Src & Dest Addresses, Metadata, IP DSCP, IP ECN, TCP Src & Dest Ports, IPv6 Label

- Note that ethernet-type MUST be 34525
- Note that IP Protocol MUST be 6

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf23</flow-name>
    <id>146</id>
    <cookie_mask>255</cookie_mask>
    <cookie>23</cookie>
    <table_id>2</table_id>
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>34525</type>
            </ethernet-type>
        </ethernet-match>
```

```
        <ipv6-source>1234:5678:9ABC:DEF0:FDCD:A987:6543:210F/76</ipv6-source>
        <ipv6-destination>fe80:2acf:e9ff:fe21::6431/94</ipv6-destination>
        <metadata>
            <metadata>12345</metadata>
        </metadata>
        <ipv6-label>
            <ipv6-flabel>33</ipv6-flabel>
        </ipv6-label>
        <ip-match>
            <ip-protocol>6</ip-protocol>
            <ip-dscp>60</ip-dscp>
            <ip-ecn>3</ip-ecn>
        </ip-match>
        <tcp-source-port>183</tcp-source-port>
        <tcp-destination-port>8080</tcp-destination-port>
    </match>
</flow>
```

Tunnel ID

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf24</flow-name>
    <id>147</id>
    <cookie_mask>255</cookie_mask>
    <cookie>24</cookie>
    <table_id>2</table_id>
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <tunnel>
            <tunnel-id>2591</tunnel-id>
        </tunnel>
    </match>
</flow>
```

IPv6 Src & Dest Addresses, Metadata, IP DSCP, IP ECN, ICMPv6 Type & Code, IPv6 Label

- Note that ethernet-type MUST be 34525
- Note that IP Protocol MUST be 58

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf25</flow-name>
    <id>148</id>
    <cookie_mask>255</cookie_mask>
    <cookie>25</cookie>
    <table_id>2</table_id>
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>34525</type>
            </ethernet-type>
        </ethernet-match>
        <ipv6-source>1234:5678:9ABC:DEF0:FDCD:A987:6543:210F/76</ipv6-source>
        <ipv6-destination>fe80:2acf:e9ff:fe21::6431/94</ipv6-destination>
        <metadata>
            <metadata>12345</metadata>
        </metadata>
        <ipv6-label>
            <ipv6-flabel>33</ipv6-flabel>
        </ipv6-label>
        <ip-match>
            <ip-protocol>58</ip-protocol>
            <ip-dscp>60</ip-dscp>
            <ip-ecn>3</ip-ecn>
        </ip-match>
        <icmpv6-match>
            <icmpv6-type>6</icmpv6-type>
            <icmpv6-code>3</icmpv6-code>
        </icmpv6-match>
    </match>
</flow>
```

IPv6 Src & Dest Addresses, Metadata, IP DSCP, IP ECN, TCP Src & Dst Ports, IPv6 Label, IPv6 Ext Header

- Note that ethernet-type MUST be 34525
- Note that IP Protocol MUST be 58

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf27</flow-name>
    <id>150</id>
    <cookie_mask>255</cookie_mask>
    <cookie>27</cookie>
    <table_id>2</table_id>
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <dec-nw-ttl/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>34525</type>
            </ethernet-type>
        </ethernet-match>
        <ipv6-source>1234:5678:9ABC:DEF0:FDCD:A987:6543:210F/76</ipv6-source>
        <ipv6-destination>fe80:2acf:e9ff:fe21::6431/94</ipv6-destination>
        <metadata>
            <metadata>12345</metadata>
        </metadata>
        <ipv6-label>
            <ipv6-flabel>33</ipv6-flabel>
        </ipv6-label>
        <ipv6-ext-header>
            <ipv6-exthdr>0</ipv6-exthdr>
        </ipv6-ext-header>
        <ip-match>
            <ip-protocol>6</ip-protocol>
            <ip-dscp>60</ip-dscp>
            <ip-ecn>3</ip-ecn>
        </ip-match>
        <tcp-source-port>183</tcp-source-port>
        <tcp-destination-port>8080</tcp-destination-port>
    </match>
</flow>
```

Actions

The format of the XML that describes OpenFlow actions is determined by the opendaylight-action-types yang model:
.

Apply Actions

Output to TABLE

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf101</flow-name>
    <id>256</id>
    <cookie_mask>255</cookie_mask>
    <cookie>101</cookie>
    <table_id>2</table_id>
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                        <output-node-connector>TABLE</output-node-connector>
                        <max-length>60</max-length>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>34525</type>
            </ethernet-type>
        </ethernet-match>
        <ipv6-source>1234:5678:9ABC:DEF0:FDCD:A987:6543:210F/76</ipv6-source>
        <ipv6-destination>fe80:2acf:e9ff:fe21::6431/94</ipv6-destination>
        <metadata>
            <metadata>12345</metadata>
        </metadata>
        <ip-match>
            <ip-protocol>6</ip-protocol>
            <ip-dscp>60</ip-dscp>
            <ip-ecn>3</ip-ecn>
        </ip-match>
        <tcp-source-port>183</tcp-source-port>
        <tcp-destination-port>8080</tcp-destination-port>
    </match>
</flow>
```

Output to INPORT

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
```

```
    <strict>false</strict>
    <flow-name>FooXf102</flow-name>
    <id>257</id>
    <cookie_mask>255</cookie_mask>
    <cookie>102</cookie>
    <table_id>2</table_id>
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                        <output-node-connector>INPORT</output-node-connector>
                        <max-length>60</max-length>
                    </output-action>
                </action>
7            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
            <ethernet-destination>
                <address>ff:ff:29:01:19:61</address>
            </ethernet-destination>
            <ethernet-source>
                <address>00:00:00:11:23:ae</address>
            </ethernet-source>
        </ethernet-match>
        <ipv4-source>17.1.2.3/8</ipv4-source>
        <ipv4-destination>172.168.5.6/16</ipv4-destination>
        <ip-match>
            <ip-protocol>6</ip-protocol>
            <ip-dscp>2</ip-dscp>
            <ip-ecn>2</ip-ecn>
        </ip-match>
        <tcp-source-port>25364</tcp-source-port>
        <tcp-destination-port>8080</tcp-destination-port>
    </match>
</flow>
```

Output to Physical Port

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf103</flow-name>
    <id>258</id>
    <cookie_mask>255</cookie_mask>
```

```
    <cookie>103</cookie>
    <table_id>2</table_id>
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                        <output-node-connector>1</output-node-connector>
                        <max-length>60</max-length>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
            <ethernet-destination>
                <address>ff:ff:29:01:19:61</address>
            </ethernet-destination>
            <ethernet-source>
                <address>00:00:00:11:23:ae</address>
            </ethernet-source>
        </ethernet-match>
        <ipv4-source>17.1.2.3/8</ipv4-source>
        <ipv4-destination>172.168.5.6/16</ipv4-destination>
        <ip-match>
            <ip-protocol>6</ip-protocol>
            <ip-dscp>2</ip-dscp>
            <ip-ecn>2</ip-ecn>
        </ip-match>
        <tcp-source-port>25364</tcp-source-port>
        <tcp-destination-port>8080</tcp-destination-port>
    </match>
</flow>
```

Output to LOCAL

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf104</flow-name>
    <id>259</id>
    <cookie_mask>255</cookie_mask>
    <cookie>104</cookie>
    <table_id>2</table_id>
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
```

```
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                        <output-node-connector>LOCAL</output-node-connector>
                        <max-length>60</max-length>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>34525</type>
            </ethernet-type>
        </ethernet-match>
        <ipv6-source>1234:5678:9ABC:DEF0:FDCD:A987:6543:210F/76</ipv6-source>
        <ipv6-destination>fe80:2acf:e9ff:fe21::6431/94</ipv6-destination>
        <metadata>
            <metadata>12345</metadata>
        </metadata>
        <ip-match>
            <ip-protocol>6</ip-protocol>
            <ip-dscp>60</ip-dscp>
            <ip-ecn>3</ip-ecn>
        </ip-match>
        <tcp-source-port>183</tcp-source-port>
        <tcp-destination-port>8080</tcp-destination-port>
    </match>
</flow>
```

Output to NORMAL

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf105</flow-name>
    <id>260</id>
    <cookie_mask>255</cookie_mask>
    <cookie>105</cookie>
    <table_id>2</table_id>
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
```

```
                    <order>0</order>
                    <output-action>
                        <output-node-connector>NORMAL</output-node-connector>
                        <max-length>60</max-length>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>34525</type>
            </ethernet-type>
        </ethernet-match>
        <ipv6-source>1234:5678:9ABC:DEF0:FDCD:A987:6543:210F/84</ipv6-source>
        <ipv6-destination>fe80:2acf:e9ff:fe21::6431/90</ipv6-destination>
        <metadata>
            <metadata>12345</metadata>
        </metadata>
        <ip-match>
            <ip-protocol>6</ip-protocol>
            <ip-dscp>45</ip-dscp>
            <ip-ecn>2</ip-ecn>
        </ip-match>
        <tcp-source-port>20345</tcp-source-port>
        <tcp-destination-port>80</tcp-destination-port>
    </match>
</flow>
```

Output to FLOOD

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf106</flow-name>
    <id>261</id>
    <cookie_mask>255</cookie_mask>
    <cookie>106</cookie>
    <table_id>2</table_id>
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                        <output-node-connector>FLOOD</output-node-connector>
                        <max-length>60</max-length>
                    </output-action>
                </action>
            </apply-actions>
```

```
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>34525</type>
            </ethernet-type>
        </ethernet-match>
        <ipv6-source>1234:5678:9ABC:DEF0:FDCD:A987:6543:210F/100</ipv6-source>
        <ipv6-destination>fe80:2acf:e9ff:fe21::6431/67</ipv6-destination>
        <metadata>
            <metadata>12345</metadata>
        </metadata>
        <ip-match>
            <ip-protocol>6</ip-protocol>
            <ip-dscp>45</ip-dscp>
            <ip-ecn>2</ip-ecn>
        </ip-match>
        <tcp-source-port>20345</tcp-source-port>
        <tcp-destination-port>80</tcp-destination-port>
    </match>
</flow>
```

Output to ALL

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf107</flow-name>
    <id>262</id>
    <cookie_mask>255</cookie_mask>
    <cookie>107</cookie>
    <table_id>2</table_id>
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                        <output-node-connector>ALL</output-node-connector>
                        <max-length>60</max-length>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
```

```
            <ethernet-destination>
                <address>20:14:29:01:19:61</address>
            </ethernet-destination>
            <ethernet-source>
                <address>00:00:00:11:23:ae</address>
            </ethernet-source>
        </ethernet-match>
        <ipv4-source>19.1.2.3/10</ipv4-source>
        <ipv4-destination>172.168.5.6/18</ipv4-destination>
        <ip-match>
            <ip-protocol>17</ip-protocol>
            <ip-dscp>8</ip-dscp>
            <ip-ecn>3</ip-ecn>
        </ip-match>
        <udp-source-port>25364</udp-source-port>
        <udp-destination-port>8080</udp-destination-port>
        <in-port>0</in-port>
    </match>
</flow>
```

Output to CONTROLLER

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf108</flow-name>
    <id>263</id>
    <cookie_mask>255</cookie_mask>
    <cookie>108</cookie>
    <table_id>2</table_id>
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                        <output-node-connector>CONTROLLER</output-node-connector>
                        <max-length>60</max-length>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
            <ethernet-destination>
                <address>20:14:29:01:19:61</address>
            </ethernet-destination>
```

```
            <ethernet-source>
                <address>00:00:00:11:23:ae</address>
            </ethernet-source>
        </ethernet-match>
        <ipv4-source>19.1.2.3/10</ipv4-source>
        <ipv4-destination>172.168.5.6/18</ipv4-destination>
        <ip-match>
            <ip-protocol>17</ip-protocol>
            <ip-dscp>8</ip-dscp>
            <ip-ecn>3</ip-ecn>
        </ip-match>
        <udp-source-port>25364</udp-source-port>
        <udp-destination-port>8080</udp-destination-port>
        <in-port>0</in-port>
    </match>
</flow>
```

Output to ANY

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FooXf109</flow-name>
    <id>264</id>
    <cookie_mask>255</cookie_mask>
    <cookie>109</cookie>
    <table_id>2</table_id>
    <priority>2</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                        <output-node-connector>ANY</output-node-connector>
                        <max-length>60</max-length>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
            <ethernet-destination>
                <address>20:14:29:01:19:61</address>
            </ethernet-destination>
            <ethernet-source>
                <address>00:00:00:11:23:ae</address>
            </ethernet-source>
```

```
        </ethernet-match>
        <ipv4-source>19.1.2.3/10</ipv4-source>
        <ipv4-destination>172.168.5.6/18</ipv4-destination>
        <ip-match>
            <ip-protocol>17</ip-protocol>
            <ip-dscp>8</ip-dscp>
            <ip-ecn>3</ip-ecn>
        </ip-match>
        <udp-source-port>25364</udp-source-port>
        <udp-destination-port>8080</udp-destination-port>
        <in-port>0</in-port>
    </match>
</flow>
```

Push VLAN

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
   <strict>false</strict>
   <instructions>
       <instruction>
           <order>0</order>
           <apply-actions>
              <action>
                 <push-vlan-action>
                     <ethernet-type>33024</ethernet-type>
                 </push-vlan-action>
                 <order>0</order>
              </action>
               <action>
                   <set-field>
                       <vlan-match>
                            <vlan-id>
                                <vlan-id>79</vlan-id>
                                <vlan-id-present>true</vlan-id-present>
                            </vlan-id>
                       </vlan-match>
                   </set-field>
                   <order>1</order>
               </action>
               <action>
                   <output-action>
                       <output-node-connector>5</output-node-connector>
                   </output-action>
                   <order>2</order>
               </action>
           </apply-actions>
       </instruction>
   </instructions>
   <table_id>0</table_id>
   <id>31</id>
   <match>
       <ethernet-match>
           <ethernet-type>
               <type>2048</type>
           </ethernet-type>
```

```
            <ethernet-destination>
                <address>FF:FF:29:01:19:61</address>
            </ethernet-destination>
            <ethernet-source>
                <address>00:00:00:11:23:AE</address>
            </ethernet-source>
        </ethernet-match>
      <in-port>1</in-port>
   </match>
   <flow-name>vlan_flow</flow-name>
   <priority>2</priority>
</flow>
```

Push MPLS

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow
    xmlns="urn:opendaylight:flow:inventory">
    <flow-name>push-mpls-action</flow-name>
    <instructions>
        <instruction>
            <order>3</order>
            <apply-actions>
                <action>
                    <push-mpls-action>
                        <ethernet-type>34887</ethernet-type>
                    </push-mpls-action>
                    <order>0</order>
                </action>
                <action>
                    <set-field>
                        <protocol-match-fields>
                            <mpls-label>27</mpls-label>
                        </protocol-match-fields>
                    </set-field>
                    <order>1</order>
                </action>
                <action>
                    <output-action>
                        <output-node-connector>2</output-node-connector>
                    </output-action>
                    <order>2</order>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <strict>false</strict>
    <id>100</id>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
        <in-port>1</in-port>
        <ipv4-destination>10.0.0.4/32</ipv4-destination>
```

```
    </match>
    <idle-timeout>0</idle-timeout>
    <cookie_mask>255</cookie_mask>
    <cookie>401</cookie>
    <priority>8</priority>
    <hard-timeout>0</hard-timeout>
    <installHw>false</installHw>
    <table_id>0</table_id>
</flow>
```

Swap MPLS

- Note that ethernet-type MUST be 34887

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow
    xmlns="urn:opendaylight:flow:inventory">
    <flow-name>push-mpls-action</flow-name>
    <instructions>
        <instruction>
            <order>2</order>
            <apply-actions>
                <action>
                    <set-field>
                        <protocol-match-fields>
                            <mpls-label>37</mpls-label>
                        </protocol-match-fields>
                    </set-field>
                    <order>1</order>
                </action>
                <action>
                    <output-action>
                        <output-node-connector>2</output-node-connector>
                    </output-action>
                    <order>2</order>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <strict>false</strict>
    <id>101</id>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>34887</type>
            </ethernet-type>
        </ethernet-match>
        <in-port>1</in-port>
        <protocol-match-fields>
            <mpls-label>27</mpls-label>
        </protocol-match-fields>
    </match>
    <idle-timeout>0</idle-timeout>
    <cookie_mask>255</cookie_mask>
    <cookie>401</cookie>
    <priority>8</priority>
```

```
    <hard-timeout>0</hard-timeout>
    <installHw>false</installHw>
    <table_id>0</table_id>
</flow>
```

Pop MPLS

- Note that ethernet-type MUST be 34887
- Issue with OVS 2.1 OVS fix

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow
    xmlns="urn:opendaylight:flow:inventory">
    <flow-name>FooXf10</flow-name>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <pop-mpls-action>
                        <ethernet-type>2048</ethernet-type>
                    </pop-mpls-action>
                    <order>1</order>
                </action>
                <action>
                    <output-action>
                        <output-node-connector>2</output-node-connector>
                        <max-length>60</max-length>
                    </output-action>
                    <order>2</order>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <id>11</id>
    <strict>false</strict>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>34887</type>
            </ethernet-type>
        </ethernet-match>
        <in-port>1</in-port>
        <protocol-match-fields>
            <mpls-label>37</mpls-label>
        </protocol-match-fields>
    </match>
    <idle-timeout>0</idle-timeout>
    <cookie>889</cookie>
    <cookie_mask>255</cookie_mask>
    <installHw>false</installHw>
    <hard-timeout>0</hard-timeout>
    <priority>10</priority>
    <table_id>0</table_id>
</flow>
```

Learn

- Nicira extension defined in https://github.com/osrg/openvswitch/blob/master/include/openflow/nicira-ext.h
- Example section is - https://github.com/osrg/openvswitch/blob/master/include/openflow/nicira-ext.h#L788

```
<flow>
  <id>ICMP_Ingress258a5a5ad-08a8-4ff7-98f5-ef0b96ca3bb8</id>
  <hard-timeout>0</hard-timeout>
  <idle-timeout>0</idle-timeout>
  <match>
    <ethernet-match>
      <ethernet-type>
        <type>2048</type>
      </ethernet-type>
    </ethernet-match>
    <metadata>
      <metadata>2199023255552</metadata>
      <metadata-mask>23058419097020661 76</metadata-mask>
    </metadata>
    <ip-match>
      <ip-protocol>1</ip-protocol>
    </ip-match>
  </match>
  <cookie>110100480</cookie>
  <instructions>
    <instruction>
      <order>0</order>
      <apply-actions>
        <action>
          <order>1</order>
          <nx-resubmit
            xmlns="urn:opendaylight:openflowplugin:extension:nicira:action">
            <table>220</table>
          </nx-resubmit>
        </action>
        <action>
          <order>0</order>
          <nx-learn
            xmlns="urn:opendaylight:openflowplugin:extension:nicira:action">
            <idle-timeout>60</idle-timeout>
            <fin-idle-timeout>0</fin-idle-timeout>
            <hard-timeout>60</hard-timeout>
            <flags>0</flags>
            <table-id>41</table-id>
            <priority>61010</priority>
            <fin-hard-timeout>0</fin-hard-timeout>
            <flow-mods>
              <flow-mod-add-match-from-value>
                <src-ofs>0</src-ofs>
                <value>2048</value>
                <src-field>1538</src-field>
                <flow-mod-num-bits>16</flow-mod-num-bits>
              </flow-mod-add-match-from-value>
            </flow-mods>
            <flow-mods>
              <flow-mod-add-match-from-field>
                <src-ofs>0</src-ofs>
```

```
                <dst-ofs>0</dst-ofs>
                <dst-field>4100</dst-field>
                <src-field>3588</src-field>
                <flow-mod-num-bits>32</flow-mod-num-bits>
              </flow-mod-add-match-from-field>
            </flow-mods>
            <flow-mods>
              <flow-mod-add-match-from-field>
                <src-ofs>0</src-ofs>
                <dst-ofs>0</dst-ofs>
                <dst-field>518</dst-field>
                <src-field>1030</src-field>
                <flow-mod-num-bits>48</flow-mod-num-bits>
              </flow-mod-add-match-from-field>
            </flow-mods>
            <flow-mods>
              <flow-mod-add-match-from-field>
                <src-ofs>0</src-ofs>
                <dst-ofs>0</dst-ofs>
                <dst-field>3073</dst-field>
                <src-field>3073</src-field>
                <flow-mod-num-bits>8</flow-mod-num-bits>
              </flow-mod-add-match-from-field>
            </flow-mods>
            <flow-mods>
              <flow-mod-copy-value-into-field>
                <dst-ofs>0</dst-ofs>
                <value>1</value>
                <dst-field>65540</dst-field>
                <flow-mod-num-bits>8</flow-mod-num-bits>
              </flow-mod-copy-value-into-field>
            </flow-mods>
            <cookie>110100480</cookie>
          </nx-learn>
        </action>
      </apply-actions>
    </instruction>
  </instructions>
  <installHw>true</installHw>
  <barrier>false</barrier>
  <strict>false</strict>
  <priority>61010</priority>
  <table_id>253</table_id>
  <flow-name>ACL</flow-name>
</flow>
```

Opendaylight OpenFlow Plugin: Troubleshooting

empty section

OpFlex agent-ovs User Guide

Introduction

agent-ovs is a policy agent that works with OVS to enforce a group-based policy networking model with locally attached virtual machines or containers. The policy agent is designed to work well with orchestration tools like OpenStack.

Agent Configuration

The agent configuration is handled using its config file which is by default found at "/etc/opflex-agent-ovs/opflex-agent-ovs.conf"

Here is an example configuration file that documents the available options:

```
{
    // Logging configuration
    // "log": {
    //     // Set the log level.
    //     // Possible values in descending order of verbosity:
    //     // "debug7"-"debug0", "debug" (synonym for "debug0"),
    //     // "info", "warning", "error", "fatal"
    //     // Default: "info"
    //     "level": "info"
    // },

    // Configuration related to the OpFlex protocol
    "opflex": {
        // The policy domain for this agent.
        "domain": "openstack",

        // The unique name in the policy domain for this agent.
        "name": "example-agent",

        // a list of peers to connect to, by hostname and port.  One
        // peer, or an anycast pseudo-peer, is sufficient to bootstrap
        // the connection without needing an exhaustive list of all
        // peers.
        "peers": [
            // EXAMPLE:
            // {"hostname": "10.0.0.30", "port": 8009}
        ],

        "ssl": {
            // SSL mode.  Possible values:
            // disabled: communicate without encryption (default)
            // encrypted: encrypt but do not verify peers
            // secure: encrypt and verify peer certificates
            "mode": "encrypted",

            // The path to a directory containing trusted certificate
            // authority public certificates, or a file containing a
            // specific CA certificate.
            // Default: "/etc/ssl/certs"
            "ca-store": "/etc/ssl/certs"
        },
```

```
        "inspector": {
            // Enable the MODB inspector service, which allows
            // inspecting the state of the managed object database.
            // Default: true
            "enabled": true,

            // Listen on the specified socket for the inspector
            // Default: "/var/run/opflex-agent-ovs-inspect.sock"
            "socket-name": "/var/run/opflex-agent-ovs-inspect.sock"
        },

        "notif": {
            // Enable the agent notification service, which sends
            // notifications to interested listeners over a UNIX
            // socket.
            // Default: true
            "enabled": true,

            // Listen on the specified socket for the inspector
            // Default: "/var/run/opflex-agent-ovs-notif.sock"
            "socket-name": "/var/run/opflex-agent-ovs-notif.sock",

            // Set the socket owner user after binding if the user
            // exists
            // Default: do not set the owner
            // "socket-owner": "root",

            // Set the socket group after binding if the group name
            // exists
            // Default: do not set the group
            "socket-group": "opflexep",

            // Set the socket permissions after binding to the
            // specified octal permissions mask
            // Default: do not set the permissions
            "socket-permissions": "770"
        }
    },

    // Endpoint sources provide metadata about local endpoints
    "endpoint-sources": {
        // Filesystem path to monitor for endpoint information
        // Default: no endpoint sources
        "filesystem": ["/var/lib/opflex-agent-ovs/endpoints"]
    },

    // Service sources provide metadata about services that can
    // provide functionality for local endpoints
    "service-sources": {
        // Filesystem path to monitor for service information
        // Default: no service sources
        "filesystem": ["/var/lib/opflex-agent-ovs/services"]
    },

    // Renderers enforce policy obtained via OpFlex.
    // Default: no renderers
    "renderers": {
        // Stitched-mode renderer for interoperating with a
```

```
        // hardware fabric such as ACI
        // EXAMPLE:
        "stitched-mode": {
            // "Integration" bridge used to enforce contracts and forward
            // packets
            "int-bridge-name": "br-int",

            // "Access" bridge used to enforce access control and enforce
            // security groups.
            "access-bridge-name": "br-access",

            // Set encapsulation type.  Must set either vxlan or vlan.
            "encap": {
                // Encapsulate traffic with VXLAN.
                "vxlan" : {
                    // The name of the tunnel interface in OVS
                    "encap-iface": "br0_vxlan0",

                    // The name of the interface whose IP should be used
                    // as the source IP in encapsulated traffic.
                    "uplink-iface": "team0.4093",

                    // The vlan tag, if any, used on the uplink interface.
                    // Set to zero or omit if the uplink is untagged.
                    "uplink-vlan": 4093,

                    // The IP address used for the destination IP in
                    // the encapsulated traffic.  This should be an
                    // anycast IP address understood by the upstream
                    // stiched-mode fabric.
                    "remote-ip": "10.0.0.32",

                    // UDP port number of the encapsulated traffic.
                    "remote-port": 8472
                }

                // Encapsulate traffic with a locally-significant VLAN
                // tag
                // EXAMPLE:
                // "vlan" : {
                //     // The name of the uplink interface in OVS
                //     "encap-iface": "team0"
                // }
            },

            // Configure forwarding policy
            "forwarding": {
                // Configure the virtual distributed router
                "virtual-router": {
                    // Enable virtual distributed router.  Set to true
                    // to enable or false to disable.
                    // Default: true.
                    "enabled": true,

                    // Override MAC address for virtual router.
                    // Default: "00:22:bd:f8:19:ff"
                    "mac": "00:22:bd:f8:19:ff",
```

```
                    // Configure IPv6-related settings for the virtual
                    // router
                    "ipv6" : {
                        // Send router advertisement messages in
                        // response to router solicitation requests as
                        // well as unsolicited advertisements.  This
                        // is not required in stitched mode since the
                        // hardware router will send them.
                        "router-advertisement": false
                    }
                },

                // Configure virtual distributed DHCP server
                "virtual-dhcp": {
                    // Enable virtual distributed DHCP server.  Set to
                    // true to enable or false to disable.
                    // Default: true
                    "enabled": true,

                    // Override MAC address for virtual dhcp server.
                    // Default: "00:22:bd:f8:19:ff"
                    "mac": "00:22:bd:f8:19:ff"
                },

                "endpoint-advertisements": {
                    // Set mode for generation of periodic ARP/NDP
                    // advertisements for endpoints.  Possible values:
                    // disabled: Do not send advertisements
                    // gratuitous-unicast: Send gratuitous endpoint
                    //   advertisements as unicast packets to the router
                    //   mac.
                    // gratuitous-broadcast: Send gratuitous endpoint
                    //   advertisements as broadcast packets.
                    // router-request: Unicast a spoofed request/solicitation
                    //   for the subnet's gateway router.
                    // Default: router-request.
                    "mode": "gratuitous-broadcast"
                }
            },

            // Location to store cached IDs for managing flow state
            // Default: "/var/lib/opflex-agent-ovs/ids"
            "flowid-cache-dir": "/var/lib/opflex-agent-ovs/ids",

            // Location to write multicast groups for the mcast-daemon
            // Default: "/var/lib/opflex-agent-ovs/mcast/opflex-groups.json"
            "mcast-group-file": "/var/lib/opflex-agent-ovs/mcast/opflex-groups.json"
        }
    }
}
```

Endpoint Registration

The agent learns about endpoints using endpoint metadata files located by default in "/var/lib/opflex-agent-ovs/endpoints".

These are JSON-format files such as the (unusually complex) example below:

```
{
    "uuid": "83f18f0b-80f7-46e2-b06c-4d9487b0c754",
    "policy-space-name": "test",
    "endpoint-group-name": "group1",
    "interface-name": "veth0",
    "ip": [
        "10.0.0.1", "fd8f:69d8:c12c:ca62::1"
    ],
    "dhcp4": {
        "ip": "10.200.44.2",
        "prefix-len": 24,
        "routers": ["10.200.44.1"],
        "dns-servers": ["8.8.8.8", "8.8.4.4"],
        "domain": "example.com",
        "static-routes": [
            {
                "dest": "169.254.169.0",
                "dest-prefix": 24,
                "next-hop": "10.0.0.1"
            }
        ]
    },
    "dhcp6": {
        "dns-servers": ["2001:4860:4860::8888", "2001:4860:4860::8844"],
        "search-list": ["test1.example.com", "example.com"]
    },
    "ip-address-mapping": [
        {
           "uuid": "91c5b217-d244-432c-922d-533c6c036ab4",
           "floating-ip": "5.5.5.1",
           "mapped-ip": "10.0.0.1",
           "policy-space-name": "common",
           "endpoint-group-name": "nat-epg"
        },
        {
           "uuid": "22bfdc01-a390-4b6f-9b10-624d4ccb957b",
           "floating-ip": "fdf1:9f86:d1af:6cc9::1",
           "mapped-ip": "fd8f:69d8:c12c:ca62::1",
           "policy-space-name": "common",
           "endpoint-group-name": "nat-epg"
        }
    ],
    "mac": "00:00:00:00:00:01",
    "promiscuous-mode": false
}
```

The possible parameters for these files are:

uuid A globally unique ID for the endpoint

endpoint-group-name The name of the endpoint group for the endpoint

policy-space-name The name of the policy space for the endpoint group.

interface-name The name of the OVS interface to which the endpoint is attached

ip A list of strings contains either IPv4 or IPv6 addresses that the endpoint is allowed to use

mac The MAC address for the endpoint's interface.

promiscuous-mode Allow traffic from this VM to bypass default port security

dhcp4 A distributed DHCPv4 configuration block (see below)

dhcp6 A distributed DHCPv6 configuration block (see below)

ip-address-mapping A list of IP address mapping configuration blocks (see below)

DHCPv4 configuration blocks can contain the following parameters:

ip the IP address to return with DHCP. Must be one of the configured IPv4 addresses.

prefix the subnet prefix length

routers a list of default gateways for the endpoint

dns a list of DNS server addresses

domain The domain name parameter to send in the DHCP reply

static-routes A list of static route configuration blocks, which contains a "dest", "dest-prefix", and "next-hop" parameters to send as static routes to the end host

DHCPv6 configuration blocks can contain the following parameters:

dns A list of DNS servers for the endpoint

search-patch The DNS search path for the endpoint

IP address mapping configuration blocks can contain the following parameters:

uuid a globally unique ID for the virtual endpoint created by the mapping.

floating-ip Map using DNAT to this floating IPv4 or IPv6 address

mapped-ip the source IPv4 or IPv6 address; must be one of the IPs assigned to the endpoint.

endpoint-group-name The name of the endpoint group for the NATed IP

policy-space-name The name of the policy space for the NATed IP

Inspector

The Opflex inspector is a useful command-line tool that will allow you to inspect the state of the managed object database for the agent for debugging and diagnosis purposes.

The command is called "gbp_inspect" and takes the following arguments:

```
# gbp_inspect -h
Usage: gbp_inspect [options]
Allowed options:
  -h [ --help ]                         Print this help message
  --log arg                             Log to the specified file (default
                                        standard out)
  --level arg (=warning)                Use the specified log level (default
                                        warning)
  --syslog                              Log to syslog instead of file or
                                        standard out
  --socket arg (=/usr/var/run/opflex-agent-ovs-inspect.sock)
                                        Connect to the specified UNIX domain
                                        socket (default /usr/var/run/opfl
                                        ex-agent-ovs-inspect.sock)
  -q [ --query ] arg                    Query for a specific object with
                                        subjectname,uri or all objects of a
                                        specific type with subjectname
  -r [ --recursive ]                    Retrieve the whole subtree for each
```

```
                                        returned object
  -f [ --follow-refs ]                  Follow references in returned objects
  --load arg                            Load managed objects from the specified
                                        file into the MODB view
  -o [ --output ] arg                   Output the results to the specified
                                        file (default standard out)
  -t [ --type ] arg (=tree)             Specify the output format: tree,
                                        asciitree, list, or dump (default tree)
  -p [ --props ]                        Include object properties in output
```

Here are some examples of the ways to use this tool.

You can get information about the running system using one or more queries, which consist of an object model class name and optionally the URI of a specific object. The simplest query is to get a single object, nonrecursively:

```
# gbp_inspect -q DmtreeRoot
-- DmtreeRoot,/
# gbp_inspect -q GbpEpGroup
-- GbpEpGroup,/PolicyUniverse/PolicySpace/test/GbpEpGroup/group1/
-- GbpEpGroup,/PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
# gbp_inspect -q GbpEpGroup,/PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
-- GbpEpGroup,/PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
```

You can also display all the properties for each object:

```
# gbp_inspect -p -q GbpeL24Classifier
-- GbpeL24Classifier,/PolicyUniverse/PolicySpace/test/GbpeL24Classifier/classifier4/
      {
        connectionTracking : 1 (reflexive)
        dFromPort          : 80
        dToPort            : 80
        etherT             : 2048 (ipv4)
        name               : classifier4
        prot               : 6
      }
-- GbpeL24Classifier,/PolicyUniverse/PolicySpace/test/GbpeL24Classifier/classifier3/
      {
        etherT : 34525 (ipv6)
        name   : classifier3
        order  : 100
        prot   : 58
      }
-- GbpeL24Classifier,/PolicyUniverse/PolicySpace/test/GbpeL24Classifier/classifier1/
      {
        etherT : 2054 (arp)
        name   : classifier1
        order  : 102
      }
-- GbpeL24Classifier,/PolicyUniverse/PolicySpace/test/GbpeL24Classifier/classifier2/
      {
        etherT : 2048 (ipv4)
        name   : classifier2
        order  : 101
        prot   : 1
      }
```

You can also request to get the all the children of an object you query for:

```
# gbp_inspect -r -q GbpEpGroup,/PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
- GbpEpGroup,/PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
  - GbpeInstContext,/PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
↪GbpeInstContext/
  - GbpEpGroupToNetworkRSrc,/PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
↪GbpEpGroupToNetworkRSrc/
```

You can also follow references found in any object downloads:

```
# gbp_inspect -fr -q GbpEpGroup,/PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
- GbpEpGroup,/PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
  - GbpeInstContext,/PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
↪GbpeInstContext/
  - GbpEpGroupToNetworkRSrc,/PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
↪GbpEpGroupToNetworkRSrc/
- GbpBridgeDomain,/PolicyUniverse/PolicySpace/common/GbpBridgeDomain/bd_ext/
  - GbpBridgeDomainToNetworkRSrc,/PolicyUniverse/PolicySpace/common/GbpBridgeDomain/
↪bd_ext/GbpBridgeDomainToNetworkRSrc/
- GbpFloodDomain,/PolicyUniverse/PolicySpace/common/GbpFloodDomain/fd_ext/
  - GbpFloodDomainToNetworkRSrc,/PolicyUniverse/PolicySpace/common/GbpFloodDomain/fd_
↪ext/GbpFloodDomainToNetworkRSrc/
- GbpRoutingDomain,/PolicyUniverse/PolicySpace/common/GbpRoutingDomain/rd_ext/
  - GbpRoutingDomainToIntSubnetsRSrc,/PolicyUniverse/PolicySpace/common/
↪GbpRoutingDomain/rd_ext/GbpRoutingDomainToIntSubnetsRSrc/152/%2fPolicyUniverse
↪%2fPolicySpace%2fcommon%2fGbpSubnets%2fsubnets_ext%2f/
  - GbpForwardingBehavioralGroupToSubnetsRSrc,/PolicyUniverse/PolicySpace/common/
↪GbpRoutingDomain/rd_ext/GbpForwardingBehavioralGroupToSubnetsRSrc/
- GbpSubnets,/PolicyUniverse/PolicySpace/common/GbpSubnets/subnets_ext/
  - GbpSubnet,/PolicyUniverse/PolicySpace/common/GbpSubnets/subnets_ext/GbpSubnet/
↪subnet_ext4/
  - GbpSubnet,/PolicyUniverse/PolicySpace/common/GbpSubnets/subnets_ext/GbpSubnet/
↪subnet_ext6/
```

OVSDB User Guide

The OVSDB project implements the OVSDB protocol (RFC 7047), as well as plugins to support OVSDB Schemas, such as the Open_vSwitch database schema and the hardware_vtep database schema.

OVSDB Plugins

Overview and Architecture

There are currently two OVSDB Southbound plugins:

- odl-ovsdb-southbound: Implements the OVSDB Open_vSwitch database schema.
- odl-ovsdb-hwvtepsouthbound: Implements the OVSDB hardware_vtep database schema.

These plugins are normally installed and used automatically by higher level applications such as odl-ovsdb-openstack; however, they can also be installed separately and used via their REST APIs as is described in the following sections.

OVSDB Southbound Plugin

The OVSDB Southbound Plugin provides support for managing OVS hosts via an OVSDB model in the MD-SAL which maps to important tables and attributes present in the Open_vSwitch schema. The OVSDB Southbound Plugin is able to connect actively or passively to OVS hosts and operate as the OVSDB manager of the OVS host. Using the OVSDB protocol it is able to manage the OVS database (OVSDB) on the OVS host as defined by the Open_vSwitch schema.

OVSDB YANG Model

The OVSDB Southbound Plugin provides a YANG model which is based on the abstract network topology model.

The details of the OVSDB YANG model are defined in the ovsdb.yang file.

The OVSDB YANG model defines three augmentations:

ovsdb-node-augmentation This augments the network-topology node and maps primarily to the Open_vSwitch table of the OVSDB schema. The ovsdb-node-augmentation is a representation of the OVS host. It contains the following attributes.

- **connection-info** - holds the local and remote IP address and TCP port numbers for the OpenDaylight to OVSDB node connections
- **db-version** - version of the OVSDB database
- **ovs-version** - version of OVS
- **list managed-node-entry** - a list of references to ovsdb-bridge-augmentation nodes, which are the OVS bridges managed by this OVSDB node
- **list datapath-type-entry** - a list of the datapath types supported by the OVSDB node (e.g. *system*, *netdev*) - depends on newer OVS versions
- **list interface-type-entry** - a list of the interface types supported by the OVSDB node (e.g. *internal*, *vxlan*, *gre*, *dpdk*, etc.) - depends on newer OVS verions
- **list openvswitch-external-ids** - a list of the key/value pairs in the Open_vSwitch table external_ids column
- **list openvswitch-other-config** - a list of the key/value pairs in the Open_vSwitch table other_config column
- **list managery-entry** - list of manager information entries and connection status
- **list qos-entries** - list of QoS entries present in the QoS table
- **list queues** - list of queue entries present in the queue table

ovsdb-bridge-augmentation This augments the network-topology node and maps to an specific bridge in the OVSDB bridge table of the associated OVSDB node. It contains the following attributes.

- **bridge-uuid** - UUID of the OVSDB bridge
- **bridge-name** - name of the OVSDB bridge
- **bridge-openflow-node-ref** - a reference (instance-identifier) of the OpenFlow node associated with this bridge
- **list protocol-entry** - the version of OpenFlow protocol to use with the OpenFlow controller
- **list controller-entry** - a list of controller-uuid and is-connected status of the OpenFlow controllers associated with this bridge
- **datapath-id** - the datapath ID associated with this bridge on the OVSDB node

- **datapath-type** - the datapath type of this bridge
- **fail-mode** - the OVSDB fail mode setting of this bridge
- **flow-node** - a reference to the flow node corresponding to this bridge
- **managed-by** - a reference to the ovsdb-node-augmentation (OVSDB node) that is managing this bridge
- **list bridge-external-ids** - a list of the key/value pairs in the bridge table external_ids column for this bridge
- **list bridge-other-configs** - a list of the key/value pairs in the bridge table other_config column for this bridge

ovsdb-termination-point-augmentation This augments the topology termination point model. The OVSDB Southbound Plugin uses this model to represent both the OVSDB port and OVSDB interface for a given port/interface in the OVSDB schema. It contains the following attributes.

- **port-uuid** - UUID of an OVSDB port row
- **interface-uuid** - UUID of an OVSDB interface row
- **name** - name of the port and interface
- **interface-type** - the interface type
- **list options** - a list of port options
- **ofport** - the OpenFlow port number of the interface
- **ofport_request** - the requested OpenFlow port number for the interface
- **vlan-tag** - the VLAN tag value
- **list trunks** - list of VLAN tag values for trunk mode
- **vlan-mode** - the VLAN mode (e.g. access, native-tagged, native-untagged, trunk)
- **list port-external-ids** - a list of the key/value pairs in the port table external_ids column for this port
- **list interface-external-ids** - a list of the key/value pairs in the interface table external_ids interface for this interface
- **list port-other-configs** - a list of the key/value pairs in the port table other_config column for this port
- **list interface-other-configs** - a list of the key/value pairs in the interface table other_config column for this interface
- **list inteface-lldp** - LLDP Auto Attach configuration for the interface
- **qos** - UUID of the QoS entry in the QoS table assigned to this port

Getting Started

To install the OVSDB Southbound Plugin, use the following command at the Karaf console:

```
feature:install odl-ovsdb-southbound-impl-ui
```

After installing the OVSDB Southbound Plugin, and before any OVSDB topology nodes have been created, the OVSDB topology will appear as follows in the configuration and operational MD-SAL.

HTTP GET:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/
 or
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/
```

Result Body:

```
{
  "topology": [
    {
      "topology-id": "ovsdb:1"
    }
  ]
}
```

Where

<controller-ip> is the IP address of the OpenDaylight controller

OpenDaylight as the OVSDB Manager

An OVS host is a system which is running the OVS software and is capable of being managed by an OVSDB manager. The OVSDB Southbound Plugin is capable of connecting to an OVS host and operating as an OVSDB manager. Depending on the configuration of the OVS host, the connection of OpenDaylight to the OVS host will be active or passive.

Active Connection to OVS Hosts

An active connection is when the OVSDB Southbound Plugin initiates the connection to an OVS host. This happens when the OVS host is configured to listen for the connection (i.e. the OVSDB Southbound Plugin is active the the OVS host is passive). The OVS host is configured with the following command:

```
sudo ovs-vsctl set-manager ptcp:6640
```

This configures the OVS host to listen on TCP port 6640.

The OVSDB Southbound Plugin can be configured via the configuration MD-SAL to actively connect to an OVS host.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1
```

Body:

```
{
  "network-topology:node": [
    {
      "node-id": "ovsdb://HOST1",
      "connection-info": {
        "ovsdb:remote-port": "6640",
        "ovsdb:remote-ip": "<ovs-host-ip>"
      }
    }
```

```
  ]
}
```

Where

<ovs-host-ip> is the IP address of the OVS Host

Note that the configuration assigns a *node-id* of "ovsdb://HOST1" to the OVSDB node. This *node-id* will be used as the identifier for this OVSDB node in the MD-SAL.

Query the configuration MD-SAL for the OVSDB topology.

HTTP GET:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/
```

Result Body:

```
{
  "topology": [
    {
      "topology-id": "ovsdb:1",
      "node": [
        {
          "node-id": "ovsdb://HOST1",
          "ovsdb:connection-info": {
            "remote-ip": "<ovs-host-ip>",
            "remote-port": 6640
          }
        }
      ]
    }
  ]
}
```

As a result of the OVSDB node configuration being added to the configuration MD-SAL, the OVSDB Southbound Plugin will attempt to connect with the specified OVS host. If the connection is successful, the plugin will connect to the OVS host as an OVSDB manager, query the schemas and databases supported by the OVS host, and register to monitor changes made to the OVSDB tables on the OVS host. It will also set an external id key and value in the external-ids column of the Open_vSwtich table of the OVS host which identifies the MD-SAL instance identifier of the OVSDB node. This ensures that the OVSDB node will use the same *node-id* in both the configuration and operational MD-SAL.

```
"opendaylight-iid" = "instance identifier of OVSDB node in the MD-SAL"
```

When the OVS host sends the OVSDB Southbound Plugin the first update message after the monitoring has been established, the plugin will populate the operational MD-SAL with the information it receives from the OVS host.

Query the operational MD-SAL for the OVSDB topology.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/
```

Result Body:

```
{
  "topology": [
    {
      "topology-id": "ovsdb:1",
      "node": [
        {
          "node-id": "ovsdb://HOST1",
          "ovsdb:openvswitch-external-ids": [
            {
              "external-id-key": "opendaylight-iid",
              "external-id-value": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb://HOST1']"
            }
          ],
          "ovsdb:connection-info": {
            "local-ip": "<controller-ip>",
            "remote-port": 6640,
            "remote-ip": "<ovs-host-ip>",
            "local-port": 39042
          },
          "ovsdb:ovs-version": "2.3.1-git4750c96",
          "ovsdb:manager-entry": [
            {
              "target": "ptcp:6640",
              "connected": true,
              "number_of_connections": 1
            }
          ]
        }
      ]
    }
  ]
}
```

To disconnect an active connection, just delete the configuration MD-SAL entry.

HTTP DELETE:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1
```

Note in the above example, that / characters which are part of the *node-id* are specified in hexadecimal format as "%2F".

Passive Connection to OVS Hosts

A passive connection is when the OVS host initiates the connection to the OVSDB Southbound Plugin. This happens when the OVS host is configured to connect to the OVSDB Southbound Plugin. The OVS host is configured with the following command:

```
sudo ovs-vsctl set-manager tcp:<controller-ip>:6640
```

The OVSDB Southbound Plugin is configured to listen for OVSDB connections on TCP port 6640. This value can be changed by editing the "./karaf/target/assembly/etc/custom.properties" file and changing the value of the "ovsdb.listenPort" attribute.

When a passive connection is made, the OVSDB node will appear first in the operational MD-SAL. If the Open_vSwitch table does not contain an external-ids value of *opendaylight-iid*, then the *node-id* of the new OVSDB node will be created in the format:

```
"ovsdb://uuid/<actual UUID value>"
```

If there an *opendaylight-iid* value was already present in the external-ids column, then the instance identifier defined there will be used to create the *node-id* instead.

Query the operational MD-SAL for an OVSDB node after a passive connection.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/
```

Result Body:

```
{
  "topology": [
    {
      "topology-id": "ovsdb:1",
      "node": [
        {
          "node-id": "ovsdb://uuid/163724f4-6a70-428a-a8a0-63b2a21f12dd",
          "ovsdb:openvswitch-external-ids": [
            {
              "external-id-key": "system-id",
              "external-id-value": "ecf160af-e78c-4f6b-a005-83a6baa5c979"
            }
          ],
          "ovsdb:connection-info": {
            "local-ip": "<controller-ip>",
            "remote-port": 46731,
            "remote-ip": "<ovs-host-ip>",
            "local-port": 6640
          },
          "ovsdb:ovs-version": "2.3.1-git4750c96",
          "ovsdb:manager-entry": [
            {
              "target": "tcp:10.11.21.7:6640",
              "connected": true,
              "number_of_connections": 1
            }
          ]
        }
      ]
    }
  ]
}
```

Take note of the *node-id* that was created in this case.

Manage Bridges

The OVSDB Southbound Plugin can be used to manage bridges on an OVS host.

This example shows how to add a bridge to the OVSDB node *ovsdb://HOST1*.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbrtest
```

Body:

```
{
  "network-topology:node": [
    {
      "node-id": "ovsdb://HOST1/bridge/brtest",
      "ovsdb:bridge-name": "brtest",
      "ovsdb:protocol-entry": [
        {
          "protocol": "ovsdb:ovsdb-bridge-protocol-openflow-13"
        }
      ],
      "ovsdb:managed-by": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb://HOST1']"
    }
  ]
}
```

Notice that the *ovsdb:managed-by* attribute is specified in the command. This indicates the association of the new bridge node with its OVSDB node.

Bridges can be updated. In the following example, OpenDaylight is configured to be the OpenFlow controller for the bridge.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbrtest
```

Body:

```
{
  "network-topology:node": [
        {
          "node-id": "ovsdb://HOST1/bridge/brtest",
             "ovsdb:bridge-name": "brtest",
              "ovsdb:controller-entry": [
                {
                  "target": "tcp:<controller-ip>:6653"
                }
              ],
             "ovsdb:managed-by": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb://HOST1']"
        }
    ]
}
```

If the OpenDaylight OpenFlow Plugin is installed, then checking on the OVS host will show that OpenDaylight has successfully connected as the controller for the bridge.

```
$ sudo ovs-vsctl show
    Manager "ptcp:6640"
```

```
        is_connected: true
    Bridge brtest
        Controller "tcp:<controller-ip>:6653"
            is_connected: true
        Port brtest
            Interface brtest
                type: internal
    ovs_version: "2.3.1-git4750c96"
```

Query the operational MD-SAL to see how the bridge appears.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbrtest/
```

Result Body:

```
{
  "node": [
    {
      "node-id": "ovsdb://HOST1/bridge/brtest",
      "ovsdb:bridge-name": "brtest",
      "ovsdb:datapath-type": "ovsdb:datapath-type-system",
      "ovsdb:datapath-id": "00:00:da:e9:0c:08:2d:45",
      "ovsdb:managed-by": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb://HOST1']",
      "ovsdb:bridge-external-ids": [
        {
          "bridge-external-id-key": "opendaylight-iid",
          "bridge-external-id-value": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb://HOST1/bridge/brtest']"
        }
      ],
      "ovsdb:protocol-entry": [
        {
          "protocol": "ovsdb:ovsdb-bridge-protocol-openflow-13"
        }
      ],
      "ovsdb:bridge-uuid": "080ce9da-101e-452d-94cd-ee8bef8a4b69",
      "ovsdb:controller-entry": [
        {
          "target": "tcp:10.11.21.7:6653",
          "is-connected": true,
          "controller-uuid": "c39b1262-0876-4613-8bfd-c67eec1a991b"
        }
      ],
      "termination-point": [
        {
          "tp-id": "brtest",
          "ovsdb:port-uuid": "c808ae8d-7af2-4323-83c1-e397696dc9c8",
          "ovsdb:ofport": 65534,
          "ovsdb:interface-type": "ovsdb:interface-type-internal",
          "ovsdb:interface-uuid": "49e9417f-4479-4ede-8faf-7c873b8c0413",
          "ovsdb:name": "brtest"
        }
```

```
      ]
    }
  ]
}
```

Notice that just like with the OVSDB node, an *opendaylight-iid* has been added to the external-ids column of the bridge since it was created via the configuration MD-SAL.

A bridge node may be deleted as well.

HTTP DELETE:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbrtest
```

Manage Ports

Similarly, ports may be managed by the OVSDB Southbound Plugin.

This example illustrates how a port and various attributes may be created on a bridge.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbrtest/termination-point/
↪testport/
```

Body:

```
{
  "network-topology:termination-point": [
    {
      "ovsdb:options": [
        {
          "ovsdb:option": "remote_ip",
          "ovsdb:value" : "10.10.14.11"
        }
      ],
      "ovsdb:name": "testport",
      "ovsdb:interface-type": "ovsdb:interface-type-vxlan",
      "tp-id": "testport",
      "vlan-tag": "1",
      "trunks": [
        {
          "trunk": "5"
        }
      ],
      "vlan-mode":"access"
    }
  ]
}
```

Ports can be updated - add another VLAN trunk.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbrtest/termination-point/
↪testport/
```

Body:

```
{
  "network-topology:termination-point": [
    {
      "ovsdb:name": "testport",
      "tp-id": "testport",
      "trunks": [
        {
          "trunk": "5"
        },
        {
          "trunk": "500"
        }
      ]
    }
  ]
}
```

Query the operational MD-SAL for the port.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbrtest/termination-point/
↪testport/
```

Result Body:

```
{
  "termination-point": [
    {
      "tp-id": "testport",
      "ovsdb:port-uuid": "b1262110-2a4f-4442-b0df-84faf145488d",
      "ovsdb:options": [
        {
          "option": "remote_ip",
          "value": "10.10.14.11"
        }
      ],
      "ovsdb:port-external-ids": [
        {
          "external-id-key": "opendaylight-iid",
          "external-id-value": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb://HOST1/bridge/brtest']/network-
↪topology:termination-point[network-topology:tp-id='testport']"
        }
      ],
      "ovsdb:interface-type": "ovsdb:interface-type-vxlan",
      "ovsdb:trunks": [
        {
          "trunk": 5
        },
        {
          "trunk": 500
        }
```

```
      ],
      "ovsdb:vlan-mode": "access",
      "ovsdb:vlan-tag": 1,
      "ovsdb:interface-uuid": "7cec653b-f407-45a8-baec-7eb36b6791c9",
      "ovsdb:name": "testport",
      "ovsdb:ofport": 1
    }
  ]
}
```

Remember that the OVSDB YANG model includes both OVSDB port and interface table attributes in the termination-point augmentation. Both kinds of attributes can be seen in the examples above. Again, note the creation of an *opendaylight-iid* value in the external-ids column of the port table.

Delete a port.

HTTP DELETE:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbrtest2/termination-point/
↪testport/
```

Overview of QoS and Queue

The OVSDB Southbound Plugin provides the capability of managing the QoS and Queue tables on an OVS host with OpenDaylight configured as the OVSDB manager.

QoS and Queue Tables in OVSDB

The OVSDB includes a QoS and Queue table. Unlike most of the other tables in the OVSDB, except the Open_vSwitch table, the QoS and Queue tables are "root set" tables, which means that entries, or rows, in these tables are not automatically deleted if they can not be reached directly or indirectly from the Open_vSwitch table. This means that QoS entries can exist and be managed independently of whether or not they are referenced in a Port entry. Similarly, Queue entries can be managed independently of whether or not they are referenced by a QoS entry.

Modelling of QoS and Queue Tables in OpenDaylight MD-SAL

Since the QoS and Queue tables are "root set" tables, they are modeled in the OpenDaylight MD-SAL as lists which are part of the attributes of the OVSDB node model.

The MD-SAL QoS and Queue models have an additonal identifier attribute per entry (e.g. "qos-id" or "queue-id") which is not present in the OVSDB schema. This identifier is used by the MD-SAL as a key for referencing the entry. If the entry is created originally from the configuration MD-SAL, then the value of the identifier is whatever is specified by the configuration. If the entry is created on the OVSDB node and received by OpenDaylight in an operational update, then the id will be created in the following format.

```
"queue-id": "queue://<UUID>"
"qos-id": "qos://<UUID>"
```

The UUID in the above identifiers is the actual UUID of the entry in the OVSDB database.

When the QoS or Queue entry is created by the configuration MD-SAL, the identifier will be configured as part of the external-ids column of the entry. This will ensure that the corresponding entry that is created in the operational MD-SAL uses the same identifier.

```
"queues-external-ids": [
  {
    "queues-external-id-key": "opendaylight-queue-id",
    "queues-external-id-value": "QUEUE-1"
  }
]
```

See more in the examples that follow in this section.

The QoS schema in OVSDB currently defines two types of QoS entries.

- linux-htb
- linux-hfsc

These QoS types are defined in the QoS model. Additional types will need to be added to the model in order to be supported. See the examples that folow for how the QoS type is specified in the model.

QoS entries can be configured with addtional attritubes such as "max-rate". These are configured via the *other-config* column of the QoS entry. Refer to OVSDB schema (in the reference section below) for all of the relevant attributes that can be configured. The examples in the rest of this section will demonstrate how the other-config column may be configured.

Similarly, the Queue entries may be configured with additional attributes via the other-config column.

Managing QoS and Queues via Configuration MD-SAL

This section will show some examples on how to manage QoS and Queue entries via the configuration MD-SAL. The examples will be illustrated by using RESTCONF (see QoS and Queue Postman Collection).

A pre-requisite for managing QoS and Queue entries is that the OVS host must be present in the configuration MD-SAL.

For the following examples, the following OVS host is configured.

HTTP POST:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/
```

Body:

```
{
  "node": [
    {
      "node-id": "ovsdb:HOST1",
      "connection-info": {
        "ovsdb:remote-ip": "<ovs-host-ip>",
        "ovsdb:remote-port": "<ovs-host-ovsdb-port>"
      }
    }
  ]
}
```

Where

- <*controller-ip*> is the IP address of the OpenDaylight controller
- <*ovs-host-ip*> is the IP address of the OVS host

- *<ovs-host-ovsdb-port>* is the TCP port of the OVSDB server on the OVS host (e.g. 6640)

This command creates an OVSDB node with the node-id "ovsdb:HOST1". This OVSDB node will be used in the following examples.

QoS and Queue entries can be created and managed without a port, but ultimately, QoS entries are associated with a port in order to use them. For the following examples a test bridge and port will be created.

Create the test bridge.

HTTP PUT

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fbr-test
```

Body:

```
{
  "network-topology:node": [
    {
      "node-id": "ovsdb:HOST1/bridge/br-test",
      "ovsdb:bridge-name": "br-test",
      "ovsdb:managed-by": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb:HOST1']"
    }
  ]
}
```

Create the test port (which is modeled as a termination point in the OpenDaylight MD-SAL).

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fbr-test/termination-point/testport/
```

Body:

```
{
  "network-topology:termination-point": [
    {
      "ovsdb:name": "testport",
      "tp-id": "testport"
    }
  ]
}
```

If all of the previous steps were successful, a query of the operational MD-SAL should look something like the following results. This indicates that the configuration commands have been successfully instantiated on the OVS host.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fbr-test
```

Result Body:

```
{
  "node": [
```

```
    {
      "node-id": "ovsdb:HOST1/bridge/br-test",
      "ovsdb:bridge-name": "br-test",
      "ovsdb:datapath-type": "ovsdb:datapath-type-system",
      "ovsdb:managed-by": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb:HOST1']",
      "ovsdb:datapath-id": "00:00:8e:5d:22:3d:09:49",
      "ovsdb:bridge-external-ids": [
        {
          "bridge-external-id-key": "opendaylight-iid",
          "bridge-external-id-value": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb:HOST1/bridge/br-test']"
        }
      ],
      "ovsdb:bridge-uuid": "3d225d8d-d060-4909-93ef-6f4db58ef7cc",
      "termination-point": [
        {
          "tp-id": "br=-est",
          "ovsdb:port-uuid": "f85f7aa7-4956-40e4-9c94-e6ca2d5cd254",
          "ovsdb:ofport": 65534,
          "ovsdb:interface-type": "ovsdb:interface-type-internal",
          "ovsdb:interface-uuid": "29ff3692-6ed4-4ad7-a077-1edc277ecb1a",
          "ovsdb:name": "br-test"
        },
        {
          "tp-id": "testport",
          "ovsdb:port-uuid": "aa79a8e2-147f-403a-9fa9-6ee5ec276f08",
          "ovsdb:port-external-ids": [
            {
              "external-id-key": "opendaylight-iid",
              "external-id-value": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb:HOST1/bridge/br-test']/network-
↪topology:termination-point[network-topology:tp-id='testport']"
            }
          ],
          "ovsdb:interface-uuid": "e96f282e-882c-41dd-a870-80e6b29136ac",
          "ovsdb:name": "testport"
        }
      ]
    }
  ]
}
```

Create Queue

Create a new Queue in the configuration MD-SAL.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:queues/QUEUE-1/
```

Body:

```
{
  "ovsdb:queues": [
    {
      "queue-id": "QUEUE-1",
      "dscp": 25,
      "queues-other-config": [
        {
          "queue-other-config-key": "max-rate",
          "queue-other-config-value": "3600000"
        }
      ]
    }
  ]
}
```

Query Queue

Now query the operational MD-SAL for the Queue entry.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:queues/QUEUE-1/
```

Result Body:

```
{
  "ovsdb:queues": [
    {
      "queue-id": "QUEUE-1",
      "queues-other-config": [
        {
          "queue-other-config-key": "max-rate",
          "queue-other-config-value": "3600000"
        }
      ],
      "queues-external-ids": [
        {
          "queues-external-id-key": "opendaylight-queue-id",
          "queues-external-id-value": "QUEUE-1"
        }
      ],
      "queue-uuid": "83640357-3596-4877-9527-b472aa854d69",
      "dscp": 25
    }
  ]
}
```

Create QoS

Create a QoS entry. Note that the UUID of the Queue entry, obtained by querying the operational MD-SAL of the Queue entry, is specified in the queue-list of the QoS entry. Queue entries may be added to the QoS entry at the creation of the QoS entry, or by a subsequent update to the QoS entry.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:qos-entries/QOS-1/
```

Body:

```
{
  "ovsdb:qos-entries": [
    {
      "qos-id": "QOS-1",
      "qos-type": "ovsdb:qos-type-linux-htb",
      "qos-other-config": [
        {
          "other-config-key": "max-rate",
          "other-config-value": "4400000"
        }
      ],
      "queue-list": [
        {
          "queue-number": "0",
          "queue-uuid": "83640357-3596-4877-9527-b472aa854d69"
        }
      ]
    }
  ]
}
```

Query QoS

Query the operational MD-SAL for the QoS entry.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:qos-entries/QOS-1/
```

Result Body:

```
{
  "ovsdb:qos-entries": [
    {
      "qos-id": "QOS-1",
      "qos-other-config": [
        {
          "other-config-key": "max-rate",
          "other-config-value": "4400000"
        }
      ],
      "queue-list": [
        {
          "queue-number": 0,
          "queue-uuid": "83640357-3596-4877-9527-b472aa854d69"
        }
      ],
      "qos-type": "ovsdb:qos-type-linux-htb",
      "qos-external-ids": [
        {
```

```
          "qos-external-id-key": "opendaylight-qos-id",
          "qos-external-id-value": "QOS-1"
        }
      ],
      "qos-uuid": "90ba9c60-3aac-499d-9be7-555f19a6bb31"
    }
  ]
}
```

Add QoS to a Port

Update the termination point entry to include the UUID of the QoS entry, obtained by querying the operational MD-SAL, to associate a QoS entry with a port.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fbr-test/termination-point/testport/
```

Body:

```
{
  "network-topology:termination-point": [
    {
      "ovsdb:name": "testport",
      "tp-id": "testport",
      "qos": "90ba9c60-3aac-499d-9be7-555f19a6bb31"
    }
  ]
}
```

Query the Port

Query the operational MD-SAL to see how the QoS entry appears in the termination point model.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fbr-test/termination-point/testport/
```

Result Body:

```
{
  "termination-point": [
    {
      "tp-id": "testport",
      "ovsdb:port-uuid": "aa79a8e2-147f-403a-9fa9-6ee5ec276f08",
      "ovsdb:port-external-ids": [
        {
          "external-id-key": "opendaylight-iid",
          "external-id-value": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb:HOST1/bridge/br-test']/network-
↪topology:termination-point[network-topology:tp-id='testport']"
        }
```

```
      ],
      "ovsdb:qos": "90ba9c60-3aac-499d-9be7-555f19a6bb31",
      "ovsdb:interface-uuid": "e96f282e-882c-41dd-a870-80e6b29136ac",
      "ovsdb:name": "testport"
    }
  ]
}
```

Query the OVSDB Node

Query the operational MD-SAL for the OVS host to see how the QoS and Queue entries appear as lists in the OVS node model.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1/
```

Result Body (edited to only show information relevant to the QoS and Queue entries):

```
{
  "node": [
    {
      "node-id": "ovsdb:HOST1",
      <content edited out>
      "ovsdb:queues": [
        {
          "queue-id": "QUEUE-1",
          "queues-other-config": [
            {
              "queue-other-config-key": "max-rate",
              "queue-other-config-value": "3600000"
            }
          ],
          "queues-external-ids": [
            {
              "queues-external-id-key": "opendaylight-queue-id",
              "queues-external-id-value": "QUEUE-1"
            }
          ],
          "queue-uuid": "83640357-3596-4877-9527-b472aa854d69",
          "dscp": 25
        }
      ],
      "ovsdb:qos-entries": [
        {
          "qos-id": "QOS-1",
          "qos-other-config": [
            {
              "other-config-key": "max-rate",
              "other-config-value": "4400000"
            }
          ],
          "queue-list": [
            {
              "queue-number": 0,
```

```
            "queue-uuid": "83640357-3596-4877-9527-b472aa854d69"
          }
        ],
        "qos-type": "ovsdb:qos-type-linux-htb",
        "qos-external-ids": [
          {
            "qos-external-id-key": "opendaylight-qos-id",
            "qos-external-id-value": "QOS-1"
          }
        ],
        "qos-uuid": "90ba9c60-3aac-499d-9be7-555f19a6bb31"
      }
    ]
    <content edited out>
  }
 ]
}
```

Remove QoS from a Port

This example removes a QoS entry from the termination point and associated port. Note that this is a PUT command on the termination point with the QoS attribute absent. Other attributes of the termination point should be included in the body of the command so that they are not inadvertantly removed.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fbr-test/termination-point/testport/
```

Body:

```
{
  "network-topology:termination-point": [
    {
      "ovsdb:name": "testport",
      "tp-id": "testport"
    }
  ]
}
```

Remove a Queue from QoS

This example removes the specific Queue entry from the queue list in the QoS entry. The queue entry is specified by the queue number, which is "0" in this example.

HTTP DELETE:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:qos-entries/QOS-1/queue-list/0/
```

Remove Queue

Once all references to a specific queue entry have been removed from QoS entries, the Queue itself can be removed.

HTTP DELETE:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:queues/QUEUE-1/
```

Remove QoS

The QoS entry may be removed when it is no longer referenced by any ports.

HTTP DELETE:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:qos-entries/QOS-1/
```

References

Openvswitch schema

OVSDB and Netvirt Postman Collection

OVSDB Hardware VTEP SouthBound Plugin

Overview

Hwvtepsouthbound plugin is used to configure a hardware VTEP which implements hardware ovsdb schema. This page will show how to use RESTConf API of hwvtepsouthbound. There are two ways to connect to ODL:

switch initiates connection..

Both will be introduced respectively.

User Initiates Connection

Prerequisite

Configure the hwvtep device/node to listen for the tcp connection in passive mode. In addition, management IP and tunnel source IP are also configured. After all this configuration is done, a physical switch is created automatically by the hwvtep node.

Connect to a hwvtep device/node

Send below Restconf request if you want to initiate the connection to a hwvtep node from the controller, where listening IP and port of hwvtep device/node are provided.

REST API: POST http://odl:8181/restconf/config/network-topology:network-topology/topology/hwvtep:1/

```
{
 "network-topology:node": [
       {
           "node-id": "hwvtep://192.168.1.115:6640",
           "hwvtep:connection-info":
```

```
            {
                "hwvtep:remote-port": 6640,
                "hwvtep:remote-ip": "192.168.1.115"
            }
        }
    ]
}
```

Please replace *odl* in the URL with the IP address of your OpenDaylight controller and change *192.168.1.115* to your hwvtep node IP.

NOTE: The format of node-id is fixed. It will be one of the two:

User initiates connection from ODL:

```
hwvtep://ip:port
```

Switch initiates connection:

```
hwvtep://uuid/<uuid of switch>
```

The reason for using UUID is that we can distinguish between multiple switches if they are behind a NAT.

After this request is completed successfully, we can get the physical switch from the operational data store.

REST API: GET http://odl:8181/restconf/operational/network-topology:network-topology/topology/hwvtep:1/node/hwvtep:%2F%2F192.168.1.115:6640

There is no body in this request.

The response of the request is:

```
{
    "node": [
        {
          "node-id": "hwvtep://192.168.1.115:6640",
          "hwvtep:switches": [
            {
              "switch-ref": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='hwvtep:1']/network-
↪topology:node[network-topology:node-id='hwvtep://192.168.1.115:6640/physicalswitch/
↪br0']"
            }
          ],
          "hwvtep:connection-info": {
            "local-ip": "192.168.92.145",
            "local-port": 47802,
            "remote-port": 6640,
            "remote-ip": "192.168.1.115"
          }
        },
        {
          "node-id": "hwvtep://192.168.1.115:6640/physicalswitch/br0",
          "hwvtep:management-ips": [
            {
              "management-ips-key": "192.168.1.115"
            }
          ],
          "hwvtep:physical-switch-uuid": "37eb5abd-a6a3-4aba-9952-a4d301bdf371",
          "hwvtep:managed-by": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='hwvtep:1']/network-
↪topology:node[network-topology:node-id='hwvtep://192.168.1.115:6640']",
```

```
            "hwvtep:hwvtep-node-description": "",
            "hwvtep:tunnel-ips": [
              {
                "tunnel-ips-key": "192.168.1.115"
              }
            ],
            "hwvtep:hwvtep-node-name": "br0"
          }
        ]
}
```

If there is a physical switch which has already been created by manual configuration, we can get the node-id of the physical switch from this response, which is presented in "swith-ref". If the switch does not exist, we need to create the physical switch. Currently, most hwvtep devices do not support running multiple switches.

Create a physical switch

REST API: POST http://odl:8181/restconf/config/network-topology:network-topology/topology/hwvtep:1/

request body:

```
{
 "network-topology:node": [
        {
            "node-id": "hwvtep://192.168.1.115:6640/physicalswitch/br0",
            "hwvtep-node-name": "ps0",
            "hwvtep-node-description": "",
            "management-ips": [
              {
                "management-ips-key": "192.168.1.115"
              }
            ],
            "tunnel-ips": [
              {
                "tunnel-ips-key": "192.168.1.115"
              }
            ],
            "managed-by": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='hwvtep:1']/network-
↪topology:node[network-topology:node-id='hwvtep://192.168.1.115:6640']"
        }
   ]
}
```

Note: "managed-by" must provided by user. We can get its value after the step *Connect to a hwvtep device/node* since the node-id of hwvtep device is provided by user. "managed-by" is a reference typed of instance identifier. Though the instance identifier is a little complicated for RestConf, the primary user of hwvtepsouthbound plugin will be provider-type code such as NetVirt and the instance identifier is much easier to write code for.

Create a logical switch

Creating a logical switch is effectively creating a logical network. For VxLAN, it is a tunnel network with the same VNI.

REST API: POST http://odl:8181/restconf/config/network-topology:network-topology/topology/hwvtep:1/node/hwvtep:%2F%2F192.168.1.115:6640

request body:

```
{
 "logical-switches": [
       {
           "hwvtep-node-name": "ls0",
           "hwvtep-node-description": "",
           "tunnel-key": "10000"
        }
   ]
}
```

Create a physical locator

After the VXLAN network is ready, we will add VTEPs to it. A VTEP is described by a physical locator.

REST API: POST http://odl:8181/restconf/config/network-topology:network-topology/topology/hwvtep:1/node/hwvtep:%2F%2F192.168.1.115:6640

request body:

```
{
 "termination-point": [
      {
          "tp-id": "vxlan_over_ipv4:192.168.0.116",
          "encapsulation-type": "encapsulation-type-vxlan-over-ipv4",
          "dst-ip": "192.168.0.116"
          }
     ]
}
```

The "tp-id" of locator is "{encapsualation-type}: {dst-ip}".

Note: As far as we know, the OVSDB database does not allow the insertion of a new locator alone. So, no locator is inserted after this request is sent. We will trigger off the creation until other entity refer to it, such as remote-mcast-macs.

Create a remote-mcast-macs entry

After adding a physical locator to a logical switch, we need to create a remote-mcast-macs entry to handle unknown traffic.

REST API: POST http://odl:8181/restconf/config/network-topology:network-topology/topology/hwvtep:1/node/hwvtep:%2F%2F192.168.1.115:6640

request body:

```
{
 "remote-mcast-macs": [
       {
           "mac-entry-key": "00:00:00:00:00:00",
           "logical-switch-ref": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='hwvtep:1']/network-
↪topology:node[network-topology:node-id='hwvtep://192.168.1.115:6640']/
↪hwvtep:logical-switches[hwvtep:hwvtep-node-name='ls0']",
```

```
            "locator-set": [
                  {
                        "locator-ref": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='hwvtep:1']/network-
↪topology:node[network-topology:node-id='hwvtep://219.141.189.115:6640']/network-
↪topology:termination-point[network-topology:tp-id='vxlan_over_ipv4:192.168.0.116']"
                  }
            ]
        }
    ]
}
```

The physical locator *vxlan_over_ipv4:192.168.0.116* is just created in "Create a physical locator". It should be noted that list "locator-set" is immutable, that is, we must provide a set of "locator-ref" as a whole.

Note: "00:00:00:00:00:00" stands for "unknown-dst" since the type of mac-entry-key is yang:mac and does not accept "unknown-dst".

Create a physical port

Now we add a physical port into the physical switch "hwvtep://192.168.1.115:6640/physicalswitch/br0". The port is attached with a physical server or an L2 network and with the vlan 100.

REST API: POST http://odl:8181/restconf/config/network-topology:network-topology/topology/hwvtep:1/node/hwvtep:%2F%2F192.168.1.115:6640%2Fphysicalswitch%2Fbr0

```
{
 "network-topology:termination-point": [
        {
            "tp-id": "port0",
            "hwvtep-node-name": "port0",
            "hwvtep-node-description": "",
            "vlan-bindings": [
                {
                  "vlan-id-key": "100",
                  "logical-switch-ref": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='hwvtep:1']/network-
↪topology:node[network-topology:node-id='hwvtep://192.168.1.115:6640']/
↪hwvtep:logical-switches[hwvtep:hwvtep-node-name='ls0']"
                }
          ]
        }
    ]
}
```

At this point, we have completed the basic configuration.

Typically, hwvtep devices learn local MAC addresses automatically. But they also support getting MAC address entries from ODL.

Create a local-mcast-macs entry

It is similar to *Create a remote-mcast-macs entry*.

Create a remote-ucast-macs

REST API: POST http://odl:8181/restconf/config/network-topology:network-topology/topology/hwvtep:1/node/hwvtep:%2F%2F192.168.1.115:6640

```
request body:
```

```
{
 "remote-ucast-macs": [
       {
           "mac-entry-key": "11:11:11:11:11:11",
           "logical-switch-ref": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='hwvtep:1']/network-
↪topology:node[network-topology:node-id='hwvtep://192.168.1.115:6640']/
↪hwvtep:logical-switches[hwvtep:hwvtep-node-name='ls0']",
           "ipaddr": "1.1.1.1",
           "locator-ref": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='hwvtep:1']/network-
↪topology:node[network-topology:node-id='hwvtep://192.168.1.115:6640']/network-
↪topology:termination-point[network-topology:tp-id='vxlan_over_ipv4:192.168.0.116']"
       }
   ]
}
```

Create a local-ucast-macs entry

This is similar to *Create a remote-ucast-macs*.

Switch Initiates Connection

We do not need to connect to a hwvtep device/node when the switch initiates the connection. After switches connect to ODL successfully, we get the node-id's of switches by reading the operational data store. Once the node-id of a hwvtep device is received, the remaining steps are the same as when the user initiates the connection.

References

https://wiki.opendaylight.org/view/User_talk:Pzhang

PCEP User Guide

This guide contains information on how to use the OpenDaylight Path Computation Element Configuration Protocol (PCEP) plugin. The user should learn about PCEP basic concepts, supported capabilities, configuration and operations.

Contents

Overview

This section provides a high-level overview of the PCEP, SDN use-cases and OpenDaylight implementation.

Contents

Path Computation Element Communication Protocol

The Path Computation Element (PCE) Communication Protocol (PCEP) is used for communication between a Path Computation Client (PCC) and a PCE in context of MPLS and GMPLS Traffic Engineering (TE) Label Switched Paths (LSPs). This interaction include path computation requests and computation replies. The PCE operates on a network graph, built from the (Traffic Engineering Database) TED, in order to compute paths based on the path computation request issued by the PCC. The path computation request includes the source and destination of the path and set of constrains to be applied during the computation. The PCE response contains the computed path or the computation failure reason. The PCEP operates on top the TCP, which provides reliable communication.

PCEP in SDN

The Path Computation Element perfectly fits into the centralized SDN controller architecture. The PCE's knowledge of the availability of network resources (i.e. TED) and active LSPs awareness (LSP-DB) allows to perform automated application-driven network operations:

- LSP Re-optimization
- Resource defragmentation
- Link failure restoration
- Auto-bandwidth adjustment
- Bandwidth scheduling
- Shared Risk Link Group (SRLG) diversity maintenance

OpenDaylight PCEP plugin

The OpenDaylight PCEP plugin provides all basic service units necessary to build-up a PCE-based controller. In addition, it offers LSP management functionality for Active Stateful PCE - the cornerstone for majority of PCE-enabled SDN solutions. It consists of the following components:

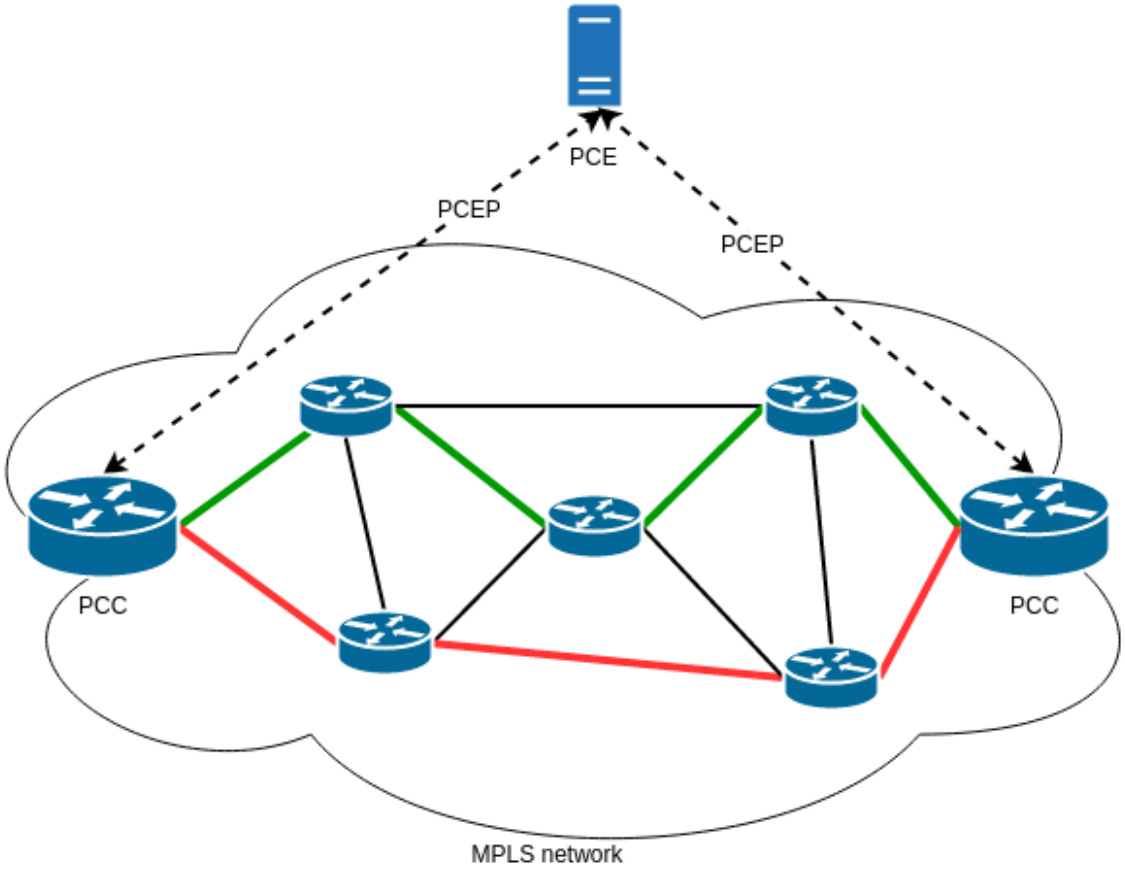


Fig. 1.82: PCE-based architecture.

- Protocol library
- PCEP session handling
- Stateful PCE LSP-DB
- Active Stateful PCE LSP Operations

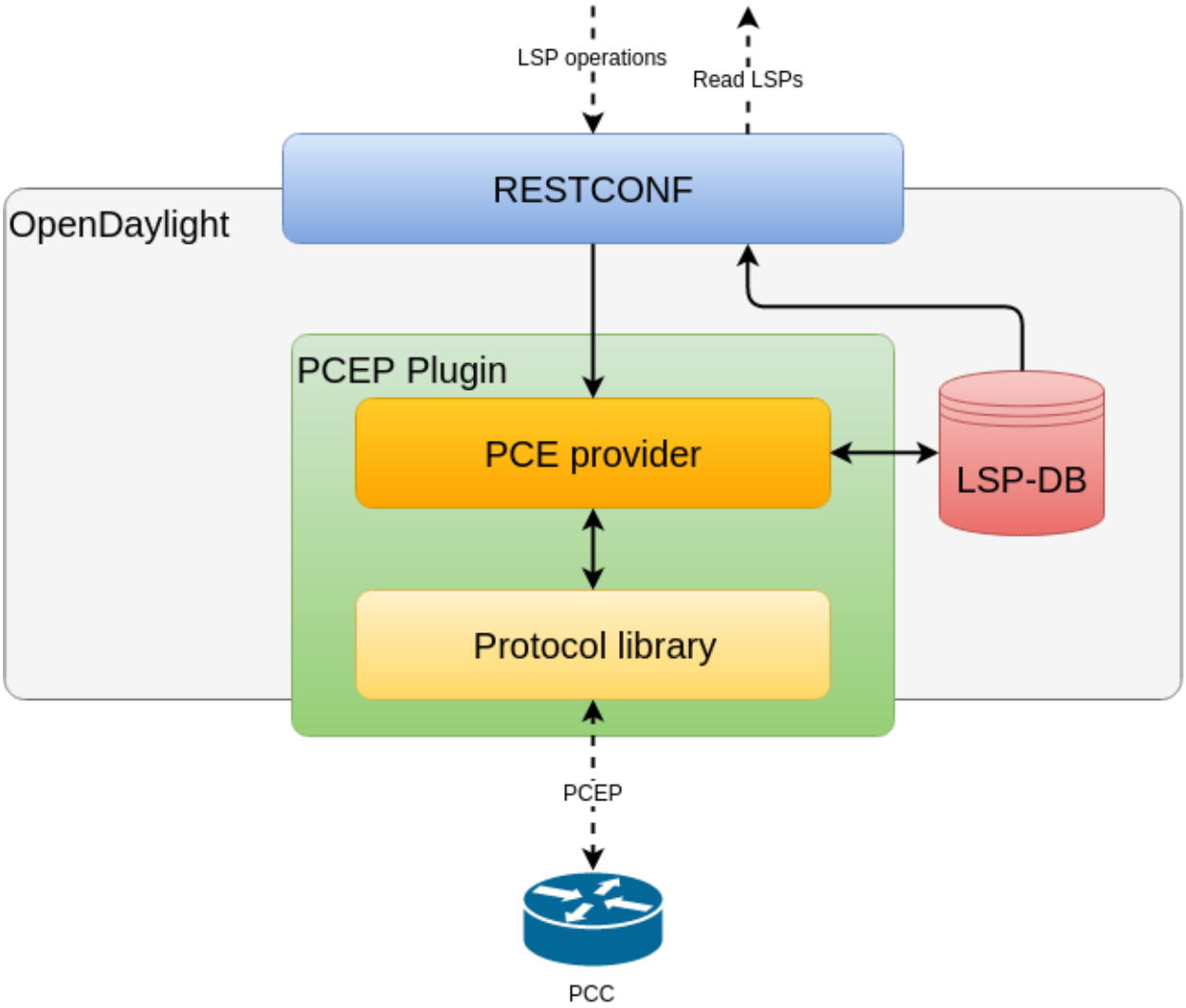


Fig. 1.83: OpenDaylight PCEP plugin overview.

Important: The PCEP plugin does not provide path computational functionality and does not build TED.

List of supported capabilities

- RFC5440 - Path Computation Element (PCE) Communication Protocol (PCEP)
- RFC5455 - Diffserv-Aware Class-Type Object for the Path Computation Element Communication Protocol

- RFC5520 - Preserving Topology Confidentiality in Inter-Domain Path Computation Using a Path-Key-Based Mechanism
- RFC5521 - Extensions to the Path Computation Element Communication Protocol (PCEP) for Route Exclusions
- RFC5541 - Encoding of Objective Functions in the Path Computation Element Communication Protocol (PCEP)
- RFC5557 - Path Computation Element Communication Protocol (PCEP) Requirements and Protocol Extensions in Support of Global Concurrent Optimization
- RFC5886 - A Set of Monitoring Tools for Path Computation Element (PCE)-Based Architecture
- RFC7470 - Conveying Vendor-Specific Constraints in the Path Computation Element Communication Protocol
- RFC7896 - Update to the Include Route Object (IRO) Specification in the Path Computation Element Communication Protocol (PCEP)
- draft-ietf-pce-stateful-pce - PCEP Extensions for Stateful PCE
  - draft-ietf-pce-pce-initiated-lsp - PCEP Extensions for PCE-initiated LSP Setup in a Stateful PCE Model
  - draft-ietf-pce-segment-routing - PCEP Extension for segment routing
  - draft-ietf-pce-lsp-setup-type - PCEP Extension for path setup type
  - draft-ietf-pce-stateful-sync-optimizations - Optimizations of Label Switched Path State Synchronization Procedures for a Stateful PCE
  - draft-sivabalan-pce-binding-label-sid - Carrying Binding Label/Segment-ID in PCE-based Networks
- draft-ietf-pce-pceps - Secure Transport for PCEP

Running PCEP

This section explains how to install PCEP plugin.

1. Install PCEP feature - `odl-bgpcep-pcep`. Also, for sake of this sample, it is required to install RESTCONF. In the Karaf console, type command:

```
feature:install odl-restconf odl-bgpcep-pcep
```

2. The PCEP plugin contains a default configuration, which is applied after the feature starts up. One instance of PCEP plugin is created (named *pcep-topology*), and its presence can be verified via REST:

   URL: `restconf/operational/network-topology:network-topology/topology/pcep-topology`

   Method: `GET`

   Response Body:

```
<topology xmlns="urn:TBD:params:xml:ns:yang:network-topology">
    <topology-id>pcep-topology</topology-id>
    <topology-types>
        <topology-pcep xmlns="urn:opendaylight:params:xml:ns:yang:topology:pcep">
↪</topology-pcep>
    </topology-types>
</topology>
```

Active Stateful PCE

The PCEP extension for Stateful PCE brings a visibility of active LSPs to PCE, in order to optimize path computation, while considering individual LSPs and their interactions. This requires state synchronization mechanism between PCE and PCC. Moreover, Active Stateful PCE is capable to address LSP parameter changes to the PCC.

Contents

Configuration

This capability is enabled by default. No additional configuration is required.

MD5 authentication configuration

The OpenDaylight PCEP implementation is supporting TCP MD5 for authentication. Sample configuration below shows how to set authentication password for a particular PCC. It is required to install `odl-netconf-connector-ssh` feature first.

URL: `/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules/module/`

`odl-pcep-topology-provider-cfg:pcep-topology-provider/pcep-topology`

Method: `PUT`

Content-Type: `application/xml`

Request Body:

```
1  <module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
2      <type xmlns:x=
   ↪"urn:opendaylight:params:xml:ns:yang:controller:pcep:topology:provider">x:pcep-
   ↪topology-provider</type>
3      <name>pcep-topology</name>
4      <data-provider xmlns=
   ↪"urn:opendaylight:params:xml:ns:yang:controller:pcep:topology:provider">
5          <type xmlns:x="urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding
   ↪">x:binding-async-data-broker</type>
6          <name>pingpong-binding-data-broker</name>
7      </data-provider>
8      <dispatcher xmlns=
   ↪"urn:opendaylight:params:xml:ns:yang:controller:pcep:topology:provider">
9          <type xmlns:x="urn:opendaylight:params:xml:ns:yang:controller:pcep">x:pcep-
   ↪dispatcher</type>
10         <name>global-pcep-dispatcher</name>
11     </dispatcher>
12     <rpc-registry xmlns=
   ↪"urn:opendaylight:params:xml:ns:yang:controller:pcep:topology:provider">
13         <type xmlns:x="urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding
   ↪">x:binding-rpc-registry</type>
14         <name>binding-rpc-broker</name>
15     </rpc-registry>
16     <scheduler xmlns=
   ↪"urn:opendaylight:params:xml:ns:yang:controller:pcep:topology:provider">
17         <type xmlns:x="urn:opendaylight:params:xml:ns:yang:controller:programming:spi
   ↪">x:instruction-scheduler</type>
18         <name>global-instruction-scheduler</name>
19     </scheduler>
20     <stateful-plugin xmlns=
   ↪"urn:opendaylight:params:xml:ns:yang:controller:pcep:topology:provider">
21         <type>pcep-topology-stateful</type>
22         <name>stateful07</name>
23     </stateful-plugin>
24     <topology-id xmlns=
   ↪"urn:opendaylight:params:xml:ns:yang:controller:pcep:topology:provider">pcep-
   ↪topology</topology-id>
25     <client xmlns=
   ↪"urn:opendaylight:params:xml:ns:yang:controller:pcep:topology:provider">
26         <address>43.43.43.43</address>
27         <password>topsecret</password>
28     </client>
29 </module>
```

@line 26: **address** - A PCC IP address.

@line 27: **password** - MD5 authentication phrase.

Warning: The PCE (*pcep-topology-provider*) configuration is going to be changed in Carbon release - moving to configuration datastore.

LSP State Database

The *LSP State Database* (LSP-DB) contains an information about all LSPs and their attributes. The LSP state is synchronized between the PCC and PCE. First, initial LSP state synchronization is performed once the session between PCC and PCE is established in order to learn PCC's LPSs. This step is a prerequisite to following LSPs manipulation operations.

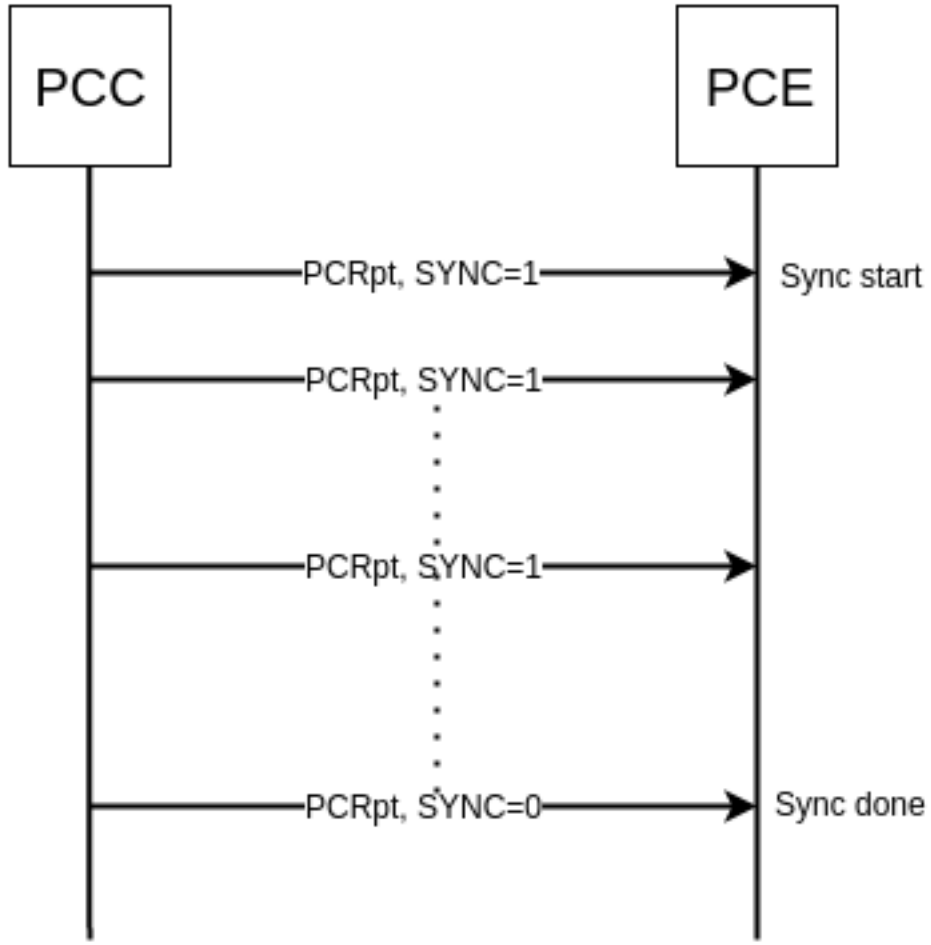


Fig. 1.84: LSP State Synchronization.

LSP-DB API

```
path-computation-client
   +--ro reported-lsp* [name]
      +--ro name         string
      +--ro path* [lsp-id]
      |  +--ro lsp-id                      rsvp:lsp-id
      |  +--ro ero
      |  |  +--ro processing-rule?   boolean
      |  |  +--ro ignore?            boolean
      |  |  +--ro subobject*
      |  |     +--ro loose         boolean
```

```
      |  |     +--ro (subobject-type)?
      |  |        +--:(as-number-case)
      |  |        |  +--ro as-number
      |  |        |     +--ro as-number    inet:as-number
      |  |        +--:(ip-prefix-case)
      |  |        |  +--ro ip-prefix
      |  |        |     +--ro ip-prefix    inet:ip-prefix
      |  |        +--:(label-case)
      |  |        |  +--ro label
      |  |        |     +--ro uni-directional             boolean
      |  |        |     +--ro (label-type)?
      |  |        |        +--:(type1-label-case)
      |  |        |        |  +--ro type1-label
      |  |        |        |     +--ro type1-label    uint32
      |  |        |        +--:(generalized-label-case)
      |  |        |        |  +--ro generalized-label
      |  |        |        |     +--ro generalized-label    binary
      |  |        |        +--:(waveband-switching-label-case)
      |  |        |           +--ro waveband-switching-label
      |  |        |              +--ro end-label      uint32
      |  |        |              +--ro start-label    uint32
      |  |        |              +--ro waveband-id    uint32
      |  |        +--:(srlg-case)
      |  |        |  +--ro srlg
      |  |        |     +--ro srlg-id    srlg-id
      |  |        +--:(unnumbered-case)
      |  |        |  +--ro unnumbered
      |  |        |     +--ro router-id       uint32
      |  |        |     +--ro interface-id    uint32
      |  |        +--:(exrs-case)
      |  |        |  +--ro exrs
      |  |        |     +--ro exrs*
      |  |        |        +--ro mandatory?    boolean
      |  |        |        +--ro attribute     enumeration
      |  |        |        +--ro (subobject-type)?
      |  |        |           +--:(as-number-case)
      |  |        |           |  +--ro as-number
      |  |        |           |     +--ro as-number    inet:as-number
      |  |        |           +--:(ip-prefix-case)
      |  |        |           |  +--ro ip-prefix
      |  |        |           |     +--ro ip-prefix    inet:ip-prefix
      |  |        |           +--:(label-case)
      |  |        |           |  +--ro label
      |  |        |           |     +--ro uni-directional             boolean
      |  |        |           |     +--ro (label-type)?
      |  |        |           |        +--:(type1-label-case)
      |  |        |           |        |  +--ro type1-label
      |  |        |           |        |     +--ro type1-label    uint32
      |  |        |           |        +--:(generalized-label-case)
      |  |        |           |        |  +--ro generalized-label
      |  |        |           |        |     +--ro generalized-label    binary
      |  |        |           |        +--:(waveband-switching-label-case)
      |  |        |           |           +--ro waveband-switching-label
      |  |        |           |              +--ro end-label      uint32
      |  |        |           |              +--ro start-label    uint32
      |  |        |           |              +--ro waveband-id    uint32
      |  |        |           +--:(srlg-case)
      |  |        |           |  +--ro srlg
```

```
|  |        |           |     +--ro srlg-id    srlg-id
|  |        |           +--:(unnumbered-case)
|  |        |              +--ro unnumbered
|  |        |                 +--ro router-id       uint32
|  |        |                 +--ro interface-id    uint32
|  |        +--:(path-key-case)
|  |           +--ro path-key
|  |              +--ro pce-id      pce-id
|  |              +--ro path-key    path-key
|  +--ro lspa
|  |  +--ro processing-rule?            boolean
|  |  +--ro ignore?                     boolean
|  |  +--ro hold-priority?              uint8
|  |  +--ro setup-priority?             uint8
|  |  +--ro local-protection-desired?   boolean
|  |  +--ro label-recording-desired?    boolean
|  |  +--ro se-style-desired?           boolean
|  |  +--ro session-name?               string
|  |  +--ro include-any?                attribute-filter
|  |  +--ro exclude-any?                attribute-filter
|  |  +--ro include-all?                attribute-filter
|  |  +--ro tlvs
|  |     +--ro vendor-information-tlv*
|  |        +--ro enterprise-number?   iana:enterprise-number
|  |        +--ro (enterprise-specific-information)?
|  +--ro bandwidth
|  |  +--ro processing-rule?   boolean
|  |  +--ro ignore?            boolean
|  |  +--ro bandwidth?         netc:bandwidth
|  +--ro reoptimization-bandwidth
|  |  +--ro processing-rule?   boolean
|  |  +--ro ignore?            boolean
|  |  +--ro bandwidth?         netc:bandwidth
|  +--ro metrics*
|  |  +--ro metric
|  |     +--ro processing-rule?   boolean
|  |     +--ro ignore?            boolean
|  |     +--ro metric-type        uint8
|  |     +--ro bound?             boolean
|  |     +--ro computed?          boolean
|  |     +--ro value?             ieee754:float32
|  +--ro iro
|  |  +--ro processing-rule?   boolean
|  |  +--ro ignore?            boolean
|  |  +--ro subobject*
|  |     +--ro loose         boolean
|  |     +--ro (subobject-type)?
|  |        +--:(as-number-case)
|  |        |  +--ro as-number
|  |        |     +--ro as-number    inet:as-number
|  |        +--:(ip-prefix-case)
|  |        |  +--ro ip-prefix
|  |        |     +--ro ip-prefix    inet:ip-prefix
|  |        +--:(label-case)
|  |        |  +--ro label
|  |        |     +--ro uni-directional            boolean
|  |        |     +--ro (label-type)?
|  |        |        +--:(type1-label-case)
```

```
|  |  |        |        |  +--ro type1-label
|  |  |        |        |     +--ro type1-label    uint32
|  |  |        |        +--:(generalized-label-case)
|  |  |        |        |  +--ro generalized-label
|  |  |        |        |     +--ro generalized-label    binary
|  |  |        |        +--:(waveband-switching-label-case)
|  |  |        |           +--ro waveband-switching-label
|  |  |        |              +--ro end-label      uint32
|  |  |        |              +--ro start-label    uint32
|  |  |        |              +--ro waveband-id    uint32
|  |  |        +--:(srlg-case)
|  |  |        |  +--ro srlg
|  |  |        |     +--ro srlg-id    srlg-id
|  |  |        +--:(unnumbered-case)
|  |  |        |  +--ro unnumbered
|  |  |        |     +--ro router-id       uint32
|  |  |        |     +--ro interface-id    uint32
|  |  |        +--:(exrs-case)
|  |  |        |  +--ro exrs
|  |  |        |     +--ro exrs*
|  |  |        |        +--ro mandatory?    boolean
|  |  |        |        +--ro attribute     enumeration
|  |  |        |        +--ro (subobject-type)?
|  |  |        |           +--:(as-number-case)
|  |  |        |           |  +--ro as-number
|  |  |        |           |     +--ro as-number    inet:as-number
|  |  |        |           +--:(ip-prefix-case)
|  |  |        |           |  +--ro ip-prefix
|  |  |        |           |     +--ro ip-prefix    inet:ip-prefix
|  |  |        |           +--:(label-case)
|  |  |        |           |  +--ro label
|  |  |        |           |     +--ro uni-directional            boolean
|  |  |        |           |     +--ro (label-type)?
|  |  |        |           |        +--:(type1-label-case)
|  |  |        |           |        |  +--ro type1-label
|  |  |        |           |        |     +--ro type1-label    uint32
|  |  |        |           |        +--:(generalized-label-case)
|  |  |        |           |        |  +--ro generalized-label
|  |  |        |           |        |     +--ro generalized-label    binary
|  |  |        |           |        +--:(waveband-switching-label-case)
|  |  |        |           |           +--ro waveband-switching-label
|  |  |        |           |              +--ro end-label      uint32
|  |  |        |           |              +--ro start-label    uint32
|  |  |        |           |              +--ro waveband-id    uint32
|  |  |        |           +--:(srlg-case)
|  |  |        |           |  +--ro srlg
|  |  |        |           |     +--ro srlg-id    srlg-id
|  |  |        |           +--:(unnumbered-case)
|  |  |        |              +--ro unnumbered
|  |  |        |                 +--ro router-id       uint32
|  |  |        |                 +--ro interface-id    uint32
|  |  |        +--:(path-key-case)
|  |  |           +--ro path-key
|  |  |              +--ro pce-id      pce-id
|  |  |              +--ro path-key    path-key
|  +--ro rro
|  |  +--ro processing-rule?   boolean
|  |  +--ro ignore?            boolean
```

```
      |  |  +--ro subobject*
      |  |     +--ro protection-available?   boolean
      |  |     +--ro protection-in-use?      boolean
      |  |     +--ro (subobject-type)?
      |  |        +--:(ip-prefix-case)
      |  |        |  +--ro ip-prefix
      |  |        |     +--ro ip-prefix    inet:ip-prefix
      |  |        +--:(label-case)
      |  |        |  +--ro label
      |  |        |     +--ro uni-directional             boolean
      |  |        |     +--ro (label-type)?
      |  |        |     |  +--:(type1-label-case)
      |  |        |     |  |  +--ro type1-label
      |  |        |     |  |     +--ro type1-label    uint32
      |  |        |     |  +--:(generalized-label-case)
      |  |        |     |  |  +--ro generalized-label
      |  |        |     |  |     +--ro generalized-label    binary
      |  |        |     |  +--:(waveband-switching-label-case)
      |  |        |     |     +--ro waveband-switching-label
      |  |        |     |        +--ro end-label      uint32
      |  |        |     |        +--ro start-label    uint32
      |  |        |     |        +--ro waveband-id    uint32
      |  |        |     +--ro global?                       boolean
      |  |        +--:(unnumbered-case)
      |  |        |  +--ro unnumbered
      |  |        |     +--ro router-id       uint32
      |  |        |     +--ro interface-id    uint32
      |  |        +--:(path-key-case)
      |  |           +--ro path-key
      |  |              +--ro pce-id      pce-id
      |  |              +--ro path-key    path-key
      |  +--ro xro
      |  |  +--ro processing-rule?   boolean
      |  |  +--ro ignore?            boolean
      |  |  +--ro flags              bits
      |  |  +--ro subobject*
      |  |     +--ro mandatory?    boolean
      |  |     +--ro attribute     enumeration
      |  |     +--ro (subobject-type)?
      |  |        +--:(as-number-case)
      |  |        |  +--ro as-number
      |  |        |     +--ro as-number    inet:as-number
      |  |        +--:(ip-prefix-case)
      |  |        |  +--ro ip-prefix
      |  |        |     +--ro ip-prefix    inet:ip-prefix
      |  |        +--:(label-case)
      |  |        |  +--ro label
      |  |        |     +--ro uni-directional             boolean
      |  |        |     +--ro (label-type)?
      |  |        |        +--:(type1-label-case)
      |  |        |        |  +--ro type1-label
      |  |        |        |     +--ro type1-label    uint32
      |  |        |        +--:(generalized-label-case)
      |  |        |        |  +--ro generalized-label
      |  |        |        |     +--ro generalized-label    binary
      |  |        |        +--:(waveband-switching-label-case)
      |  |        |           +--ro waveband-switching-label
      |  |        |              +--ro end-label      uint32
```

```
      |  |        |              +--ro start-label    uint32
      |  |        |              +--ro waveband-id    uint32
      |  |        +--:(srlg-case)
      |  |        |  +--ro srlg
      |  |        |     +--ro srlg-id    srlg-id
      |  |        +--:(unnumbered-case)
      |  |           +--ro unnumbered
      |  |              +--ro router-id       uint32
      |  |              +--ro interface-id    uint32
      |  +--ro of
      |  |  +--ro processing-rule?   boolean
      |  |  +--ro ignore?            boolean
      |  |  +--ro code               of-id
      |  |  +--ro tlvs
      |  |     +--ro vendor-information-tlv*
      |  |        +--ro enterprise-number?   iana:enterprise-number
      |  |        +--ro (enterprise-specific-information)?
      |  +--ro class-type
      |     +--ro processing-rule?   boolean
      |     +--ro ignore?            boolean
      |     +--ro class-type         class-type
      +--ro metadata
      +--ro lsp
      |  +--ro processing-rule?   boolean
      |  +--ro ignore?            boolean
      |  +--ro tlvs
      |  |  +--ro lsp-error-code
      |  |  |  +--ro error-code?   uint32
      |  |  +--ro lsp-identifiers
      |  |  |  +--ro lsp-id?      rsvp:lsp-id
      |  |  |  +--ro tunnel-id?   rsvp:tunnel-id
      |  |  |  +--ro (address-family)?
      |  |  |     +--:(ipv4-case)
      |  |  |     |  +--ro ipv4
      |  |  |     |     +--ro ipv4-tunnel-sender-address      inet:ipv4-address
      |  |  |     |     +--ro ipv4-extended-tunnel-id         rsvp:ipv4-extended-
↪tunnel-id
      |  |  |     |     +--ro ipv4-tunnel-endpoint-address    inet:ipv4-address
      |  |  |     +--:(ipv6-case)
      |  |  |        +--ro ipv6
      |  |  |           +--ro ipv6-tunnel-sender-address      inet:ipv6-address
      |  |  |           +--ro ipv6-extended-tunnel-id         rsvp:ipv6-extended-
↪tunnel-id
      |  |  |           +--ro ipv6-tunnel-endpoint-address    inet:ipv6-address
      |  |  +--ro rsvp-error-spec
      |  |  |  +--ro (error-type)?
      |  |  |     +--:(rsvp-case)
      |  |  |     |  +--ro rsvp-error
      |  |  |     +--:(user-case)
      |  |  |        +--ro user-error
      |  |  +--ro symbolic-path-name
      |  |  |  +--ro path-name?   symbolic-path-name
      |  |  o--ro vs-tlv
      |  |  |  +--ro enterprise-number?   iana:enterprise-number
      |  |  |  +--ro (vendor-payload)?
      |  |  +--ro vendor-information-tlv*
      |  |  |  +--ro enterprise-number?   iana:enterprise-number
      |  |  |  +--ro (enterprise-specific-information)?
```

```
      |  |  +--ro path-binding
      |  |     x--ro binding-type?      uint8
      |  |     x--ro binding-value?     binary
      |  |     +--ro (binding-type-value)?
      |  |        +--:(mpls-label)
      |  |        |  +--ro mpls-label?        netc:mpls-label
      |  |        +--:(mpls-label-entry)
      |  |           +--ro label?            netc:mpls-label
      |  |           +--ro traffic-class?    uint8
      |  |           +--ro bottom-of-stack?  boolean
      |  |           +--ro time-to-live?     uint8
      |  +--ro plsp-id?           plsp-id
      |  +--ro delegate?          boolean
      |  +--ro sync?              boolean
      |  +--ro remove?            boolean
      |  +--ro administrative?    boolean
      |  +--ro operational?       operational-status
      +--ro path-setup-type
         +--ro pst?   uint8
```

The LSP-DB is accessible via RESTCONF. The PCC's LSPs are stored in the `pcep-topology` while the session is active. In a next example, there is one PCEP session with PCC identified by its IP address (*43.43.43.43*) and one reported LSP (*foo*).

URL: `/restconf/operational/network-topology:network-topology/topology/pcep-topology/node/pcc:%2F%2F43.43.43.43`

Method: `GET`

Response Body:

```
<node>
   <node-id>pcc://43.43.43.43</node-id>
   <path-computation-client>
      <ip-address>43.43.43.43</ip-address>
      <state-sync>synchronized</state-sync>
      <stateful-tlv>
         <stateful>
            <lsp-update-capability>true</lsp-update-capability>
         </stateful>
      </stateful-tlv>
      <reported-lsp>
         <name>foo</name>
         <lsp>
            <operational>up</operational>
            <sync>true</sync>
            <plsp-id>1</plsp-id>
            <create>false</create>
            <administrative>true</administrative>
            <remove>false</remove>
            <delegate>true</delegate>
            <tlvs>
               <lsp-identifiers>
                  <ipv4>
                     <ipv4-tunnel-sender-address>43.43.43.43</ipv4-tunnel-sender-
↪address>
                     <ipv4-tunnel-endpoint-address>39.39.39.39</ipv4-tunnel-endpoint-
↪address>
```

```
26                      <ipv4-extended-tunnel-id>39.39.39.39</ipv4-extended-tunnel-id>
27                   </ipv4>
28                   <tunnel-id>1</tunnel-id>
29                   <lsp-id>1</lsp-id>
30                </lsp-identifiers>
31                <symbolic-path-name>
32                   <path-name>Zm9v</path-name>
33                </symbolic-path-name>
34             </tlvs>
35          </lsp>
36          <ero>
37             <subobject>
38                <loose>false</loose>
39                <ip-prefix>
40                   <ip-prefix>201.20.160.40/32</ip-prefix>
41                </ip-prefix>
42             </subobject>
43             <subobject>
44                <loose>false</loose>
45                <ip-prefix>
46                   <ip-prefix>195.20.160.39/32</ip-prefix>
47                </ip-prefix>
48             </subobject>
49             <subobject>
50                <loose>false</loose>
51                <ip-prefix>
52                   <ip-prefix>39.39.39.39/32</ip-prefix>
53                </ip-prefix>
54             </subobject>
55          </ero>
56       </reported-lsp>
57    </path-computation-client>
58 </node>
```

@line 2: **node-id** The PCC identifier.

@line 4: **ip-address** IP address of the PCC.

@line 5: **state-sync** Synchronization status of the PCC's LSPs. The *synchronized* indicates the State Synchronization is done.

@line 8: **lsp-update-capability** - Indicates that PCC allows LSP modifications.

@line 12: **name** - Textual representation of LPS's name.

@line 14: **operational** - Represent operational status of the LSP:

- *down* - not active.
- *up* - signaled.
- *active* - up and carrying traffic.
- *going-down* - LSP is being torn down, resources are being released.
- *going-up* - LSP is being signaled.

@line 15: **sync** - The flag set by PCC during LSPs State Synchronization.

@line 16: **plsp-id** - A PCEP-specific identifier for the LSP. It is assigned by PCC and it is constant for a lifetime of a PCEP session.

@line 17: **create** - The *false* indicates that LSP is PCC-initiated.

@line 18: **administrative** - The flag indicates target operational status of the LSP.

@line 20: **delegate** - The delegate flag indicates that the PCC is delegating the LSP to the PCE.

@line 24: **ipv4-tunnel-sender-address** - Contains the sender node's IP address.

@line 25: **ipv4-tunnel-endpoint-address** - Contains the egress node's IP address.

@line 26: **ipv4-extended-tunnel-id** - The *Extended Tunnel ID* identifier.

@line 28: **tunnel-id** - The *Tunnel ID* identifier.

@line 29: **lsp-id** - The *LSP ID* identifier.

@line 32: **path-name** - The symbolic name for the LSP.

@line 36: **ero** - The *Explicit Route Object* is encoding the path of the TE LSP through the network.

LSP Delegation

The LSP control delegations is an mechanism, where PCC grants to a PCE the temporary right in order to modify LSP attributes. The PCC can revoke the delegation or the PCE may waive the delegation at any time. The LSP control is delegated to at most one PCE at the same time.

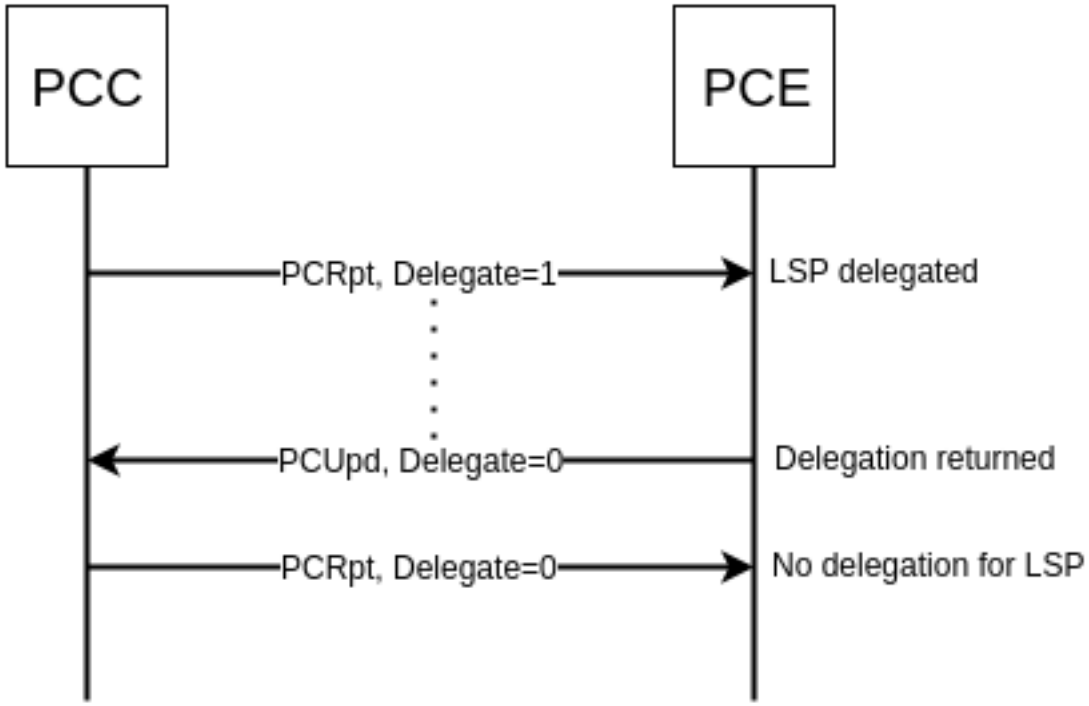


Fig. 1.85: Returning a Delegation.

Following RPC example illustrates a request for the LSP delegation give up:

URL: `/restconf/operations/network-topology-pcep:update-lsp`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<input>
   <node>pcc://43.43.43.43</node>
   <name>foo</name>
   <arguments>
      <lsp xmlns:stateful="urn:opendaylight:params:xml:ns:yang:pcep:ietf:stateful">
         <delegate>false</delegate>
         <administrative>true</administrative>
         <tlvs>
            <symbolic-path-name>
               <path-name>Zm9v</path-name>
            </symbolic-path-name>
         </tlvs>
      </lsp>
   </arguments>
   <network-topology-ref xmlns:topo="urn:TBD:params:xml:ns:yang:network-topology">/
↪topo:network-topology/topo:topology[topo:topology-id="pcep-topology"]</network-
↪topology-ref>
</input>
```

@line 2: **node** The PCC identifier.

@line 3: **name** The name of the LSP.

@line 6: **delegate** - Delegation flag set *false* in order to return the LSP delegation.

@line 10: **path-name** - The Symbolic Path Name TLV must be present when sending a request to give up the delegation.

LSP Update

The LSP Update Request is an operation where a PCE requests a PCC to update attributes of an LSP and to rebuild the LSP with updated attributes. In order to update LSP, the PCE must hold a LSP delegation. The LSP update is done in *make-before-break* fashion - first, new LSP is initiated and then the old LSP is torn down.

Following RPC example shows a request for the LSP update:

URL: `/restconf/operations/network-topology-pcep:update-lsp`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<input xmlns="urn:opendaylight:params:xml:ns:yang:topology:pcep">
   <node>pcc://43.43.43.43</node>
   <name>foo</name>
   <arguments>
      <lsp xmlns="urn:opendaylight:params:xml:ns:yang:pcep:ietf:stateful">
         <delegate>true</delegate>
         <administrative>true</administrative>
      </lsp>
      <ero>
         <subobject>
            <loose>false</loose>
            <ip-prefix>
```

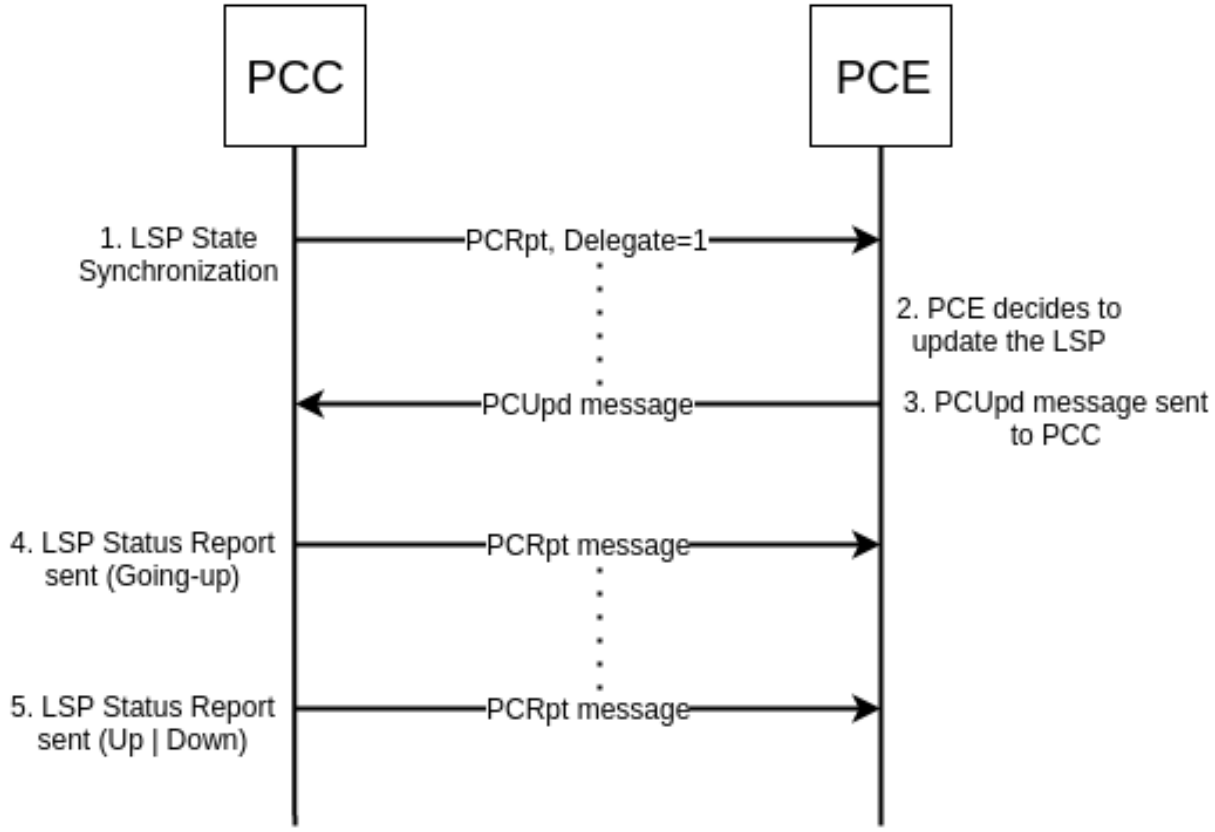


Fig. 1.86: Active Stateful PCE LSP Update.

```
13                <ip-prefix>200.20.160.41/32</ip-prefix>
14             </ip-prefix>
15          </subobject>
16          <subobject>
17             <loose>false</loose>
18             <ip-prefix>
19                <ip-prefix>196.20.160.39/32</ip-prefix>
20             </ip-prefix>
21          </subobject>
22          <subobject>
23             <loose>false</loose>
24             <ip-prefix>
25                <ip-prefix>39.39.39.39/32</ip-prefix>
26             </ip-prefix>
27          </subobject>
28       </ero>
29    </arguments>
30    <network-topology-ref xmlns:topo="urn:TBD:params:xml:ns:yang:network-topology">/
 ↪topo:network-topology/topo:topology[topo:topology-id="pcep-topology"]</network-
 ↪topology-ref>
31 </input>
```

@line 2: **node** The PCC identifier.

@line 3: **name** The name of the LSP to be updated.

@line 6: **delegate** - Delegation flag set *true* in order to keep the LSP control.

@line 7: **administrative** - Desired administrative status of the LSP is active.

@line 9: **ero** - This LSP attribute is changed.

PCE-initiated LSP Setup

The PCEP Extension for PCE-initiated LSP Setup allows PCE to request a creation and deletion of LSPs.

Configuration

This capability is enabled by default. No additional configuration is required.

LSP Instantiation

The PCE can request LSP creation. The LSP instantiation is done by sending an LSP Initiate Message to PCC. The PCC assign delegation to PCE which triggered creation. PCE-initiated LSPs are identified by *Create* flag.

Following RPC example shows a request for the LSP initiation:

URL: `/restconf/operations/network-topology-pcep:add-lsp`

Method: `POST`

Content-Type: `application/xml`

Request Body:

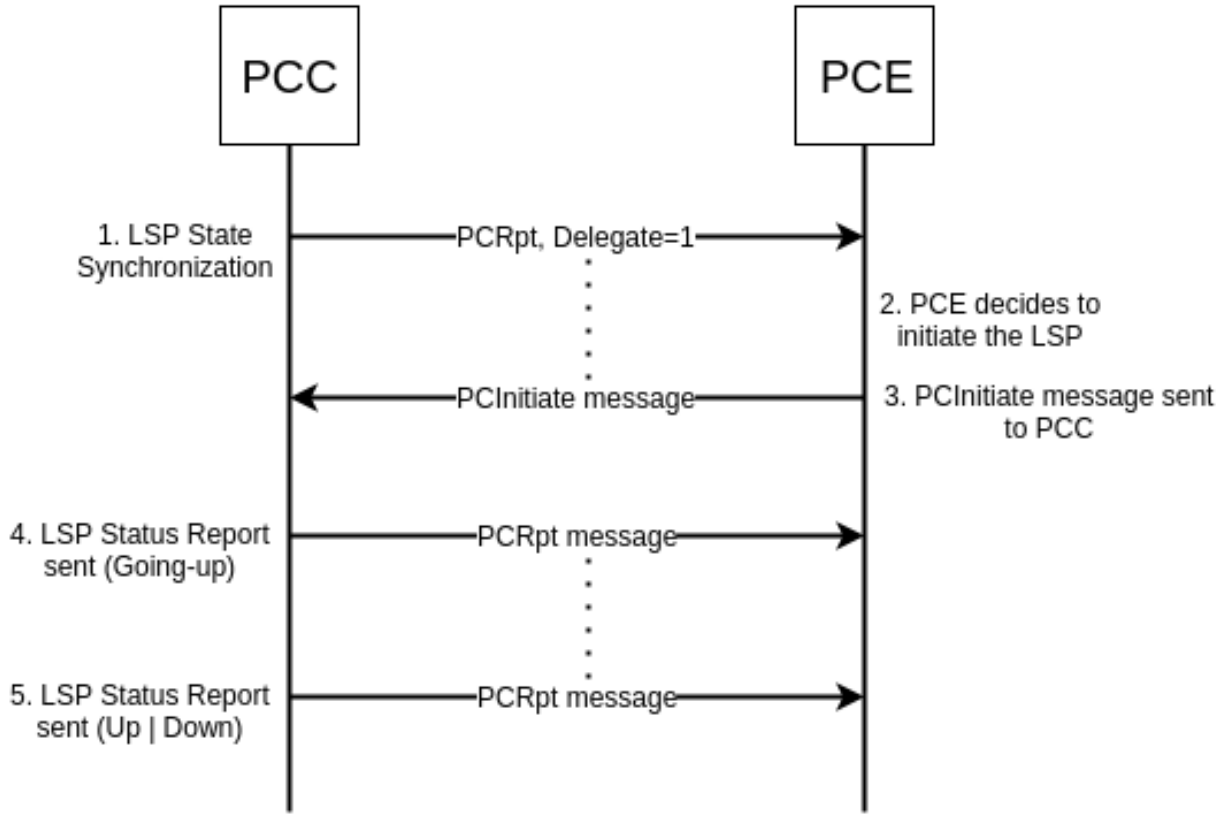


Fig. 1.87: LSP instantiation.

```
<input xmlns="urn:opendaylight:params:xml:ns:yang:topology:pcep">
   <node>pcc://43.43.43.43</node>
   <name>update-tunel</name>
      <arguments>
         <lsp xmlns="urn:opendaylight:params:xml:ns:yang:pcep:ietf:stateful">
            <delegate>true</delegate>
            <administrative>true</administrative>
         </lsp>
         <endpoints-obj>
            <ipv4>
               <source-ipv4-address>43.43.43.43</source-ipv4-address>
               <destination-ipv4-address>39.39.39.39</destination-ipv4-address>
            </ipv4>
         </endpoints-obj>
         <ero>
            <subobject>
               <loose>false</loose>
               <ip-prefix>
                  <ip-prefix>201.20.160.40/32</ip-prefix>
               </ip-prefix>
            </subobject>
            <subobject>
               <loose>false</loose>
               <ip-prefix>
                  <ip-prefix>195.20.160.39/32</ip-prefix>
               </ip-prefix>
            </subobject>
            <subobject>
               <loose>false</loose>
               <ip-prefix>
                  <ip-prefix>39.39.39.39/32</ip-prefix>
               </ip-prefix>
            </subobject>
         </ero>
      </arguments>
   <network-topology-ref xmlns:topo="urn:TBD:params:xml:ns:yang:network-topology">/
↪topo:network-topology/topo:topology[topo:topology-id="pcep-topology"]</network-
↪topology-ref>
</input>
```

@line 2: **node** The PCC identifier.

@line 3: **name** The name of the LSP to be created.

@line 8: **endpoints-obj** - The *END-POINT* Object is mandatory for an instantiation request of an RSVP-signaled LSP. It contains source and destination addresses for provisioning the LSP.

@line 14: **ero** - The *ERO* object is mandatory for LSP initiation request.

LSP Deletion

The PCE may request a deletion of PCE-initiated LSPs. The PCE must be delegation holder for this particular LSP.

Following RPC example shows a request for the LSP deletion:

URL: `/restconf/operations/network-topology-pcep:remove-lsp`

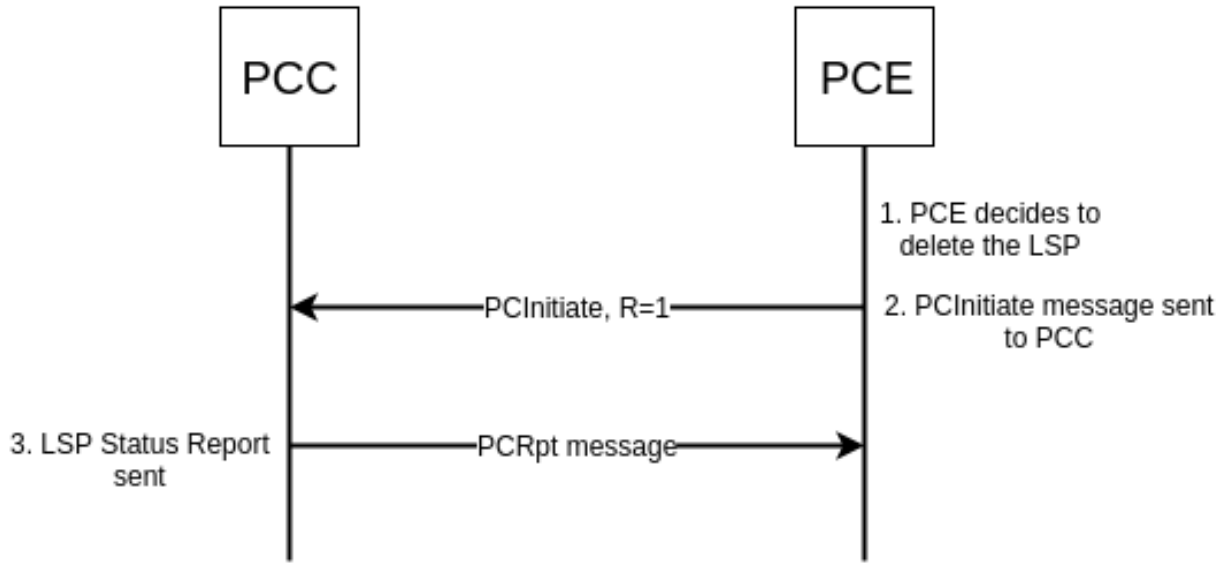


Fig. 1.88: LSP deletion.

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<input xmlns="urn:opendaylight:params:xml:ns:yang:topology:pcep">
   <node>pcc://43.43.43.43</node>
   <name>update-tunel</name>
   <network-topology-ref xmlns:topo="urn:TBD:params:xml:ns:yang:network-topology">/
↪topo:network-topology/topo:topology[topo:topology-id="pcep-topology"]</network-
↪topology-ref>
</input>
```

@line 2: **node** The PCC identifier.

@line 3: **name** The name of the LSP to be removed.

PCE-initiated LSP Delegation

The PCE-initiated LSP control is delegated to the PCE which requested the initiation. The PCC cannot revoke delegation of PCE-initiated LSP. When PCE returns delegation for such LSP or PCE fails, then the LSP become orphan and can be removed by a PCC after some time. The PCE may ask for a delegation of the orphan LSP.

Following RPC example illustrates a request for the LSP delegation:

URL: `/restconf/operations/network-topology-pcep:update-lsp`

Method: `POST`

Content-Type: `application/xml`

Request Body:

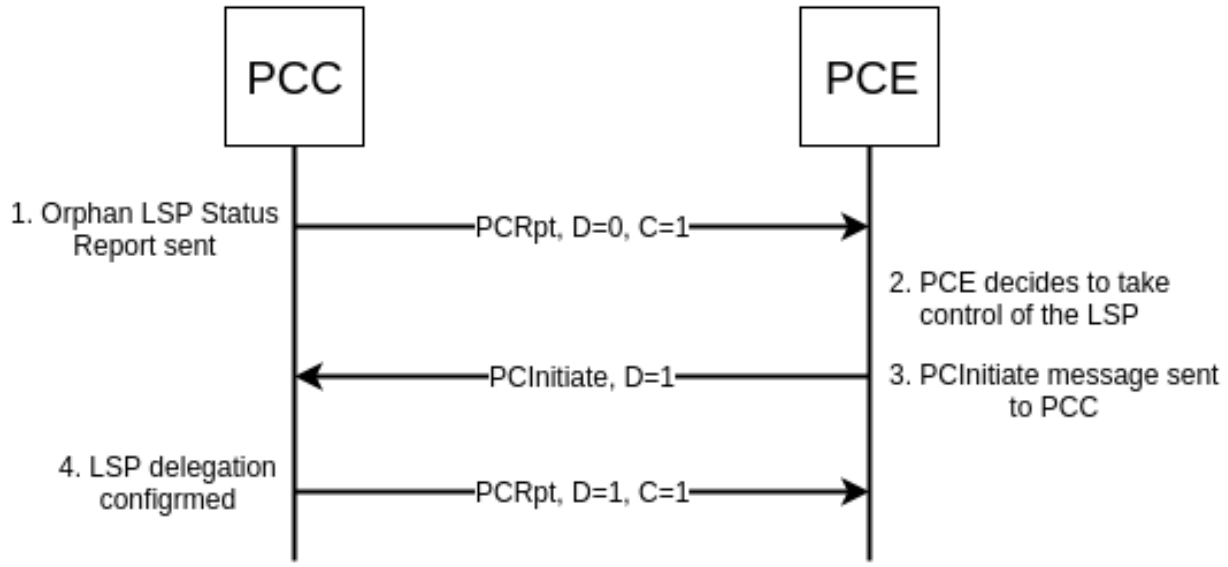


Fig. 1.89: Orphan PCE-initiated LSP - control taken by PCE.

```
<input>
   <node>pcc://43.43.43.43</node>
   <name>update-tunel</name>
   <arguments>
      <lsp xmlns:stateful="urn:opendaylight:params:xml:ns:yang:pcep:ietf:stateful">
         <delegate>true</delegate>
         <administrative>true</administrative>
         <tlvs>
            <symbolic-path-name>
               <path-name>dXBkYXRlLXR1bmVs</path-name>
            </symbolic-path-name>
         </tlvs>
      </lsp>
   </arguments>
   <network-topology-ref xmlns:topo="urn:TBD:params:xml:ns:yang:network-topology">/
↪topo:network-topology/topo:topology[topo:topology-id="pcep-topology"]</network-
↪topology-ref>
</input>
```

@line 2: **node** The PCC identifier.

@line 3: **name** The name of the LSP.

@line 6: **delegate** - *Delegation* flag set *true* in order to take the LSP delegation.

@line 10: **path-name** - The *Symbolic Path Name* TLV must be present when sending a request to take a delegation.

Segment Routing

The PCEP Extensions for Segment Routing (SR) allow a stateful PCE to compute and initiate TE paths in SR networks. The SR path is defined as an order list of *segments*. Segment Routing architecture can be directly applied to the MPLS forwarding plane without changes. Segment Identifier (SID) is encoded as a MPLS label.

Configuration

This capability is enabled by default. In PCEP-SR draft version 6, SR Explicit Route Object/Record Route Object subobjects IANA code points change was proposed. In order to use the latest code points, a configuration should be changed in following way:

URL: `/restconf/config/pcep-segment-routing-app-config:pcep-segment-routing-app-config`

Method: `PUT`

Content-Type: `application/xml`

Request Body:

```
<pcep-segment-routing-config xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:pcep:segment-routing-app-config">
   <iana-sr-subobjects-type>true</iana-sr-subobjects-type>
</pcep-segment-routing-config>
```

LSP Operations for PCEP SR

The PCEP SR extension defines new ERO subobject - *SR-ERO subobject* capable of carrying a SID.

```
sr-ero-type
   +---- c-flag?                boolean
   +---- m-flag?                boolean
   +---- sid-type?              sid-type
   +---- sid?                   uint32
   +---- (nai)?
      +--:(ip-node-id)
      |  +---- ip-address              inet:ip-address
      +--:(ip-adjacency)
      |  +---- local-ip-address       inet:ip-address
      |  +---- remote-ip-address      inet:ip-address
      +--:(unnumbered-adjacency)
         +---- local-node-id          uint32
         +---- local-interface-id     uint32
         +---- remote-node-id         uint32
         +---- remote-interface-id    uint32
```

Following RPC example illustrates a request for the SR-TE LSP creation:

URL: `/restconf/operations/network-topology-pcep:add-lsp`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<input xmlns="urn:opendaylight:params:xml:ns:yang:topology:pcep">
   <node>pcc://43.43.43.43</node>
   <name>sr-path</name>
   <arguments>
      <lsp xmlns="urn:opendaylight:params:xml:ns:yang:pcep:ietf:stateful">
         <delegate>true</delegate>
         <administrative>true</administrative>
```

```
      </lsp>
      <endpoints-obj>
         <ipv4>
            <source-ipv4-address>43.43.43.43</source-ipv4-address>
            <destination-ipv4-address>39.39.39.39</destination-ipv4-address>
         </ipv4>
      </endpoints-obj>
      <path-setup-type xmlns="urn:opendaylight:params:xml:ns:yang:pcep:ietf:stateful">
         <pst>1</pst>
      </path-setup-type>
      <ero>
         <subobject>
            <loose>false</loose>
            <sid-type xmlns="urn:opendaylight:params:xml:ns:yang:pcep:segment:routing
↪">ipv4-node-id</sid-type>
            <m-flag xmlns="urn:opendaylight:params:xml:ns:yang:pcep:segment:routing">
↪true</m-flag>
            <sid xmlns="urn:opendaylight:params:xml:ns:yang:pcep:segment:routing">
↪24001</sid>
            <ip-address xmlns=
↪"urn:opendaylight:params:xml:ns:yang:pcep:segment:routing">39.39.39.39</ip-address>
        </subobject>
      </ero>
   </arguments>
   <network-topology-ref xmlns:topo="urn:TBD:params:xml:ns:yang:network-topology">/
↪topo:network-topology/topo:topology[topo:topology-id="pcep-topology"]</network-
↪topology-ref>
</input>
```

@line 16: **path-setup-type** - Set *1* for SR-TE LSP

@line 21: **ipv4-node-id** - The SR-ERO subobject represents *IPv4 Node ID* NAI.

@line 22: **m-flag** - The SID value represents an MPLS label.

@line 23: **sid** - The Segment Identifier.

Following RPC example illustrates a request for the SR-TE LSP update including modified path:

URL: `/restconf/operations/network-topology-pcep:update-lsp`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<input xmlns="urn:opendaylight:params:xml:ns:yang:topology:pcep">
   <node>pcc://43.43.43.43</node>
   <name>update-tunnel</name>
   <arguments>
      <lsp xmlns="urn:opendaylight:params:xml:ns:yang:pcep:ietf:stateful">
         <delegate>true</delegate>
         <administrative>true</administrative>
      </lsp>
      <path-setup-type xmlns="urn:opendaylight:params:xml:ns:yang:pcep:ietf:stateful">
         <pst>1</pst>
      </path-setup-type>
      <ero>
```

```
        <subobject>
           <loose>false</loose>
           <sid-type xmlns="urn:opendaylight:params:xml:ns:yang:pcep:segment:routing
↪">ipv4-node-id</sid-type>
           <m-flag xmlns="urn:opendaylight:params:xml:ns:yang:pcep:segment:routing">
↪true</m-flag>
           <sid xmlns="urn:opendaylight:params:xml:ns:yang:pcep:segment:routing">
↪24002</sid>
           <ip-address xmlns=
↪"urn:opendaylight:params:xml:ns:yang:pcep:segment:routing">200.20.160.41</ip-
↪address>
        </subobject>
        <subobject>
           <loose>false</loose>
           <sid-type xmlns="urn:opendaylight:params:xml:ns:yang:pcep:segment:routing
↪">ipv4-node-id</sid-type>
           <m-flag xmlns="urn:opendaylight:params:xml:ns:yang:pcep:segment:routing">
↪true</m-flag>
           <sid xmlns="urn:opendaylight:params:xml:ns:yang:pcep:segment:routing">
↪24001</sid>
           <ip-address xmlns=
↪"urn:opendaylight:params:xml:ns:yang:pcep:segment:routing">39.39.39.39</ip-address>
        </subobject>
     </ero>
  </arguments>
  <network-topology-ref xmlns:topo="urn:TBD:params:xml:ns:yang:network-topology">/
↪topo:network-topology/topo:topology[topo:topology-id="pcep-topology"]</network-
↪topology-ref>
</input>
```

LSP State Synchronization Optimization Procedures

This extension bring optimizations for state synchronization:

- State Synchronization Avoidance
- Incremental State Synchronization
- PCE-triggered Initial Synchronization
- PCE-triggered Re-synchronization

Configuration

This capability is enabled by default. No additional configuration is required.

State Synchronization Avoidance

The State Synchronization Avoidance procedure is intended to skip state synchronization if the state has survived and not changed during session restart.

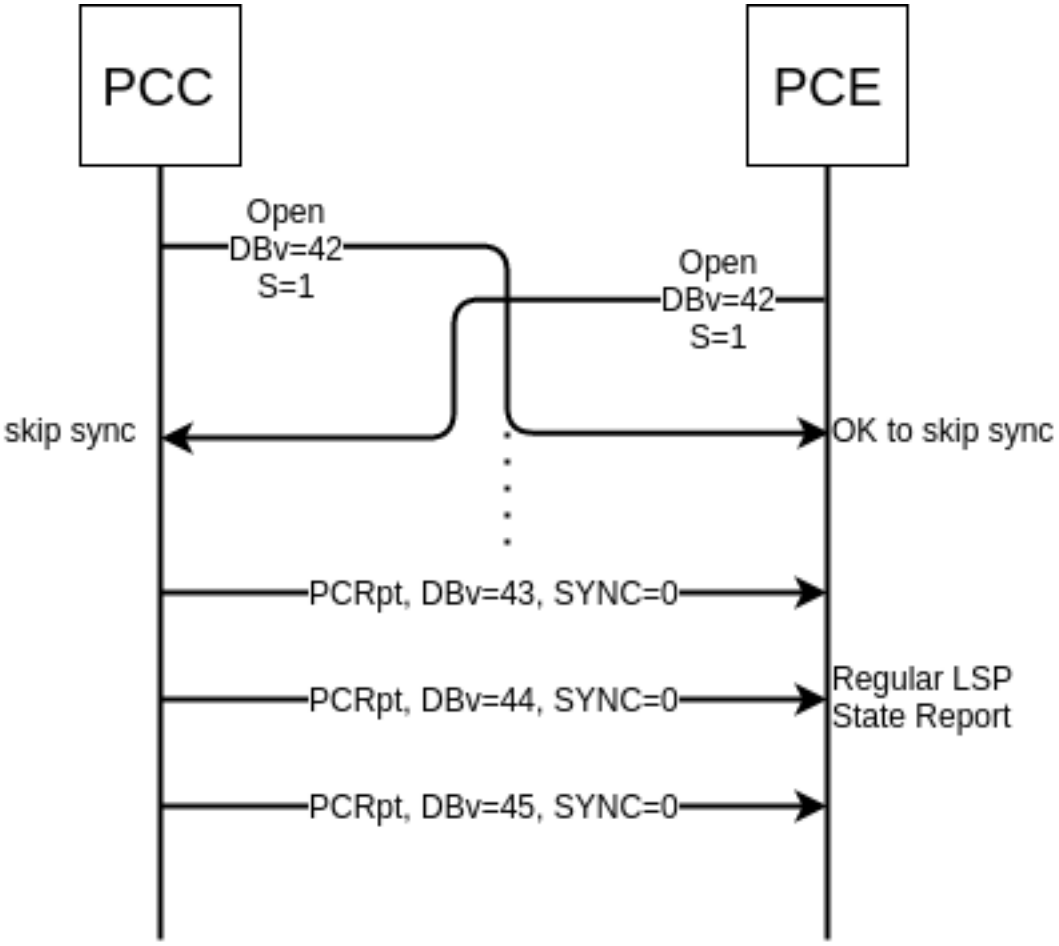


Fig. 1.90: State Synchronization Skipped.

Incremental State Synchronization

The Incremental State Synchronization procedure is intended to do incremental (delta) state synchronization when possible.

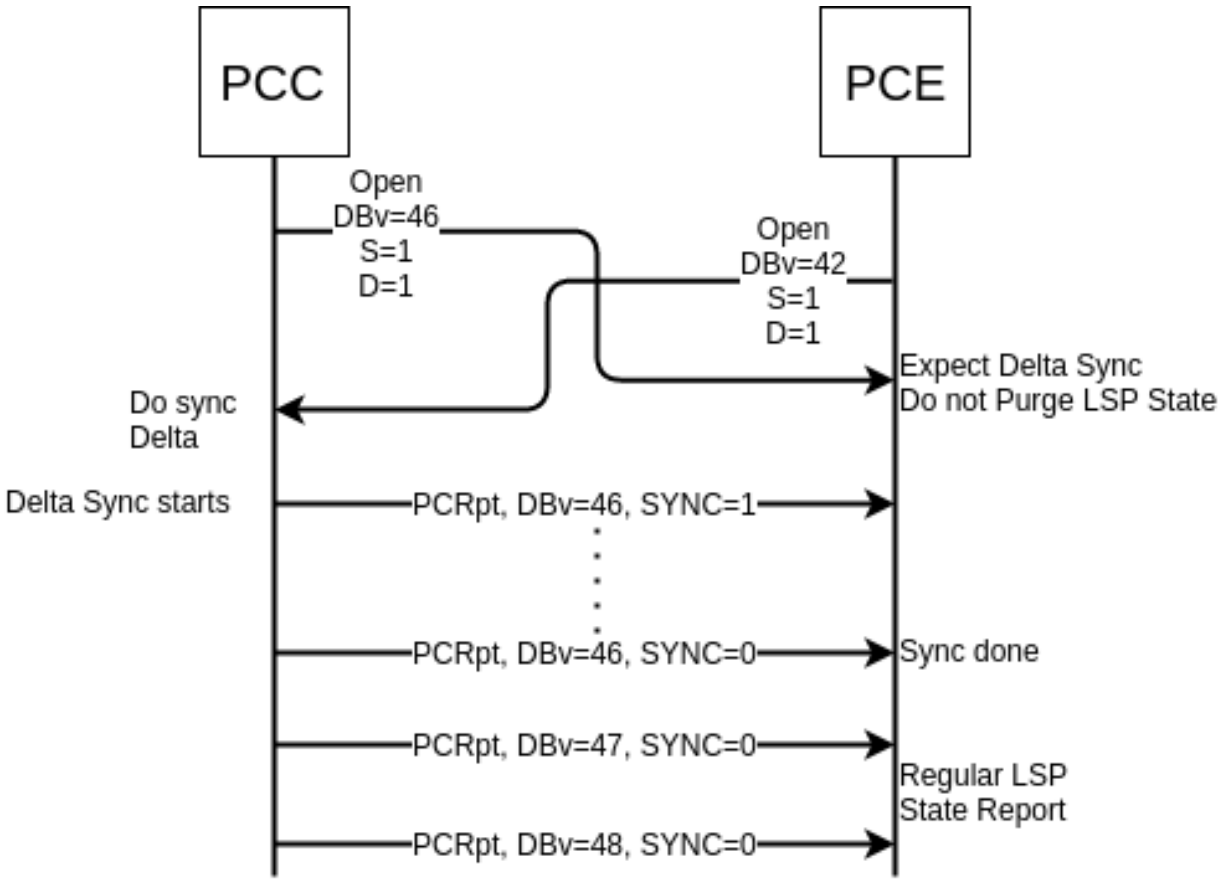


Fig. 1.91: Incremental Synchronization Procedure.

PCE-triggered Initial Synchronization

The PCE-triggered Initial Synchronization procedure is intended to do let PCE control the timing of the initial state synchronization.

Following RPC example illustrates a request for the initial synchronization:

URL: `/restconf/operations/network-topology-pcep:trigger-sync`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
<input xmlns="urn:opendaylight:params:xml:ns:yang:topology:pcep">
   <node>pcc://43.43.43.43</node>
```

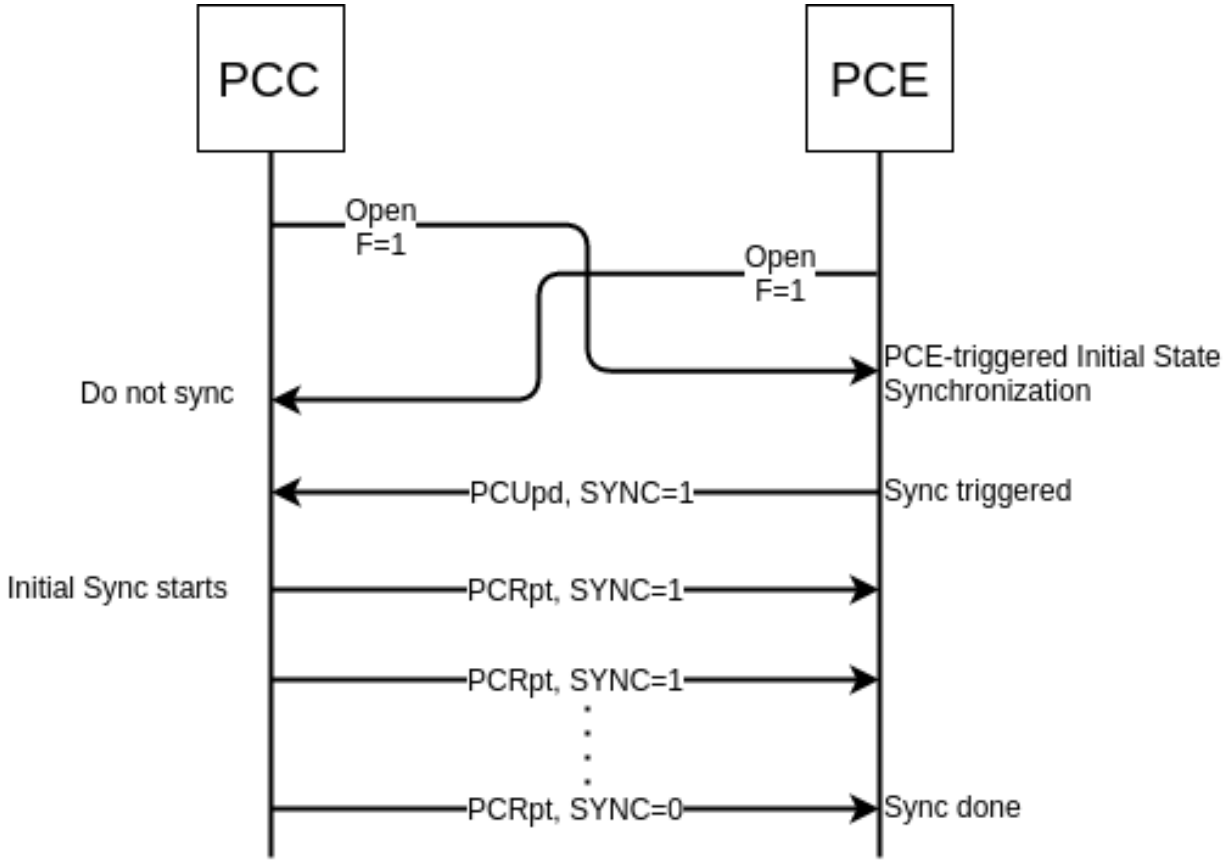


Fig. 1.92: PCE-triggered Initial State Synchronization Procedure.

```
    <network-topology-ref xmlns:topo="urn:TBD:params:xml:ns:yang:network-topology">/
↪topo:network-topology/topo:topology[topo:topology-id="pcep-topology"]</network-
↪topology-ref>
</input>
```

PCE-triggered Re-synchronization

The PCE-triggered Re-synchronization: To let PCE re-synchronize the state for sanity check.

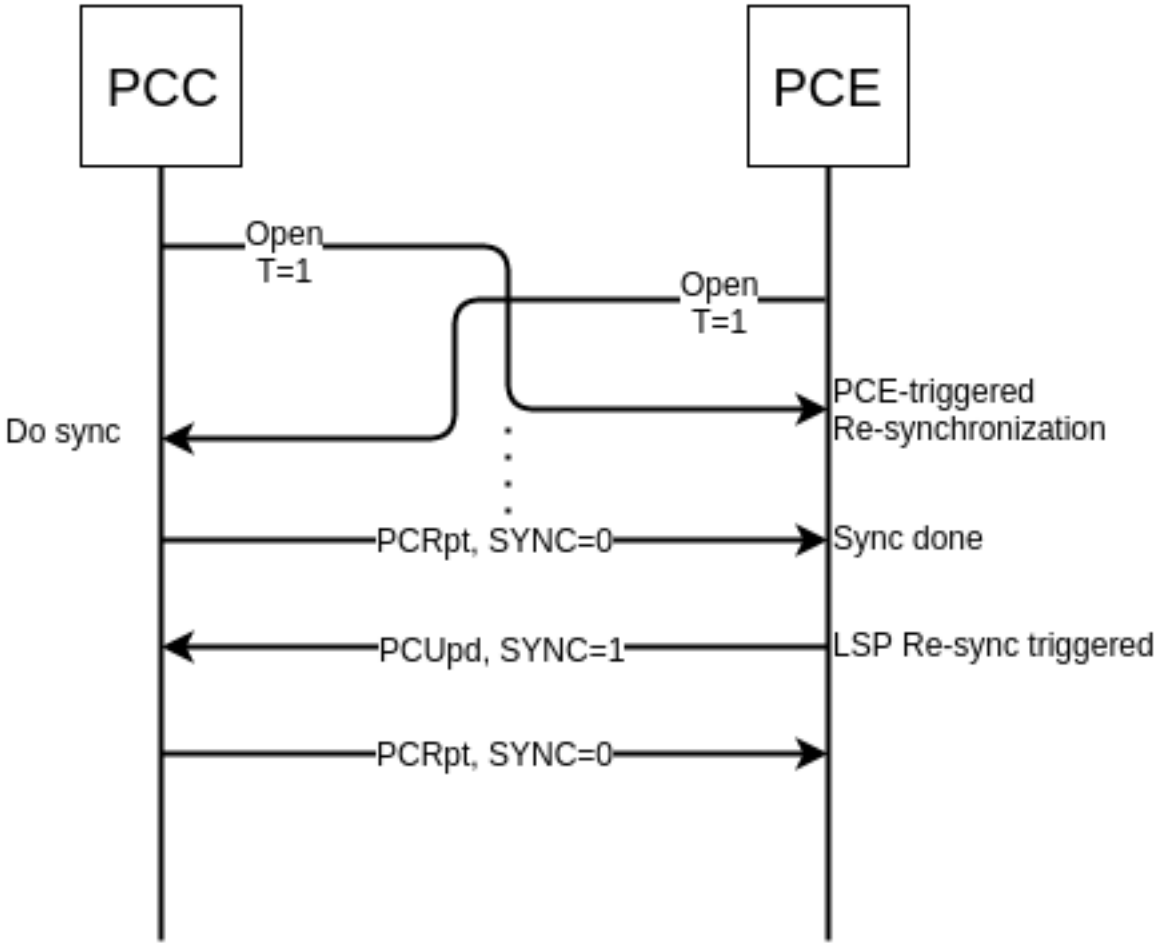


Fig. 1.93: PCE-triggered Re-synchronization Procedure.

Following RPC example illustrates a request for the LSP re-synchronization:

URL: `/restconf/operations/network-topology-pcep:trigger-sync`

Method: `POST`

Content-Type: `application/xml`

Request Body:

```
1 <input xmlns="urn:opendaylight:params:xml:ns:yang:topology:pcep">
2    <node>pcc://43.43.43.43</node>
3    <name>update-lsp</name>
4    <network-topology-ref xmlns:topo="urn:TBD:params:xml:ns:yang:network-topology">/
  ↪topo:network-topology/topo:topology[topo:topology-id="pcep-topology"]</network-
  ↪topology-ref>
5 </input>
```

@line 3: **name** - The LSP name. If this parameter is omitted, re-synchronization is requested for all PCC's LSPs.

Test tools

PCC Mock

The PCC Mock is a stand-alone Java application purposed to simulate a PCC(s). The simulator is capable to report sample LSPs, respond to delegation, LSP management operations and synchronization optimization procedures. This application is not part of the OpenDaylight Karaf distribution, however it can be downloaded from OpenDaylight's Nexus (use latest release version):

`https://nexus.opendaylight.org/content/repositories/opendaylight.release/org/opendaylight/bgpcep/pcep-pcc-mock`

Usage

The application can be run from command line:

```
java -jar pcep-pcc-mock-*-executable.jar
```

with optional input parameters:

```
--local-address <Address:Port> (optional, default 127.0.0.1)
   The first PCC IP address. If more PCCs are required, the IP address will be␣
  ↪incremented. Port number can be optionally specified.

--remote-address <Address1:Port1,Address2:Port2,Address3:Port3,...> (optional,␣
  ↪default 127.0.0.1:4189)
   The list of IP address for the PCE servers. Port number can be optionally␣
  ↪specified, otherwise default port number 4189 is used.

--pcc <N> (optional, default 1)
   Number of mocked PCC instances.

--lsp <N> (optional, default 1)
   Number of tunnels (LSPs) reported per PCC, might be zero.

--pcerr (optional flag)
   If the flag is present, response with PCErr, otherwise PCUpd.

--log-level <LEVEL> (optional, default INFO)
   Set logging level for pcc-mock.

-d, --deadtimer <0..255> (optional, default 120)
   DeadTimer value in seconds.
```

```
-ka, --keepalive <0.255> (optional, default 30)
   KeepAlive timer value in seconds.

--password <password> (optional)
   If the password is present, it is used in TCP MD5 signature, otherwise plain TCP␣
↪is used.

--reconnect <seconds> (optional)
   If the argument is present, the value in seconds, is used as a delay before each␣
↪new reconnect (initial connect or connection re-establishment) attempt.
   The number of reconnect attempts is unlimited. If the argument is omitted, pcc-
↪mock is not trying to reconnect.

--redelegation-timeout <seconds> (optional, default 0)
   The timeout starts when LSP delegation is returned or PCE fails, stops when LSP is␣
↪re-delegated to PCE.
   When timeout expires, LSP delegation is revoked and held by PCC.

--state-timeout <seconds> (optional, default -1 (disabled))
   The timeout starts when LSP delegation is returned or PCE fails, stops when LSP is␣
↪re-delegated to PCE.
   When timeout expires, PCE-initiated LSP is removed.

--state-sync-avoidance <disconnect_after_x_seconds> <reconnect_after_x_seconds>
↪<dbVersion>
   Synchronization avoidance capability enabled.
      - disconnect_after_x_seconds: seconds that will pass until disconnections is␣
↪forced. If set to smaller number than 1, disconnection wont be performed.
      - reconnect_after_x_seconds: seconds that will pass between disconnection and␣
↪new connection attempt. Only happens if disconnection has been performed.
      - dbVersion: dbVersion used in new Open and must be always equal or bigger than␣
↪LSP. If equal than LSP skip synchronization will be performed,
        if not full synchronization will be performed taking in account new starting␣
↪dbVersion desired.
 --incremental-sync-procedure <disconnect_after_x_seconds> <reconnect_after_x_seconds>
↪ <dbVersion>
   Incremental synchronization capability enabled.
      - dbVersion: dbVersion used in new Open and must be always bigger than LSP.␣
↪Incremental synchronization will be performed taking in account new starting␣
↪dbVersion desired.

 --triggered-initial-sync
   PCE-triggered synchronization capability enabled. Can be combined combined with␣
↪state-sync-avoidance/incremental-sync-procedure.

 --triggered-re-sync
   PCE-triggered re-synchronization capability enabled.
```

Data Change Counter Tool

Data Change Counter tool registers a Data Change Listener to a specified topology's subtree. This will allow us to know the quantity of changes produced under it, with each data change event counter will be incremented.

Installation

Installing data change counter tool

```
feature:install odl-restconf odl-bgpcep-data-change-counter
```

Configuration

Once we set the configuration, a new data change counter will be created and registers to example-linkstate-topology.

Important: Clustering - Each Counter Identifier should be unique.

URL: `/restconf/config/odl-data-change-counter-config:data-change-counter-config/data-change-counter`

Method: `PUT`

Content-Type: `application/xml`

Request Body:

```
1 <data-change-counter-config xmlns="urn:opendaylight:params:xml:ns:yang:bgpcep:data-
  ↪change-counter-config">
2     <counter-id>data-change-counter</counter-id>
3     <topology-name>example-linkstate-topology</topology-name>
4 </data-change-counter-config>
```

@line 2: **Counter Id** - Unique counter change identifier.

@line 3: **Topology Name** - An identifier for a topology.

Usage

Counter state for topology

URL: `/restconf/operational/data-change-counter:data-change-counter/counter/data-change-counter`

Method: `GET`

Response Body:

```
1 <counter xmlns="urn:opendaylight:params:xml:ns:yang:bgp-data-change-counter">
2     <id>data-change-counter</id>
3     <count>0</count>
4 </counter>
```

@line 2: **Counter Id** - Unique counter change identifier.

@line 3: **Count** - Number of changes under registered topology's subtree.

Troubleshooting

This section offers advices in a case OpenDaylight PCEP plugin is not working as expected.

Contents

PCEP is not working...

- First of all, ensure that all required features are installed, local PCE and remote PCC configuration is correct.

  To list all installed features in OpenDaylight use the following command at the Karaf console:

```
feature:list -i
```

- Check OpenDaylight Karaf logs:

  From Karaf console:

```
log:tail
```

  or open log file: `data/log/karaf.log`

  Possibly, a reason/hint for a cause of the problem can be found there.

- Try to minimize effect of other OpenDaylight features, when searching for a reason of the problem.
- Try to set DEBUG severity level for PCEP logger via Karaf console commands, in order to collect more information:

```
log:set DEBUG org.opendaylight.protocol.pcep
```

```
log:set DEBUG org.opendaylight.bgpcep.pcep
```

Bug reporting

Before you report a bug, check BGPCEP Jira to ensure same/similar bug is not already filed there.

Write an e-mail to bgpcep-users@lists.opendaylight.org and provide following information:

1. State OpenDaylight version
2. Describe your use-case and provide as much details related to PCEP as possible
3. Steps to reproduce
4. Attach Karaf log files, optionally packet captures, REST input/output

References

- A Path Computation Element (PCE)-Based Architecture
- Path Computation Element (PCE) Communication Protocol Generic Requirements
- Unanswered Questions in the Path Computation Element Architecture
- A PCE-Based Architecture for Application-Based Network Operations

- Framework for PCE-Based Inter-Layer MPLS and GMPLS Traffic Engineering
- Applicability of a Stateful Path Computation Element (PCE)

PacketCable User Guide

Overview

These components introduce a DOCSIS QoS Gates management using the PCMM protocol. The driver component is responsible for the PCMM/COPS/PDP functionality required to service requests from PacketCable Provider and FlowManager. Requests are transposed into PCMM Gate Control messages and transmitted via COPS to the CMTS. This plugin adheres to the PCMM/COPS/PDP functionality defined in the CableLabs specification. PacketCable solution is an MDSAL compliant component.

PacketCable Components

PacketCable is comprised of two OpenDaylight bundles:

Bundle	Description
odl-packetcable-policy-server	Plugin that provides PCMM model implementation based on CMTS structure and COPS protocol.
odl-packetcable-policy-model	The Model provided provides a direct mapping to the underlying QoS Gates of CMTS.

See the PacketCable YANG Models.

Installing PacketCable

To install PacketCable, run the following `feature:install` command from the Karaf CLI

```
feature:install odl-packetcable-policy-server-all odl-restconf odl-mdsal-apidocs
```

Explore and exercise the PacketCable REST API

To see the PacketCable APIs, browse to this URL: http://localhost:8181/apidoc/explorer/index.html

Replace localhost with the IP address or hostname where OpenDaylight is running if you are not running OpenDaylight locally on your machine.

Note: Prior to setting any PCMM gates, a CCAP must first be added.

Postman

Install the Chrome extension

Download and import sample packetcable collection

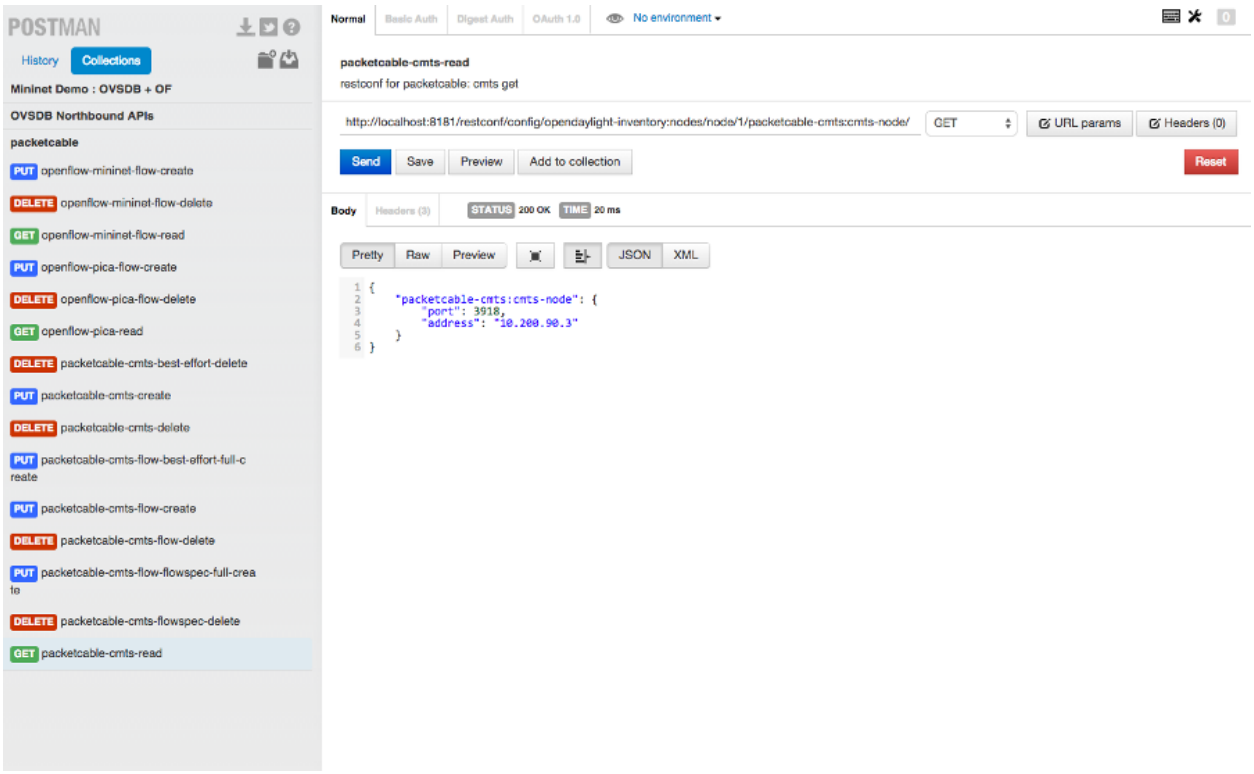


Fig. 1.94: Postman Operations

Postman Operations

PacketCable REST API Usage Examples

- CCAP "CONFIG" DATASTORE API STRUCTURE
  - Add and view CCAPConfigDatastore(add triggers also the CCAP COPS connection):

```
PUT http://localhost:8181/restconf/config/packetcable:ccaps/ccap/CMTS-1

{"ccap":[
   {"ccapId":"CMTS-1",
    "amId": {
          "am-tag": 51930,
          "am-type": 1
    },
    "connection": {
          "ipAddress": "10.20.30.40",
          "port":3918
    },"subscriber-subnets": [
          "2001:4978:030d:1000:0:0:0:0/52",
          "44.137.0.0/16"
    ],"upstream-scns": [
          "SCNA",
          "extrm_up"
    ],"downstream-scns": [
          "extrm_dn",
          "ipvideo_dn",
```

```
          "SCNC"
    ]}
]}

GET http://localhost:8181/restconf/config/packetcable:ccaps/ccap/CMTS-1
```

- CCAP OPERATIONAL STATUS - GET CCAP (COPS) CONNECTION STATUS
  - Shows the Operational Datastorecontents for the CCAP COPS connection.
  - The status is updated when the COPS connection is initiated or after an RPC poll:

```
GET http://localhost:8181/restconf/operational/packetcable:ccaps/ccap/CMTS-1/
Response: 200 OK

{
  "ccap": [
        {
             "ccapId": "CMTS-1",
             "connection": {
                  "error": [
                        "E6-CTO: CCAP client is connected"
                  ],
                  "timestamp": "2016-03-23T14:15:54.129-05:00",
                  "connected": true
             }
        }
    ]
}
```

- CCAP OPERATIONAL STATUS - RPC CCAP POLL CONNECTION
  - A CCAP RPC poll returns the COPS connectivity status info and also triggers an Operational Datastore status update with the same data:

```
POST http://localhost:8181/restconf/operations/packetcable:ccap-poll-
↪connection
{
     "input": {
           "ccapId": "/packetcable:ccaps/packetcable:ccap[packetcable:ccapId=
↪'CMTS-1']"
     }
}
Response: 200 OK
{
"output": {
      "response": "CMTS-1: CCAP poll complete",
      "timestamp": "2016-03-23T14:15:54.131-05:00",
      "ccap": {
            "ccapId": "CMTS-1",
            "connection": {
                  "connection": {
                        "error": [
                              "CMTS-1: CCAP client is connected"
                        ],
                        "timestamp": "2016-03-23T14:15:54.129-05:00",
                        "connected": true
                  }
             }
```

```
        }
    }
}
```

- CCAP OPERATIONAL STATUS - RPC CCAP POLL CONNECTION (2) - CONNECTION DOWN:

```
POST http://localhost:8181/restconf/operations/packetcable:ccap-poll-connection
{
     "input": {
           "ccapId": "/packetcable:ccaps/packetcable:ccap[packetcable:ccapId=
↪'CMTS-1']"
     }
}
Response: 200 OK
{
"output": {
      "response": "CMTS-1: CCAP poll complete",
      "timestamp": "2016-03-23T14:15:54.131-05:00",
      "ccap": {
            "ccapId": "CMTS-1",
            "connection": {
                  "error": [
                        "CMTS-1: CCAP client is disconnected with error: null",
                        "CMTS-1: CCAP Cops socket is closed"],
                  "timestamp": "2016-03-23T14:15:54.129-05:00",
                  "connected": false
            }
        }
    }
}
```

- CCAP OPERATIONAL STATUS - RPC CCAP SET CONNECTION
  - A CCAP RPC sets the CCAP COPS connection; possible values true or false meaning that the connection should be up or down.
  - RPC responds with the same info as RPC POLL CONNECTION, and also updates the Operational Datastore:

```
POST http://localhost:8181/restconf/operations/packetcable:ccap-set-connection
{
     "input": {
           "ccapId": "/packetcable:ccaps/packetcable:ccap[packetcable:ccapId=
↪'CMTS-1']",
            "connection": {
                  "connected": true
           }
     }
}
Response: 200 OK
{
       "output": {

              "response": "CMTS-1: CCAP set complete",
              "timestamp": "2016-03-23T17:47:29.446-05:00",
              "ccap": {
                     "ccapId": "CMTS-1",
                     "connection": {
```

```
                                "error": [
                                        "CMTS-1: CCAP client is connected",
                                        "CMTS-1: CCAP COPS socket is already open
↪"],
                                "timestamp": "2016-03-23T17:47:29.436-05:00",
                                "connected": true
                        }
                }
        }
}
```

- CCAP OPERATIONAL STATUS - RPC CCAP SET CONNECTION (2) - SHUTDOWN COPS CONNECTION:

```
POST http://localhost:8181/restconf/operations/packetcable:ccap-set-connection
{
     "input": {
           "ccapId": "/packetcable:ccaps/packetcable:ccap[packetcable:ccapId='E6-
↪CTO']",
            "connection": {
                  "connected": false
           }
     }
}
Response: 200 OK
{
       "output": {
              "response": "E6-CTO: CCAP set complete",
              "timestamp": "2016-03-23T17:47:29.446-05:00",
              "ccap": {
                     "ccapId": "E6-CTO",
                     "connection": {
                            "error": [
                                    "E60CTO: CCAP client is disconnected with␣
↪error: null"],
                            "timestamp": "2016-03-23T17:47:29.436-05:00",
                            "connected": false
                     }
              }
       }
}
```

Note: A "null" in the error information means that the CCAP connection has been disconnected as a result of a RPC SET.

- GATES "CONFIG" DATASTORE API STRUCTURE CHANGED
  - A CCAP RPC poll returns the gate status info, and also triggers a Operational Datastorestatus update.
  - At a minimum the appIdneeds to be included in the input, subscriberIdand gateIdare optional.
  - A gate status response is only included if the RPC request is done for a specific gate (subscriberIdand gateIdincluded in input).
  - Add and retrieve gates to/from the Config Datastore:

```
PUT http://localhost:8181/restconf/config/packetcable:qos/apps/app/cto-app/
↪subscribers/subscriber/44.137.0.12/gates/gate/gate88/

{
  "gate": [
    {
      "gateId": "gate88",
      "gate0spec": {
        "dscp-tos-overwrite": "0xa0",
        "dscp-tos-mask": "0xff"
      },
      "traffic-profile": {
        "service-class-name": "extrm_dn"
      },
      "classifiers": {
        "classifier-container": [
          {
            "classifier-id": "1",
            "classifier": {
              "srcIp": "44.137.0.0",
              "dstIp": "44.137.0.11",
              "protocol": "0",
              "srcPort": "1234",
              "dstPort": "4321",
              "tos-byte": "0xa0",
              "tos-mask": "0xe0"
            }
          }
        ]
      }
    }
  ]
}

GET http://localhost:8181/restconf/config/packetcable:qos/apps/app/cto-app/
↪subscribers/subscriber/44.137.0.12/gates/gate/gate88/
```

- GATES SUPPORT MULTIPLE (UP TO FOUR) CLASSIFIERS
  - Please note that there is a classifier container now that can have up to four classifiers:

```
PUT http://localhost:8181/restconf/config/packetcable:qos/apps/app/cto-app/
↪subscribers/subscriber/44.137.0.12/gates/gate/gate88/
{ "gate":{
    "gateId": "gate44",
    "gate-spec": {
    "dscp-tos-overwrite": "0xa0",
              "dscp-tos-mask": "0xff" },
    "traffic-profile": {
              "service-class-name": "extrm_dn"},
    "classifiers":
              { "classifier-container":[
                        { "classifier-id": "1",
                                  "ipv6-classifier": {
                                            "srcIp6":
↪"2001:4978:030d:1100:0:0:0:0/64",
                                                                "dstIp6":
↪"2001:4978:030d:1000:0:0:0:0/64",
```

```
                                        "flow-label": "102",
                                        "tc-low": "0xa0",
                                        "tc-high": "0xc0",
                                        "tc-mask": "0xe0",
                                        "next-hdr": "256",
                                        "srcPort-start": "4321",
                                        "srcPort-end": "4322",
                                        "dstPort-start": "1234",
                                        "dstPort-end": "1235"
                            }},
                            { "classifier-id": "2",
                                    "ext-classifier" : {
                                             "srcIp": "44.137.0.12",
                                             "srcIpMask": "255.255.255.255",
                                             "dstIp": "10.10.10.0",
                                             "dstIpMask": "255.255.255.0",
                                             "tos-byte": "0xa0",
                                             "tos-mask": "0xe0",
                                             "protocol": "0",
                                             "srcPort-start": "4321",
                                             "srcPort-end": "4322",
                                             "dstPort-start": "1234",
                                             "dstPort-end": "1235"
                                    }
                            }]
                }
    }
}
```

- CCAP OPERATIONAL STATUS - GET GATE STATUS FROM OPERATIONAL DATASTORE
  - Shows the Operational Datastore contents for the gate.
  - The gate status is updated at the time when the gate is configured or during an RPC poll:

```
GET http://localhost:8181/restconf/operational/packetcable:qos/apps/app/cto-
↪app/subscribers/subscriber/44.137.0.12/gates/gate/gate88

Response: 200
{
    "gate":[{
            "gateId":"gate88",
            "cops-gate-usage-info": "0",
            "cops-gate-state": "Committed(4)/Other(-1)",
            "gatePath": "cto-app/44.137.0.12/gate88",
            "cops-gate-time-info": "0",
            "cops-gateId": "3e0800e9",
            "timestamp": "2016-03-24T10:30:18.763-05:00",
            "ccapId": "E6-CTO"
    }]
}
```

- CCAP OPERATIONAL STATUS - RPC GATE STATUS POLL
  - A CCAP RPC poll returns the gate status info and also triggers an Operational Datastore status update.
  - At a minimum, the appId needs to be included in the input; subscriberId and gateId are optional.
  - A gate status response is only included if the RPC request is done for a specific gate (subscriberId and gateId included in input):

```
POST http://localhost:8181/restconf/operations/packetcable:qos-poll-gates
{
     "input": {
           "appId": "/packetcable:apps/packetcable:apps[packetcable:appId=
↪'cto-app]",
           "subscriberId": "44.137.0.11",
           "gateId": "gate44"
     }
}
Response: 200 OK
{
     "output": {
                "gate": {
                         "cops-gate-usage-info": "0",
                         "cops-gate-state": "Committed(4)/Other(-1)",
                         "gatePath": "ctoapp/44.137.0.12/gate88",
                         "cops-gate-time-info": "0",
                         "cops-gateId": "1ceb0001",
                         "error": [""],
                         "timestamp": "2016-03-24T13:22:59.900-05:00",
                         "ccapId": "E6-CTO"
                },
                "response": "cto-app/44.137.0.12/gate88: gate poll complete",
                "timestamp": "2016-03-24T13:22:59.906-05:00"
     }
}
```

– When multiple gates are polled (only appId or appId and subscriberId are provided), a generic response is returned and the Operational Datastore is updated in the background:

```
{  "output": {
       "gate": {},
       "response": "cto-app/: gate subtree poll in progress",
       "timestamp": "2016-03-24T13:25:30.471-05:00"
   }
}
```

Service Function Chaining

OpenDaylight Service Function Chaining (SFC) Overview

OpenDaylight Service Function Chaining (SFC) provides the ability to define an ordered list of a network services (e.g. firewalls, load balancers). These service are then "stitched" together in the network to create a service chain. This project provides the infrastructure (chaining logic, APIs) needed for ODL to provision a service chain in the network and an end-user application for defining such chains.

- ACE - Access Control Entry
- ACL - Access Control List
- SCF - Service Classifier Function
- SF - Service Function
- SFC - Service Function Chain
- SFF - Service Function Forwarder

- SFG - Service Function Group
- SFP - Service Function Path
- RSP - Rendered Service Path
- NSH - Network Service Header

SFC User Interface

Overview

SFC User Interface (SFC-UI) is based on Dlux project. It provides an easy way to create, read, update and delete configuration stored in datastore. Moreover, it shows the status of all SFC features (e.g installed, uninstalled) and Karaf log messages as well.

SFC-UI Architecture

SFC-UI operates purely by using RESTCONF.

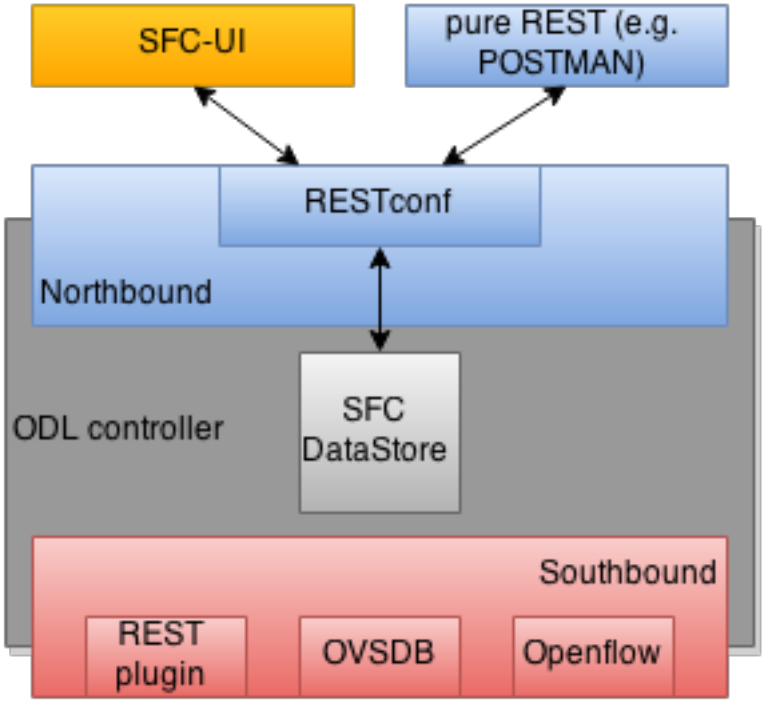


Fig. 1.95: SFC-UI integration into ODL

Configuring SFC-UI

1. Run ODL distribution (run karaf)
2. In Karaf console execute: `feature:install odl-sfc-ui`
3. Visit SFC-UI on: `http://<odl_ip_address>:8181/sfc/index.html`

SFC Southbound REST Plug-in

Overview

The Southbound REST Plug-in is used to send configuration from datastore down to network devices supporting a REST API (i.e. they have a configured REST URI). It supports POST/PUT/DELETE operations, which are triggered accordingly by changes in the SFC data stores.

- Access Control List (ACL)
- Service Classifier Function (SCF)
- Service Function (SF)
- Service Function Group (SFG)
- Service Function Schedule Type (SFST)
- Service Function Forwarder (SFF)
- Rendered Service Path (RSP)

Southbound REST Plug-in Architecture

From the user perspective, the REST plug-in is another SFC Southbound plug-in used to communicate with network devices.

Configuring Southbound REST Plugin

1. Run ODL distribution (run karaf)
2. In Karaf console execute: `feature:install odl-sfc-sb-rest`
3. Configure REST URIs for SF/SFF through SFC User Interface or RESTCONF (required configuration steps can be found in the tutorial stated bellow)

Tutorial

Comprehensive tutorial on how to use the Southbound REST Plug-in and how to control network devices with it can be found on: https://wiki.opendaylight.org/view/Service_Function_Chaining:Main#SFC_101

SFC-OVS integration

Overview

SFC-OVS provides integration of SFC with Open vSwitch (OVS) devices. Integration is realized through mapping of SFC objects (like SF, SFF, Classifier, etc.) to OVS objects (like Bridge, TerminationPoint=Port/Interface). The mapping takes care of automatic instantiation (setup) of corresponding object whenever its counterpart is created. For example, when a new SFF is created, the SFC-OVS plug-in will create a new OVS bridge and when a new OVS Bridge is created, the SFC-OVS plug-in will create a new SFF.

The feature is intended for SFC users willing to use Open vSwitch as underlying network infrastructure for deploying RSPs (Rendered Service Paths).

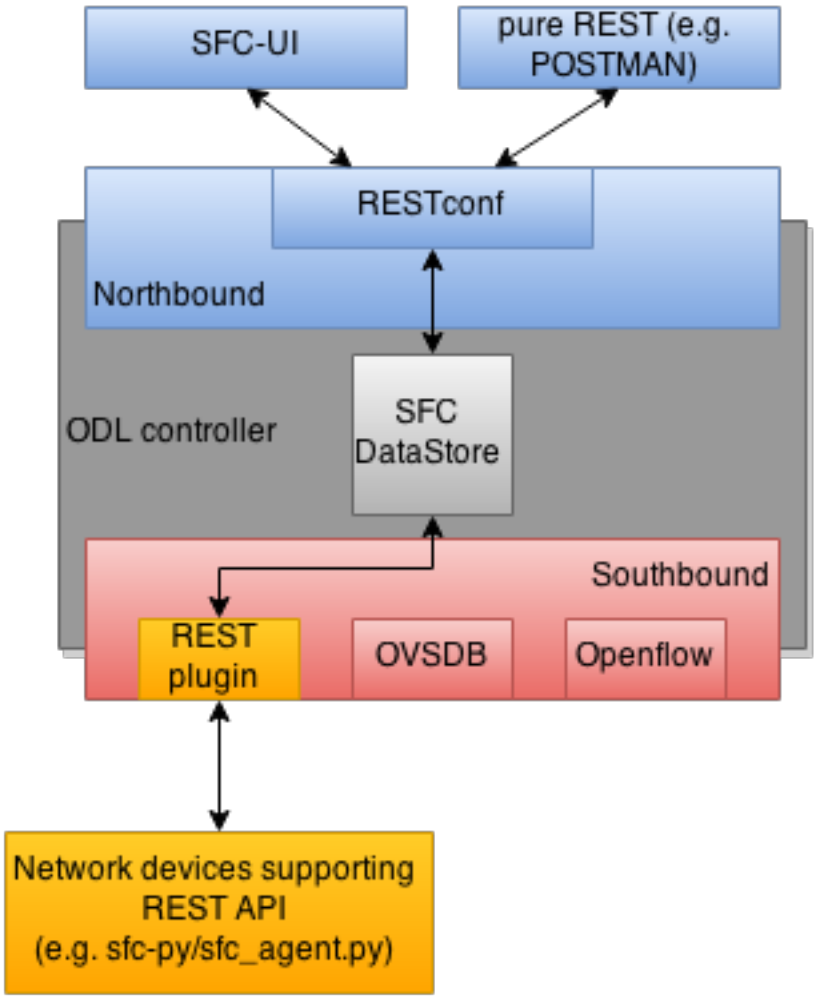


Fig. 1.96: Southbound REST Plug-in integration into ODL

SFC-OVS Architecture

SFC-OVS uses the OVSDB MD-SAL Southbound API for getting/writing information from/to OVS devices. From the user perspective SFC-OVS acts as a layer between SFC datastore and OVSDB.

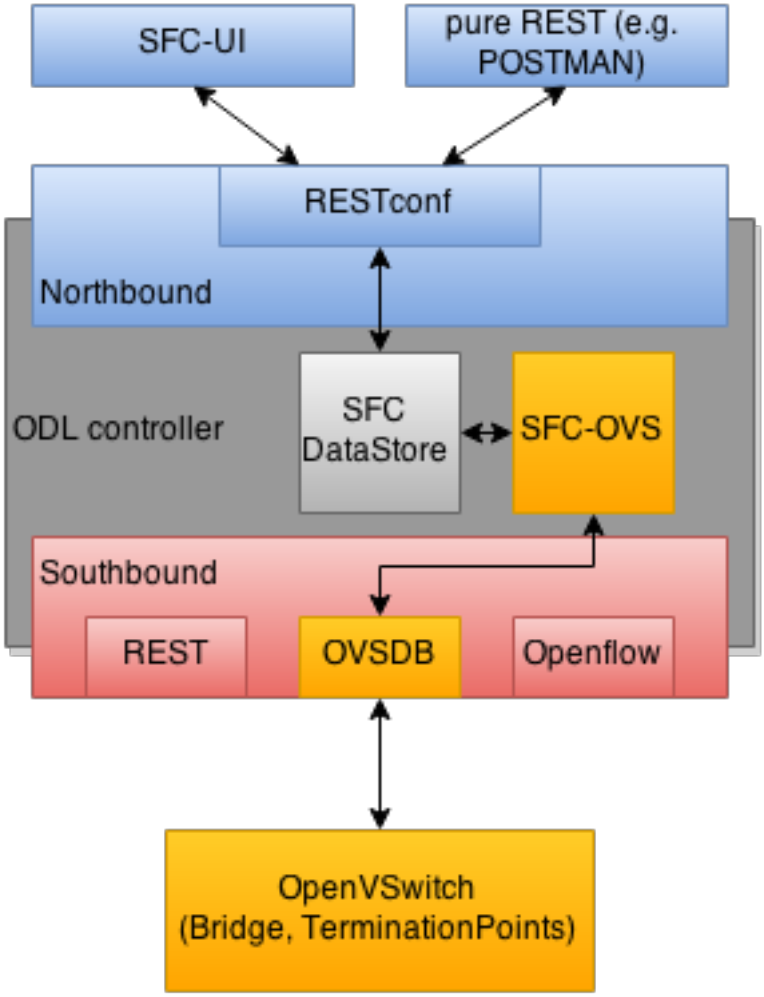


Fig. 1.97: SFC-OVS integration into ODL

Configuring SFC-OVS

1. Run ODL distribution (run karaf)
2. In Karaf console execute: `feature:install odl-sfc-ovs`
3. Configure Open vSwitch to use ODL as a manager, using following command: `ovs-vsctl set-manager tcp:<odl_ip_address>:6640`

Tutorials

Verifying mapping from OVS to SFF

Overview

This tutorial shows the usual work flow when OVS configuration is transformed to corresponding SFC objects (in this case SFF).

Prerequisites

- Open vSwitch installed (ovs-vsctl command available in shell)
- SFC-OVS feature configured as stated above

Instructions

1. `ovs-vsctl set-manager tcp:<odl_ip_address>:6640`
2. `ovs-vsctl add-br br1`
3. `ovs-vsctl add-port br1 testPort`

Verification

1. visit SFC User Interface: `http://<odl_ip_address>:8181/sfc/index.html#/sfc/serviceforwarder`
2. use pure RESTCONF and send GET request to URL: `http://<odl_ip_address>:8181/restconf/config/service-function-forwarder:service-function-forwarders`

There should be SFF, which name will be ending with *br1* and the SFF should containt two DataPlane locators: *br1* and *testPort*.

Verifying mapping from SFF to OVS

Overview

This tutorial shows the usual workflow during creation of OVS Bridge with use of SFC APIs.

Prerequisites

- Open vSwitch installed (ovs-vsctl command available in shell)
- SFC-OVS feature configured as stated above

Instructions

1. In shell execute: `ovs-vsctl set-manager tcp:<odl_ip_address>:6640`
2. Send POST request to URL: `http://<odl_ip_address>:8181/restconf/operations/service-function-forwarder-ovs:create-ovs-bridge` Use Basic auth with credentials: "admin", "admin" and set `Content-Type: application/json`. The content of POST request should be following:

```
{
    "input":
    {
        "name": "br-test",
        "ovs-node": {
            "ip": "<Open_vSwitch_ip_address>"
        }
    }
}
```

Open_vSwitch_ip_address is IP address of machine, where Open vSwitch is installed.

Verification

In shell execute: `ovs-vsctl show`. There should be Bridge with name *br-test* and one port/interface called *br-test*.

Also, corresponding SFF for this OVS Bridge should be configured, which can be verified through SFC User Interface or RESTCONF as stated in previous tutorial.

SFC Classifier User Guide

Overview

Description of classifier can be found in: https://datatracker.ietf.org/doc/draft-ietf-sfc-architecture/

There are two types of classifier:

1. OpenFlow Classifier
2. Iptables Classifier

OpenFlow Classifier

OpenFlow Classifier implements the classification criteria based on OpenFlow rules deployed into an OpenFlow switch. An Open vSwitch will take the role of a classifier and performs various encapsulations such NSH, VLAN, MPLS, etc. In the existing implementation, classifier can support NSH encapsulation. Matching information is based on ACL for MAC addresses, ports, protocol, IPv4 and IPv6. Supported protocols are TCP, UDP and SCTP. Actions information in the OF rules, shall be forwarding of the encapsulated packets with specific information related to the RSP.

Classifier Architecture

The OVSDB Southbound interface is used to create an instance of a bridge in a specific location (via IP address). This bridge contains the OpenFlow rules that perform the classification of the packets and react accordingly. The OpenFlow

Southbound interface is used to translate the ACL information into OF rules within the Open vSwitch.

Note: in order to create the instance of the bridge that takes the role of a classifier, an "empty" SFF must be created.

Configuring Classifier

1. An empty SFF must be created in order to host the ACL that contains the classification information.
2. SFF data plane locator must be configured
3. Classifier interface must be manually added to SFF bridge.

Administering or Managing Classifier

Classification information is based on MAC addresses, protocol, ports and IP. ACL gathers this information and is assigned to an RSP which turns to be a specific path for a Service Chain.

Iptables Classifier

Classifier manages everything from starting the packet listener to creation (and removal) of appropriate ip(6)tables rules and marking received packets accordingly. Its functionality is **available only on Linux** as it leverdges **NetfilterQueue**, which provides access to packets matched by an **iptables** rule. Classifier requires **root privileges** to be able to operate.

So far it is capable of processing ACL for MAC addresses, ports, IPv4 and IPv6. Supported protocols are TCP and UDP.

Classifier Architecture

Python code located in the project repository sfc-py/common/classifier.py.

Note: classifier assumes that Rendered Service Path (RSP) **already exists** in ODL when an ACL referencing it is obtained

1. sfc_agent receives an ACL and passes it for processing to the classifier
2. the RSP (its SFF locator) referenced by ACL is requested from ODL
3. if the RSP exists in the ODL then ACL based iptables rules for it are applied

After this process is over, every packet successfully matched to an iptables rule (i.e. successfully classified) will be NSH encapsulated and forwarded to a related SFF, which knows how to traverse the RSP.

Rules are created using appropriate iptables command. If the Access Control Entry (ACE) rule is MAC address related both iptables and IPv6 tables rules re issued. If ACE rule is IPv4 address related, only iptables rules are issued, same for IPv6.

Note: iptables **raw** table contains all created rules

Configuring Classifier

Classfier does't need any configuration.
Its only requirement is that the **second (2) Netfilter Queue** is not used by any other process and is **avalilable for the classifier**.

Administering or Managing Classifier

Classifier runs alongside sfc_agent, therefore the command for starting it locally is:

```
sudo python3.4 sfc-py/sfc_agent.py --rest --odl-ip-port localhost:8181 --auto-sff-
↪name --nfq-class
```

SFC OpenFlow Renderer User Guide

Overview

The Service Function Chaining (SFC) OpenFlow Renderer (SFC OF Renderer) implements Service Chaining on OpenFlow switches. It listens for the creation of a Rendered Service Path (RSP), and once received it programs Service Function Forwarders (SFF) that are hosted on OpenFlow capable switches to steer packets through the service chain.

Common acronyms used in the following sections:

- SF - Service Function
- SFF - Service Function Forwarder
- SFC - Service Function Chain
- SFP - Service Function Path
- RSP - Rendered Service Path

SFC OpenFlow Renderer Architecture

The SFC OF Renderer is invoked after a RSP is created using an MD-SAL listener called `SfcOfRspDataListener`. Upon SFC OF Renderer initialization, the `SfcOfRspDataListener` registers itself to listen for RSP changes. When invoked, the `SfcOfRspDataListener` processes the RSP and calls the `SfcOfFlowProgrammerImpl` to create the necessary flows in the Service Function Forwarders configured in the RSP. Refer to the following diagram for more details.

SFC OpenFlow Switch Flow pipeline

The SFC OpenFlow Renderer uses the following tables for its Flow pipeline:

- Table 0, Classifier
- Table 1, Transport Ingress
- Table 2, Path Mapper
- Table 3, Path Mapper ACL

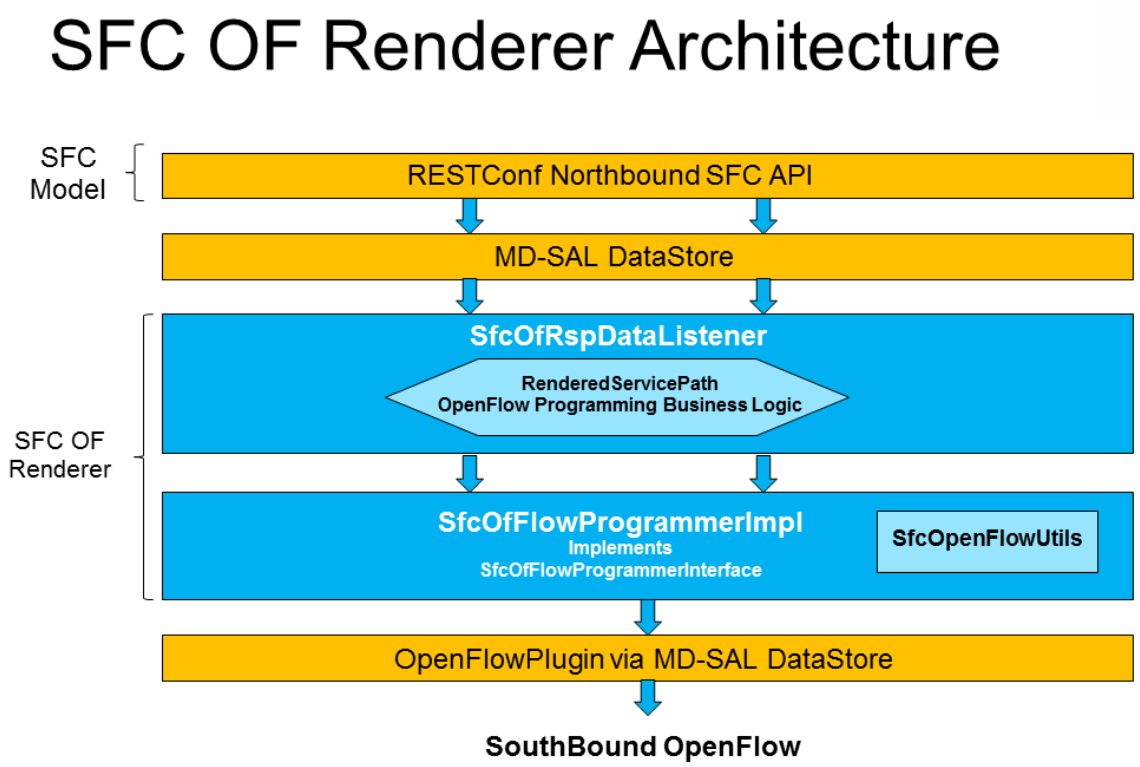


Fig. 1.98: SFC OpenFlow Renderer High Level Architecture

- Table 4, Next Hop
- Table 10, Transport Egress

The OpenFlow Table Pipeline is intended to be generic to work for all of the different encapsulations supported by SFC.

All of the tables are explained in detail in the following section.

The SFFs (SFF1 and SFF2), SFs (SF1), and topology used for the flow tables in the following sections are as described in the following diagram.

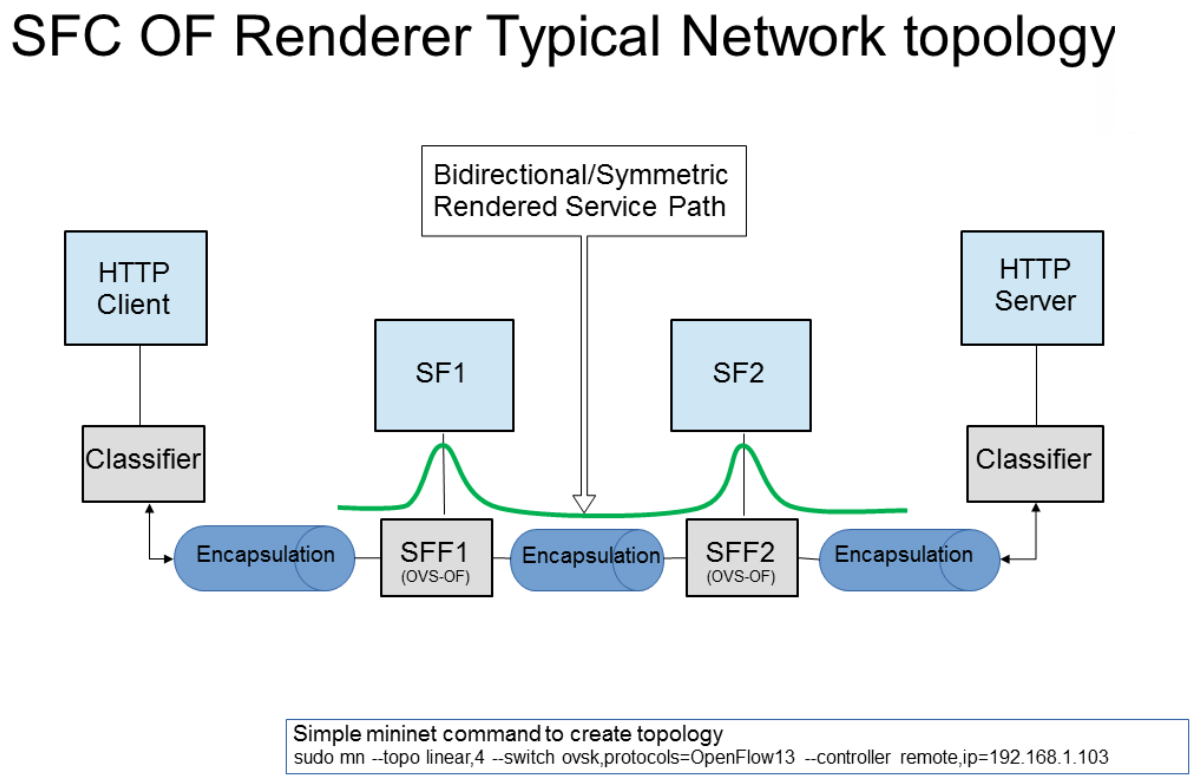


Fig. 1.99: SFC OpenFlow Renderer Typical Network Topology

Classifier Table detailed

It is possible for the SFF to also act as a classifier. This table maps subscriber traffic to RSPs, and is explained in detail in the classifier documentation.

If the SFF is not a classifier, then this table will just have a simple Goto Table 1 flow.

Transport Ingress Table detailed

The Transport Ingress table has an entry per expected tunnel transport type to be received in a particular SFF, as established in the SFC configuration.

Here are two example on SFF1: one where the RSP ingress tunnel is MPLS assuming VLAN is used for the SFF-SF, and the other where the RSP ingress tunnel is NSH GRE (UDP port 4789):

Priority	Match	Action
256	EtherType==0x8847 (MPLS unicast)	Goto Table 2
256	EtherType==0x8100 (VLAN)	Goto Table 2
256	EtherType==0x0800,udp,tp_dst==4789 (IP v4)	Goto Table 2
5	Match Any	Drop

Table: Table Transport Ingress

Path Mapper Table detailed

The Path Mapper table has an entry per expected tunnel transport info to be received in a particular SFF, as established in the SFC configuration. The tunnel transport info is used to determine the RSP Path ID, and is stored in the OpenFlow Metadata. This table is not used for NSH, since the RSP Path ID is stored in the NSH header.

For SF nodes that do not support NSH tunneling, the IP header DSCP field is used to store the RSP Path Id. The RSP Path Id is written to the DSCP field in the Transport Egress table for those packets sent to an SF.

Here is an example on SFF1, assuming the following details:

- VLAN ID 1000 is used for the SFF-SF
- The RSP Path 1 tunnel uses MPLS label 100 for ingress and 101 for egress
- The RSP Path 2 (symmetric downlink path) uses MPLS label 101 for ingress and 100 for egress

Priority	Match	Action
256	MPLS Label==100	RSP Path=1, Pop MPLS, Goto Table 4
256	MPLS Label==101	RSP Path=2, Pop MPLS, Goto Table 4
256	VLAN ID==1000, IP DSCP==1	RSP Path=1, Pop VLAN, Goto Table 4
256	VLAN ID==1000, IP DSCP==2	RSP Path=2, Pop VLAN, Goto Table 4
5	Match Any	Goto Table 3

Table: Table Path Mapper

Path Mapper ACL Table detailed

This table is only populated when PacketIn packets are received from the switch for TcpProxy type SFs. These flows are created with an inactivity timer of 60 seconds and will be automatically deleted upon expiration.

Next Hop Table detailed

The Next Hop table uses the RSP Path Id and appropriate packet fields to determine where to send the packet next. For NSH, only the NSP (Network Services Path, RSP ID) and NSI (Network Services Index, next hop) fields from the NSH header are needed to determine the VXLAN tunnel destination IP. For VLAN or MPLS, then the source MAC address is used to determine the destination MAC address.

Here are two examples on SFF1, assuming SFF1 is connected to SFF2. RSP Paths 1 and 2 are symmetric VLAN paths. RSP Paths 3 and 4 are symmetric NSH paths. RSP Path 1 ingress packets come from external to SFC, for which we don't have the source MAC address (MacSrc).

Priority	Match	Action
256	RSP Path==1, MacSrc==SF1	MacDst=SFF2, Goto Table 10
256	RSP Path==2, MacSrc==SF1	Goto Table 10
256	RSP Path==2, MacSrc==SFF2	MacDst=SF1, Goto Table 10
246	RSP Path==1	MacDst=SF1, Goto Table 10
256	nsp=3,nsi=255 (SFF Ingress RSP 3)	load:0xa000002→NXM_NX_TUN_I PV4_DST[], Goto Table 10
256	nsp=3,nsi=254 (SFF Ingress from SF, RSP 3)	load:0xa00000a→NXM_NX_TUN_I PV4_DST[], Goto Table 10
256	nsp=4,nsi=254 (SFF1 Ingress from SFF2)	load:0xa00000a→NXM_NX_TUN_I PV4_DST[], Goto Table 10
5	Match Any	Drop

Table: Table Next Hop

Transport Egress Table detailed

The Transport Egress table prepares egress tunnel information and sends the packets out.

Here are two examples on SFF1. RSP Paths 1 and 2 are symmetric MPLS paths that use VLAN for the SFF-SF. RSP Paths 3 and 4 are symmetric NSH paths. Since it is assumed that switches used for NSH will only have one VXLAN port, the NSH packets are just sent back where they came from.

Priority	Match	Action
256	RSP Path==1, MacDst==SF1	Push VLAN ID 1000, Port=SF1
256	RSP Path==1, MacDst==SFF2	Push MPLS Label 101, Port=SFF2
256	RSP Path==2, MacDst==SF1	Push VLAN ID 1000, Port=SF1
246	RSP Path==2	Push MPLS Label 100, Port=Ingress
256	nsp=3,nsi=255 (SFF Ingress RSP 3)	IN_PORT
256	nsp=3,nsi=254 (SFF Ingress from SF, RSP 3)	IN_PORT
256	nsp=4,nsi=254 (SFF1 Ingress from SFF2)	IN_PORT
5	Match Any	Drop

Table: Table Transport Egress

Administering SFC OF Renderer

To use the SFC OpenFlow Renderer Karaf, at least the following Karaf features must be installed.

- odl-openflowplugin-nxm-extensions
- odl-openflowplugin-flow-services
- odl-sfc-provider
- odl-sfc-model
- odl-sfc-openflow-renderer
- odl-sfc-ui (optional)

The following command can be used to view all of the currently installed Karaf features:

```
opendaylight-user@root>feature:list -i
```

Or, pipe the command to a grep to see a subset of the currently installed Karaf features:

```
opendaylight-user@root>feature:list -i | grep sfc
```

To install a particular feature, use the Karaf `feature:install` command.

SFC OF Renderer Tutorial

Overview

In this tutorial, 2 different encapsulations will be shown: MPLS and NSH. The following Network Topology diagram is a logical view of the SFFs and SFs involved in creating the Service Chains.

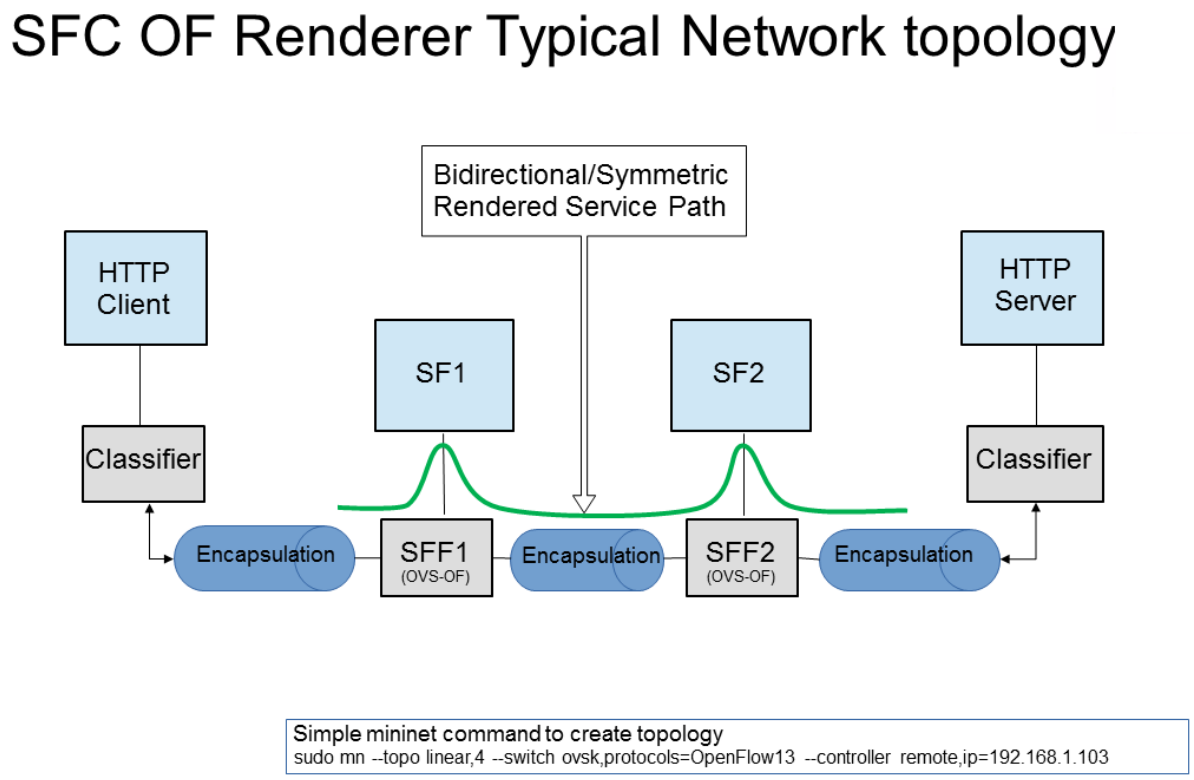


Fig. 1.100: SFC OpenFlow Renderer Typical Network Topology

Prerequisites

To use this example, SFF OpenFlow switches must be created and connected as illustrated above. Additionally, the SFs must be created and connected.

Note that RSP symmetry depends on Service Function Path symmetric field, if present. If not, the RSP will be symmetric if any of the SFs involved in the chain has the bidirectional field set to true.

Target Environment

The target environment is not important, but this use-case was created and tested on Linux.

Instructions

The steps to use this tutorial are as follows. The referenced configuration in the steps is listed in the following sections.

There are numerous ways to send the configuration. In the following configuration chapters, the appropriate `curl` command is shown for each configuration to be sent, including the URL.

Steps to configure the SFC OF Renderer tutorial:

1. Send the `SF` RESTCONF configuration
2. Send the `SFF` RESTCONF configuration
3. Send the `SFC` RESTCONF configuration
4. Send the `SFP` RESTCONF configuration
5. Create the `RSP` with a RESTCONF RPC command

Once the configuration has been successfully created, query the Rendered Service Paths with either the SFC UI or via RESTCONF. Notice that the RSP is symmetrical, so the following 2 RSPs will be created:

- sfc-path1
- sfc-path1-Reverse

At this point the Service Chains have been created, and the OpenFlow Switches are programmed to steer traffic through the Service Chain. Traffic can now be injected from a client into the Service Chain. To debug problems, the OpenFlow tables can be dumped with the following commands, assuming SFF1 is called `s1` and SFF2 is called `s2`.

```
sudo ovs-ofctl -O OpenFlow13  dump-flows s1
```

```
sudo ovs-ofctl -O OpenFlow13  dump-flows s2
```

In all the following configuration sections, replace the `${JSON}` string with the appropriate JSON configuration. Also, change the `localhost` destination in the URL accordingly.

SFC OF Renderer NSH Tutorial

The following configuration sections show how to create the different elements using NSH encapsulation.

NSH Service Function configuration

The Service Function configuration can be sent with the following command:

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪{JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/config/service-
↪function:service-functions/
```

SF configuration JSON.

```
{
 "service-functions": {
   "service-function": [
     {
       "name": "sf1",
       "type": "http-header-enrichment",
       "ip-mgmt-address": "10.0.0.2",
       "sf-data-plane-locator": [
         {
           "name": "sf1dpl",
           "ip": "10.0.0.10",
           "port": 4789,
           "transport": "service-locator:vxlan-gpe",
           "service-function-forwarder": "sff1"
         }
       ]
     },
     {
       "name": "sf2",
       "type": "firewall",
       "ip-mgmt-address": "10.0.0.3",
       "sf-data-plane-locator": [
         {
           "name": "sf2dpl",
            "ip": "10.0.0.20",
            "port": 4789,
            "transport": "service-locator:vxlan-gpe",
           "service-function-forwarder": "sff2"
         }
       ]
     }
   ]
 }
}
```

NSH Service Function Forwarder configuration

The Service Function Forwarder configuration can be sent with the following command:

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪{JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/config/service-
↪function-forwarder:service-function-forwarders/
```

SFF configuration JSON.

```
{
 "service-function-forwarders": {
   "service-function-forwarder": [
     {
       "name": "sff1",
       "service-node": "openflow:2",
       "sff-data-plane-locator": [
         {
           "name": "sff1dpl",
```

```
            "data-plane-locator":
            {
                "ip": "10.0.0.1",
                "port": 4789,
                "transport": "service-locator:vxlan-gpe"
            }
          }
        ],
        "service-function-dictionary": [
          {
            "name": "sf1",
            "sff-sf-data-plane-locator":
            {
                "sf-dpl-name": "sf1dpl",
                "sff-dpl-name": "sff1dpl"
            }
          }
        ]
      },
      {
        "name": "sff2",
        "service-node": "openflow:3",
        "sff-data-plane-locator": [
          {
            "name": "sff2dpl",
            "data-plane-locator":
            {
                "ip": "10.0.0.2",
                "port": 4789,
                "transport": "service-locator:vxlan-gpe"
            }
          }
        ],
        "service-function-dictionary": [
          {
            "name": "sf2",
            "sff-sf-data-plane-locator":
            {
                "sf-dpl-name": "sf2dpl",
                "sff-dpl-name": "sff2dpl"
            }
          }
        ]
      }
    ]
  }
}
```

NSH Service Function Chain configuration

The Service Function Chain configuration can be sent with the following command:

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪{JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/config/service-
↪function-chain:service-function-chains/
```

SFC configuration JSON.

```
{
 "service-function-chains": {
   "service-function-chain": [
     {
       "name": "sfc-chain1",
       "sfc-service-function": [
         {
           "name": "hdr-enrich-abstract1",
           "type": "http-header-enrichment"
         },
         {
           "name": "firewall-abstract1",
           "type": "firewall"
         }
       ]
     }
   ]
 }
}
```

NSH Service Function Path configuration

The Service Function Path configuration can be sent with the following command:

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪{JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/config/service-
↪function-path:service-function-paths/
```

SFP configuration JSON.

```
{
  "service-function-paths": {
    "service-function-path": [
      {
        "name": "sfc-path1",
        "service-chain-name": "sfc-chain1",
        "transport-type": "service-locator:vxlan-gpe",
        "symmetric": true
      }
    ]
  }
}
```

NSH Rendered Service Path creation

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪{JSON}' -X POST --user admin:admin http://localhost:8181/restconf/operations/
↪rendered-service-path:create-rendered-path/
```

RSP creation JSON.

```
{
 "input": {
     "name": "sfc-path1",
     "parent-service-function-path": "sfc-path1"
 }
}
```

NSH Rendered Service Path removal

The following command can be used to remove a Rendered Service Path called `sfc-path1`:

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '{
↪"input": {"name": "sfc-path1" } }' -X POST --user admin:admin http://localhost:8181/
↪restconf/operations/rendered-service-path:delete-rendered-path/
```

NSH Rendered Service Path Query

The following command can be used to query all of the created Rendered Service Paths:

```
curl -H "Content-Type: application/json" -H "Cache-Control: no-cache" -X GET --user␣
↪admin:admin http://localhost:8181/restconf/operational/rendered-service-
↪path:rendered-service-paths/
```

SFC OF Renderer MPLS Tutorial

The following configuration sections show how to create the different elements using MPLS encapsulation.

MPLS Service Function configuration

The Service Function configuration can be sent with the following command:

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪{JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/config/service-
↪function:service-functions/
```

SF configuration JSON.

```
{
 "service-functions": {
   "service-function": [
     {
       "name": "sf1",
       "type": "http-header-enrichment",
       "ip-mgmt-address": "10.0.0.2",
       "sf-data-plane-locator": [
         {
           "name": "sf1-sff1",
           "mac": "00:00:08:01:02:01",
```

```
            "vlan-id": 1000,
            "transport": "service-locator:mac",
            "service-function-forwarder": "sff1"
          }
        ]
      },
      {
        "name": "sf2",
        "type": "firewall",
        "ip-mgmt-address": "10.0.0.3",
        "sf-data-plane-locator": [
          {
            "name": "sf2-sff2",
            "mac": "00:00:08:01:03:01",
            "vlan-id": 2000,
            "transport": "service-locator:mac",
            "service-function-forwarder": "sff2"
          }
        ]
      }
    ]
  }
}
```

MPLS Service Function Forwarder configuration

The Service Function Forwarder configuration can be sent with the following command:

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪{JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/config/service-
↪function-forwarder:service-function-forwarders/
```

SFF configuration JSON.

```
{
 "service-function-forwarders": {
   "service-function-forwarder": [
     {
       "name": "sff1",
       "service-node": "openflow:2",
       "sff-data-plane-locator": [
         {
           "name": "ulSff1Ingress",
           "data-plane-locator":
           {
               "mpls-label": 100,
               "transport": "service-locator:mpls"
           },
           "service-function-forwarder-ofs:ofs-port":
           {
               "mac": "11:11:11:11:11:11",
               "port-id" : "1"
           }
         },
         {
```

```
            "name": "ulSff1ToSff2",
            "data-plane-locator":
            {
                "mpls-label": 101,
                "transport": "service-locator:mpls"
            },
            "service-function-forwarder-ofs:ofs-port":
            {
                "mac": "33:33:33:33:33:33",
                "port-id" : "2"
            }
          },
          {
            "name": "toSf1",
            "data-plane-locator":
            {
                "mac": "22:22:22:22:22:22",
                "vlan-id": 1000,
                "transport": "service-locator:mac",
            },
            "service-function-forwarder-ofs:ofs-port":
            {
                "mac": "33:33:33:33:33:33",
                "port-id" : "3"
            }
          }
        ],
        "service-function-dictionary": [
          {
            "name": "sf1",
            "sff-sf-data-plane-locator":
            {
                "sf-dpl-name": "sf1-sff1",
                "sff-dpl-name": "toSf1"
            }
          }
        ]
      },
      {
        "name": "sff2",
        "service-node": "openflow:3",
        "sff-data-plane-locator": [
          {
            "name": "ulSff2Ingress",
            "data-plane-locator":
            {
                "mpls-label": 101,
                "transport": "service-locator:mpls"
            },
            "service-function-forwarder-ofs:ofs-port":
            {
                "mac": "44:44:44:44:44:44",
                "port-id" : "1"
            }
          },
          {
            "name": "ulSff2Egress",
            "data-plane-locator":
```

```
            {
                "mpls-label": 102,
                "transport": "service-locator:mpls"
            },
            "service-function-forwarder-ofs:ofs-port":
            {
                "mac": "66:66:66:66:66:66",
                "port-id" : "2"
            }
          },
          {
            "name": "toSf2",
            "data-plane-locator":
            {
                "mac": "55:55:55:55:55:55",
                "vlan-id": 2000,
                "transport": "service-locator:mac"
            },
            "service-function-forwarder-ofs:ofs-port":
            {
                "port-id" : "3"
            }
          }
        ],
        "service-function-dictionary": [
          {
            "name": "sf2",
            "sff-sf-data-plane-locator":
            {
                "sf-dpl-name": "sf2-sff2",
                "sff-dpl-name": "toSf2"

            },
            "service-function-forwarder-ofs:ofs-port":
            {
                "port-id" : "3"
            }
          }
        ]
      }
    ]
  }
}
```

MPLS Service Function Chain configuration

The Service Function Chain configuration can be sent with the following command:

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪{JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/config/service-
↪function-chain:service-function-chains/
```

SFC configuration JSON.

```
{
 "service-function-chains": {
   "service-function-chain": [
     {
       "name": "sfc-chain1",
       "sfc-service-function": [
         {
           "name": "hdr-enrich-abstract1",
           "type": "http-header-enrichment"
         },
         {
           "name": "firewall-abstract1",
           "type": "firewall"
         }
       ]
     }
   ]
 }
}
```

MPLS Service Function Path configuration

The Service Function Path configuration can be sent with the following command:

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪{JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/config/service-
↪function-path:service-function-paths/
```

SFP configuration JSON.

```
{
  "service-function-paths": {
    "service-function-path": [
      {
        "name": "sfc-path1",
        "service-chain-name": "sfc-chain1",
        "transport-type": "service-locator:mpls",
        "symmetric": true
      }
    ]
  }
}
```

MPLS Rendered Service Path creation

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪{JSON}' -X POST --user admin:admin http://localhost:8181/restconf/operations/
↪rendered-service-path:create-rendered-path/
```

RSP creation JSON.

```
{
 "input": {
     "name": "sfc-path1",
     "parent-service-function-path": "sfc-path1"
 }
}
```

MPLS Rendered Service Path removal

The following command can be used to remove a Rendered Service Path called `sfc-path1`:

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '{
↪"input": {"name": "sfc-path1" } }' -X POST --user admin:admin http://localhost:8181/
↪restconf/operations/rendered-service-path:delete-rendered-path/
```

MPLS Rendered Service Path Query

The following command can be used to query all of the created Rendered Service Paths:

```
curl -H "Content-Type: application/json" -H "Cache-Control: no-cache" -X GET --user␣
↪admin:admin http://localhost:8181/restconf/operational/rendered-service-
↪path:rendered-service-paths/
```

SFC IOS XE Renderer User Guide

Overview

The early Service Function Chaining (SFC) renderer for IOS-XE devices (SFC IOS-XE renderer) implements Service Chaining functionality on IOS-XE capable switches. It listens for the creation of a Rendered Service Path (RSP) and sets up Service Function Forwarders (SFF) that are hosted on IOS-XE switches to steer traffic through the service chain.

Common acronyms used in the following sections:

- SF - Service Function
- SFF - Service Function Forwarder
- SFC - Service Function Chain
- SP - Service Path
- SFP - Service Function Path
- RSP - Rendered Service Path
- LSF - Local Service Forwarder
- RSF - Remote Service Forwarder

SFC IOS-XE Renderer Architecture

When the SFC IOS-XE renderer is initialized, all required listeners are registered to handle incoming data. It involves CSR/IOS-XE `NodeListener` which stores data about all configurable devices including their mountpoints (used here as databrokers), `ServiceFunctionListener`, `ServiceForwarderListener` (see mapping) and `RenderedPathListener` used to listen for RSP changes. When the SFC IOS-XE renderer is invoked, `RenderedPathListener` calls the `IosXeRspProcessor` which processes the RSP change and creates all necessary Service Paths and Remote Service Forwarders (if necessary) on IOS-XE devices.

Service Path details

Each Service Path is defined by index (represented by NSP) and contains service path entries. Each entry has appropriate service index (NSI) and definition of next hop. Next hop can be Service Function, different Service Function Forwarder or definition of end of chain - terminate. After terminating, the packet is sent to destination. If a SFF is defined as a next hop, it has to be present on device in the form of Remote Service Forwarder. RSFs are also created during RSP processing.

Example of Service Path:

```
service-chain service-path 200
   service-index 255 service-function firewall-1
   service-index 254 service-function dpi-1
   service-index 253 terminate
```

Mapping to IOS-XE SFC entities

Renderer contains mappers for SFs and SFFs. IOS-XE capable device is using its own definition of Service Functions and Service Function Forwarders according to appropriate .yang file. `ServiceFunctionListener` serves as a listener for SF changes. If SF appears in datastore, listener extracts its management ip address and looks into cached IOS-XE nodes. If some of available nodes match, Service function is mapped in `IosXeServiceFunctionMapper` to be understandable by IOS-XE device and it's written into device's config. `ServiceForwarderListener` is used in a similar way. All SFFs with suitable management ip address it mapped in `IosXeServiceForwarderMapper`. Remapped SFFs are configured as a Local Service Forwarders. It is not possible to directly create Remote Service Forwarder using IOS-XE renderer. RSF is created only during RSP processing.

Administering SFC IOS-XE renderer

To use the SFC IOS-XE Renderer Karaf, at least the following Karaf features must be installed:

- odl-aaa-shiro
- odl-sfc-model
- odl-sfc-provider
- odl-restconf
- odl-netconf-topology
- odl-sfc-ios-xe-renderer

SFC IOS-XE renderer Tutorial

Overview

This tutorial is a simple example how to create Service Path on IOS-XE capable device using IOS-XE renderer

Preconditions

To connect to IOS-XE device, it is necessary to use several modified yang models and override device's ones. All .yang files are in the `Yang/netconf` folder in the `sfc-ios-xe-renderer module` in the SFC project. These files have to be copied to the `cache/schema` directory, before Karaf is started. After that, custom capabilities have to be sent to network-topology:

```
PUT ./config/network-topology:network-topology/topology/topology-netconf/node/<device-
↪name>

<node xmlns="urn:TBD:params:xml:ns:yang:network-topology">
  <node-id>device-name</node-id>
  <host xmlns="urn:opendaylight:netconf-node-topology">device-ip</host>
  <port xmlns="urn:opendaylight:netconf-node-topology">2022</port>
  <username xmlns="urn:opendaylight:netconf-node-topology">login</username>
  <password xmlns="urn:opendaylight:netconf-node-topology">password</password>
  <tcp-only xmlns="urn:opendaylight:netconf-node-topology">false</tcp-only>
  <keepalive-delay xmlns="urn:opendaylight:netconf-node-topology">0</keepalive-delay>
  <yang-module-capabilities xmlns="urn:opendaylight:netconf-node-topology">
     <override>true</override>
     <capability xmlns="urn:opendaylight:netconf-node-topology">
        urn:ietf:params:xml:ns:yang:ietf-inet-types?module=ietf-inet-types&amp;
↪revision=2013-07-15
     </capability>
     <capability xmlns="urn:opendaylight:netconf-node-topology">
        urn:ietf:params:xml:ns:yang:ietf-yang-types?module=ietf-yang-types&amp;
↪revision=2013-07-15
     </capability>
     <capability xmlns="urn:opendaylight:netconf-node-topology">
        urn:ios?module=ned&amp;revision=2016-03-08
     </capability>
     <capability xmlns="urn:opendaylight:netconf-node-topology">
        http://tail-f.com/yang/common?module=tailf-common&amp;revision=2015-05-22
     </capability>
     <capability xmlns="urn:opendaylight:netconf-node-topology">
        http://tail-f.com/yang/common?module=tailf-meta-extensions&amp;revision=2013-
↪11-07
     </capability>
     <capability xmlns="urn:opendaylight:netconf-node-topology">
        http://tail-f.com/yang/common?module=tailf-cli-extensions&amp;revision=2015-
↪03-19
     </capability>
  </yang-module-capabilities>
</node>
```

Note: The device name in the URL and in the XML must match.

Instructions

When the IOS-XE renderer is installed, all NETCONF nodes in topology-netconf are processed and all capable nodes with accessible mountpoints are cached. The first step is to create LSF on node.

`Service Function Forwarder configuration`

```
PUT ./config/service-function-forwarder:service-function-forwarders

{
    "service-function-forwarders": {
        "service-function-forwarder": [
            {
                "name": "CSR1Kv-2",
                "ip-mgmt-address": "172.25.73.23",
                "sff-data-plane-locator": [
                    {
                        "name": "CSR1Kv-2-dpl",
                        "data-plane-locator": {
                            "transport": "service-locator:vxlan-gpe",
                            "port": 6633,
                            "ip": "10.99.150.10"
                        }
                    }
                ]
            }
        ]
    }
}
```

If the IOS-XE node with appropriate management IP exists, this configuration is mapped and LSF is created on the device. The same approach is used for Service Functions.

```
PUT ./config/service-function:service-functions

{
    "service-functions": {
        "service-function": [
            {
                "name": "Firewall",
                "ip-mgmt-address": "172.25.73.23",
                "type": "firewall",
                "sf-data-plane-locator": [
                    {
                        "name": "firewall-dpl",
                        "port": 6633,
                        "ip": "12.1.1.2",
                        "transport": "service-locator:gre",
                        "service-function-forwarder": "CSR1Kv-2"
                    }
                ]
            },
            {
                "name": "Dpi",
                "ip-mgmt-address": "172.25.73.23",
                "type":"dpi",
                "sf-data-plane-locator": [
                    {
```

```
                        "name": "dpi-dpl",
                        "port": 6633,
                        "ip": "12.1.1.1",
                        "transport": "service-locator:gre",
                        "service-function-forwarder": "CSR1Kv-2"
                    }
                ]
            },
            {
                "name": "Qos",
                "ip-mgmt-address": "172.25.73.23",
                "type":"qos",
                "sf-data-plane-locator": [
                    {
                        "name": "qos-dpl",
                        "port": 6633,
                        "ip": "12.1.1.4",
                        "transport": "service-locator:gre",
                        "service-function-forwarder": "CSR1Kv-2"
                    }
                ]
            }
        ]
    }
}
```

All these SFs are configured on the same device as the LSF. The next step is to prepare Service Function Chain.

```
PUT ./config/service-function-chain:service-function-chains/

{
    "service-function-chains": {
        "service-function-chain": [
            {
                "name": "CSR3XSF",
                "sfc-service-function": [
                    {
                        "name": "Firewall",
                        "type": "firewall"
                    },
                    {
                        "name": "Dpi",
                        "type": "dpi"
                    },
                    {
                        "name": "Qos",
                        "type": "qos"
                    }
                ]
            }
        ]
    }
}
```

Service Function Path:

```
PUT ./config/service-function-path:service-function-paths/
```

```
{
    "service-function-paths": {
        "service-function-path": [
            {
                "name": "CSR3XSF-Path",
                "service-chain-name": "CSR3XSF",
                "starting-index": 255,
                "symmetric": "true"
            }
        ]
    }
}
```

Without a classifier, there is possibility to POST RSP directly.

```
POST ./operations/rendered-service-path:create-rendered-path

{
  "input": {
      "name": "CSR3XSF-Path-RSP",
      "parent-service-function-path": "CSR3XSF-Path"
  }
}
```

The resulting configuration:

```
!
service-chain service-function-forwarder local
  ip address 10.99.150.10
!
service-chain service-function firewall
ip address 12.1.1.2
  encapsulation gre enhanced divert
!
service-chain service-function dpi
ip address 12.1.1.1
  encapsulation gre enhanced divert
!
service-chain service-function qos
ip address 12.1.1.4
  encapsulation gre enhanced divert
!
service-chain service-path 1
  service-index 255 service-function firewall
  service-index 254 service-function dpi
  service-index 253 service-function qos
  service-index 252 terminate
!
service-chain service-path 2
  service-index 255 service-function qos
  service-index 254 service-function dpi
  service-index 253 service-function firewall
  service-index 252 terminate
!
```

Service Path 1 is direct, Service Path 2 is reversed. Path numbers may vary.

Service Function Scheduling Algorithms

Overview

When creating the Rendered Service Path, the origin SFC controller chose the first available service function from a list of service function names. This may result in many issues such as overloaded service functions and a longer service path as SFC has no means to understand the status of service functions and network topology. The service function selection framework supports at least four algorithms (Random, Round Robin, Load Balancing and Shortest Path) to select the most appropriate service function when instantiating the Rendered Service Path. In addition, it is an extensible framework that allows 3rd party selection algorithm to be plugged in.

Architecture

The following figure illustrates the service function selection framework and algorithms.

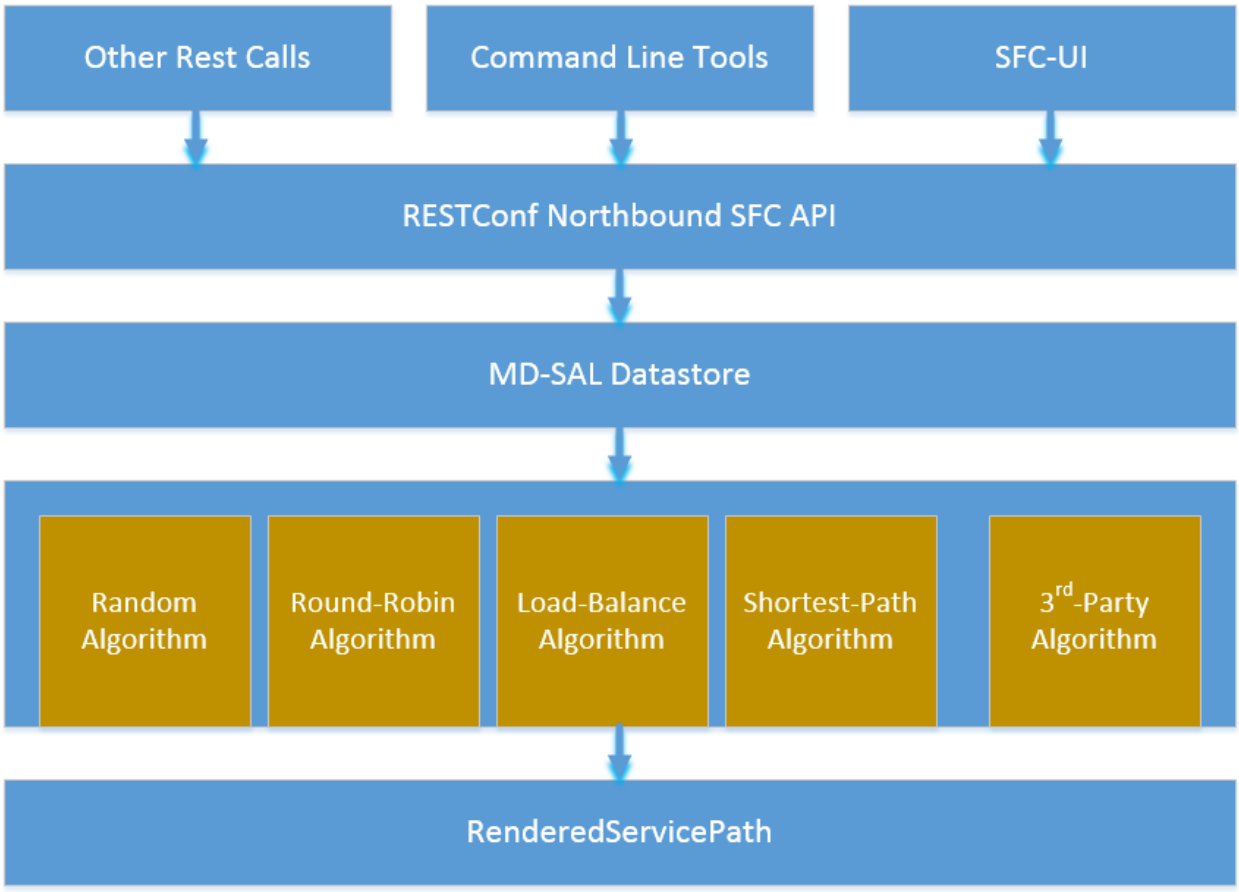


Fig. 1.101: SF Selection Architecture

A user has three different ways to select one service function selection algorithm:

1. Integrated RESTCONF Calls. OpenStack and/or other administration system could provide plugins to call the APIs to select one scheduling algorithm.
2. Command line tools. Command line tools such as curl or browser plugins such as POSTMAN (for Google Chrome) and RESTClient (for Mozilla Firefox) could select schedule algorithm by making RESTCONF calls.

3. SFC-UI. Now the SFC-UI provides an option for choosing a selection algorithm when creating a Rendered Service Path.

The RESTCONF northbound SFC API provides GUI/RESTCONF interactions for choosing the service function selection algorithm. MD-SAL data store provides all supported service function selection algorithms, and provides APIs to enable one of the provided service function selection algorithms. Once a service function selection algorithm is enabled, the service function selection algorithm will work when creating a Rendered Service Path.

Select SFs with Scheduler

Administrator could use both the following ways to select one of the selection algorithm when creating a Rendered Service Path.

- Command line tools. Command line tools includes Linux commands curl or even browser plugins such as POSTMAN(for Google Chrome) or RESTClient(for Mozilla Firefox). In this case, the following JSON content is needed at the moment: Service_function_schudule_type.json

```
{
  "service-function-scheduler-types": {
    "service-function-scheduler-type": [
      {
        "name": "random",
        "type": "service-function-scheduler-type:random",
        "enabled": false
      },
      {
        "name": "roundrobin",
        "type": "service-function-scheduler-type:round-robin",
        "enabled": true
      },
      {
        "name": "loadbalance",
        "type": "service-function-scheduler-type:load-balance",
        "enabled": false
      },
      {
        "name": "shortestpath",
        "type": "service-function-scheduler-type:shortest-path",
        "enabled": false
      }
    ]
  }
}
```

If using the Linux curl command, it could be:

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪${Service_function_schudule_type.json}'
-X PUT --user admin:admin http://localhost:8181/restconf/config/service-function-
↪scheduler-type:service-function-scheduler-types/
```

Here is also a snapshot for using the RESTClient plugin:

- SFC-UI.SFC-UI provides a drop down menu for service function selection algorithm. Here is a snapshot for the user interaction from SFC-UI when creating a Rendered Service Path.

Note: Some service function selection algorithms in the drop list are not implemented yet. Only the first three

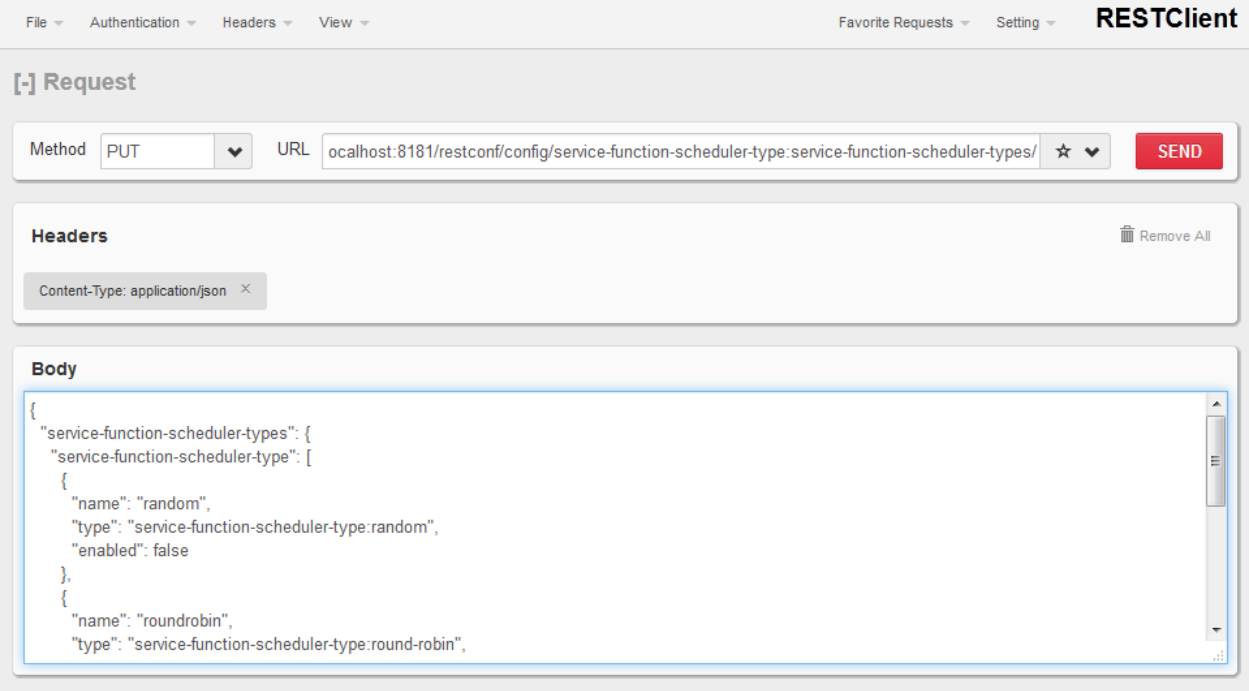


Fig. 1.102: Mozilla Firefox RESTClient

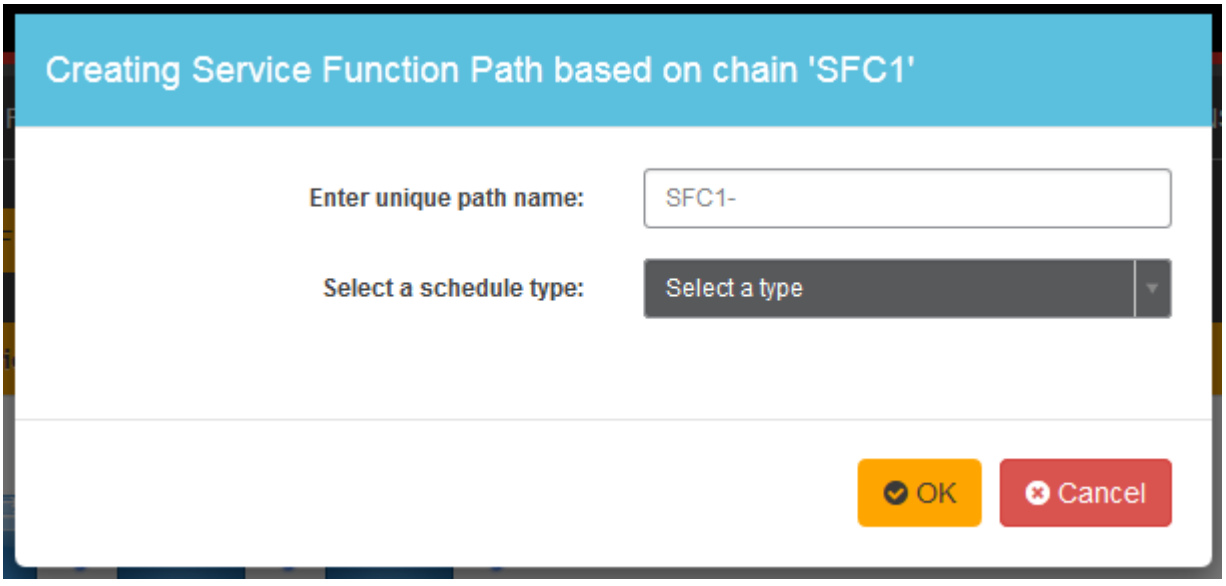


Fig. 1.103: Karaf Web UI

algorithms are committed at the moment.

Random

Select Service Function from the name list randomly.

Overview

The Random algorithm is used to select one Service Function from the name list which it gets from the Service Function Type randomly.

Prerequisites

- Service Function information are stored in datastore.
- Either no algorithm or the Random algorithm is selected.

Target Environment

The Random algorithm will work either no algorithm type is selected or the Random algorithm is selected.

Instructions

Once the plugins are installed into Karaf successfully, a user can use his favorite method to select the Random scheduling algorithm type. There are no special instructions for using the Random algorithm.

Round Robin

Select Service Function from the name list in Round Robin manner.

Overview

The Round Robin algorithm is used to select one Service Function from the name list which it gets from the Service Function Type in a Round Robin manner, this will balance workloads to all Service Functions. However, this method cannot help all Service Functions load the same workload because it's flow-based Round Robin.

Prerequisites

- Service Function information are stored in datastore.
- Round Robin algorithm is selected

Target Environment

The Round Robin algorithm will work one the Round Robin algorithm is selected.

Instructions

Once the plugins are installed into Karaf successfully, a user can use his favorite method to select the Round Robin scheduling algorithm type. There are no special instructions for using the Round Robin algorithm.

Load Balance Algorithm

Select appropriate Service Function by actual CPU utilization.

Overview

The Load Balance Algorithm is used to select appropriate Service Function by actual CPU utilization of service functions. The CPU utilization of service function obtained from monitoring information reported via NETCONF.

Prerequisites

- CPU-utilization for Service Function.
- NETCONF server.
- NETCONF client.
- Each VM has a NETCONF server and it could work with NETCONF client well.

Instructions

Set up VMs as Service Functions. enable NETCONF server in VMs. Ensure that you specify them separately. For example:

1. Set up 4 VMs include 2 SFs' type are Firewall, Others are Napt44. Name them as firewall-1, firewall-2, napt44-1, napt44-2 as Service Function. The four VMs can run either the same server or different servers.
2. Install NETCONF server on every VM and enable it. More information on NETCONF can be found on the OpenDaylight wiki here: https://wiki.opendaylight.org/view/OpenDaylight_Controller:Config:Examples:Netconf:Manual_netopeer_installation
3. Get Monitoring data from NETCONF server. These monitoring data should be get from the NETCONF server which is running in VMs. The following static XML data is an example:

static XML data like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<service-function-description-monitor-report>
  <SF-description>
    <number-of-dataports>2</number-of-dataports>
    <capabilities>
      <supported-packet-rate>5</supported-packet-rate>
      <supported-bandwidth>10</supported-bandwidth>
      <supported-ACL-number>2000</supported-ACL-number>
      <RIB-size>200</RIB-size>
      <FIB-size>100</FIB-size>
      <ports-bandwidth>
        <port-bandwidth>
          <port-id>1</port-id>
```

```
          <ipaddress>10.0.0.1</ipaddress>
          <macaddress>00:1e:67:a2:5f:f4</macaddress>
          <supported-bandwidth>20</supported-bandwidth>
        </port-bandwidth>
        <port-bandwidth>
          <port-id>2</port-id>
          <ipaddress>10.0.0.2</ipaddress>
          <macaddress>01:1e:67:a2:5f:f6</macaddress>
          <supported-bandwidth>10</supported-bandwidth>
        </port-bandwidth>
      </ports-bandwidth>
    </capabilities>
  </SF-description>
  <SF-monitoring-info>
    <liveness>true</liveness>
    <resource-utilization>
        <packet-rate-utilization>10</packet-rate-utilization>
        <bandwidth-utilization>15</bandwidth-utilization>
        <CPU-utilization>12</CPU-utilization>
        <memory-utilization>17</memory-utilization>
        <available-memory>8</available-memory>
        <RIB-utilization>20</RIB-utilization>
        <FIB-utilization>25</FIB-utilization>
        <power-utilization>30</power-utilization>
        <SF-ports-bandwidth-utilization>
          <port-bandwidth-utilization>
            <port-id>1</port-id>
            <bandwidth-utilization>20</bandwidth-utilization>
          </port-bandwidth-utilization>
          <port-bandwidth-utilization>
            <port-id>2</port-id>
            <bandwidth-utilization>30</bandwidth-utilization>
          </port-bandwidth-utilization>
        </SF-ports-bandwidth-utilization>
    </resource-utilization>
  </SF-monitoring-info>
</service-function-description-monitor-report>
```

1. Unzip SFC release tarball.
2. Run SFC: ${sfc}/bin/karaf. More information on Service Function Chaining can be found on the OpenDaylight SFC's wiki page: https://wiki.opendaylight.org/view/Service_Function_Chaining:Main

1. Deploy the SFC2 (firewall-abstract2napt44-abstract2) and click button to Create Rendered Service Path in SFC UI (http://localhost:8181/sfc/index.html).
2. Verify the Rendered Service Path to ensure the CPU utilization of the selected hop is the minimum one among all the service functions with same type. The correct RSP is firewall-1napt44-2

Shortest Path Algorithm

Select appropriate Service Function by Dijkstra's algorithm. Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph.

Overview

The Shortest Path Algorithm is used to select appropriate Service Function by actual topology.

Prerequisites

- Deployed topology (include SFFs, SFs and their links).
- Dijkstra's algorithm. More information on Dijkstra's algorithm can be found on the wiki here: http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Instructions

1. Unzip SFC release tarball.
2. Run SFC: ${sfc}/bin/karaf.
3. Depoly SFFs and SFs. import the service-function-forwarders.json and service-functions.json in UI (http://localhost:8181/sfc/index.html#/sfc/config)

service-function-forwarders.json:

```
{
  "service-function-forwarders": {
    "service-function-forwarder": [
      {
        "name": "SFF-br1",
        "service-node": "OVSDB-test01",
        "rest-uri": "http://localhost:5001",
        "sff-data-plane-locator": [
          {
            "name": "eth0",
            "service-function-forwarder-ovs:ovs-bridge": {
              "uuid": "4c3778e4-840d-47f4-b45e-0988e514d26c",
              "bridge-name": "br-tun"
            },
            "data-plane-locator": {
              "port": 5000,
              "ip": "192.168.1.1",
              "transport": "service-locator:vxlan-gpe"
            }
          }
        ],
        "service-function-dictionary": [
          {
            "sff-sf-data-plane-locator": {
               "sf-dpl-name": "sf1dpl",
               "sff-dpl-name": "sff1dpl"
            },
            "name": "napt44-1",
            "type": "napt44"
          },
          {
            "sff-sf-data-plane-locator": {
               "sf-dpl-name": "sf2dpl",
               "sff-dpl-name": "sff2dpl"
```

```
            },
            "name": "firewall-1",
            "type": "firewall"
          }
        ],
        "connected-sff-dictionary": [
          {
            "name": "SFF-br3"
          }
        ]
      },
      {
        "name": "SFF-br2",
        "service-node": "OVSDB-test01",
        "rest-uri": "http://localhost:5002",
        "sff-data-plane-locator": [
          {
            "name": "eth0",
            "service-function-forwarder-ovs:ovs-bridge": {
              "uuid": "fd4d849f-5140-48cd-bc60-6ad1f5fc0a1",
              "bridge-name": "br-tun"
            },
            "data-plane-locator": {
              "port": 5000,
              "ip": "192.168.1.2",
              "transport": "service-locator:vxlan-gpe"
            }
          }
        ],
        "service-function-dictionary": [
          {
            "sff-sf-data-plane-locator": {
               "sf-dpl-name": "sf1dpl",
               "sff-dpl-name": "sff1dpl"
            },
            "name": "napt44-2",
            "type": "napt44"
          },
          {
            "sff-sf-data-plane-locator": {
               "sf-dpl-name": "sf2dpl",
               "sff-dpl-name": "sff2dpl"
            },
            "name": "firewall-2",
            "type": "firewall"
          }
        ],
        "connected-sff-dictionary": [
          {
            "name": "SFF-br3"
          }
        ]
      },
      {
        "name": "SFF-br3",
        "service-node": "OVSDB-test01",
        "rest-uri": "http://localhost:5005",
        "sff-data-plane-locator": [
```

```
          {
            "name": "eth0",
            "service-function-forwarder-ovs:ovs-bridge": {
              "uuid": "fd4d849f-5140-48cd-bc60-6ad1f5fc0a4",
              "bridge-name": "br-tun"
            },
            "data-plane-locator": {
              "port": 5000,
              "ip": "192.168.1.2",
              "transport": "service-locator:vxlan-gpe"
            }
          }
        ],
        "service-function-dictionary": [
          {
            "sff-sf-data-plane-locator": {
               "sf-dpl-name": "sf1dpl",
               "sff-dpl-name": "sff1dpl"
            },
            "name": "test-server",
            "type": "dpi"
          },
          {
            "sff-sf-data-plane-locator": {
               "sf-dpl-name": "sf2dpl",
               "sff-dpl-name": "sff2dpl"
            },
            "name": "test-client",
            "type": "dpi"
          }
        ],
        "connected-sff-dictionary": [
          {
            "name": "SFF-br1"
          },
          {
            "name": "SFF-br2"
          }
        ]
      }
    ]
  }
}
```

service-functions.json:

```
{
  "service-functions": {
    "service-function": [
      {
        "rest-uri": "http://localhost:10001",
        "ip-mgmt-address": "10.3.1.103",
        "sf-data-plane-locator": [
          {
            "name": "preferred",
            "port": 10001,
            "ip": "10.3.1.103",
            "service-function-forwarder": "SFF-br1"
```

```
        }
      ],
      "name": "napt44-1",
      "type": "napt44"
    },
    {
      "rest-uri": "http://localhost:10002",
      "ip-mgmt-address": "10.3.1.103",
      "sf-data-plane-locator": [
        {
          "name": "master",
          "port": 10002,
          "ip": "10.3.1.103",
          "service-function-forwarder": "SFF-br2"
        }
      ],
      "name": "napt44-2",
      "type": "napt44"
    },
    {
      "rest-uri": "http://localhost:10003",
      "ip-mgmt-address": "10.3.1.103",
      "sf-data-plane-locator": [
        {
          "name": "1",
          "port": 10003,
          "ip": "10.3.1.102",
          "service-function-forwarder": "SFF-br1"
        }
      ],
      "name": "firewall-1",
      "type": "firewall"
    },
    {
      "rest-uri": "http://localhost:10004",
      "ip-mgmt-address": "10.3.1.103",
      "sf-data-plane-locator": [
        {
          "name": "2",
          "port": 10004,
          "ip": "10.3.1.101",
          "service-function-forwarder": "SFF-br2"
        }
      ],
      "name": "firewall-2",
      "type": "firewall"
    },
    {
      "rest-uri": "http://localhost:10005",
      "ip-mgmt-address": "10.3.1.103",
      "sf-data-plane-locator": [
        {
          "name": "3",
          "port": 10005,
          "ip": "10.3.1.104",
          "service-function-forwarder": "SFF-br3"
        }
      ],
```

```
        "name": "test-server",
        "type": "dpi"
      },
      {
        "rest-uri": "http://localhost:10006",
        "ip-mgmt-address": "10.3.1.103",
        "sf-data-plane-locator": [
          {
            "name": "4",
            "port": 10006,
            "ip": "10.3.1.102",
            "service-function-forwarder": "SFF-br3"
          }
        ],
        "name": "test-client",
        "type": "dpi"
      }
    ]
  }
}
```

The deployed topology like this:

```
          +----+           +----+          +----+
          |sff1|+----------|sff3|----------+|sff2|
          +----+           +----+          +----+
            |                                |
     +--------------+                  +--------------+
     |              |                  |              |
+----------+   +--------+          +----------+   +--------+
|firewall-1|   |napt44-1|          |firewall-2|   |napt44-2|
+----------+   +--------+          +----------+   +--------+
```

- Deploy the SFC2(firewall-abstract2napt44-abstract2), select "Shortest Path" as schedule type and click button to Create Rendered Service Path in SFC UI (http://localhost:8181/sfc/index.html).
- Verify the Rendered Service Path to ensure the selected hops are linked in one SFF. The correct RSP is firewall-1napt44-1 or firewall-2napt44-2. The first SF type is Firewall in Service Function Chain. So the algorithm will select first Hop randomly among all the SFs type is Firewall. Assume the first selected SF is firewall-2. All the path from firewall-1 to SF which type is Napt44 are list:
  - Path1: firewall-2 → sff2 → napt44-2
  - Path2: firewall-2 → sff2 → sff3 → sff1 → napt44-1 The shortest path is Path1, so the selected next hop is napt44-2.

Service Function Load Balancing User Guide

Overview

SFC Load-Balancing feature implements load balancing of Service Functions, rather than a one-to-one mapping between Service-Function-Forwarder and Service-Function.

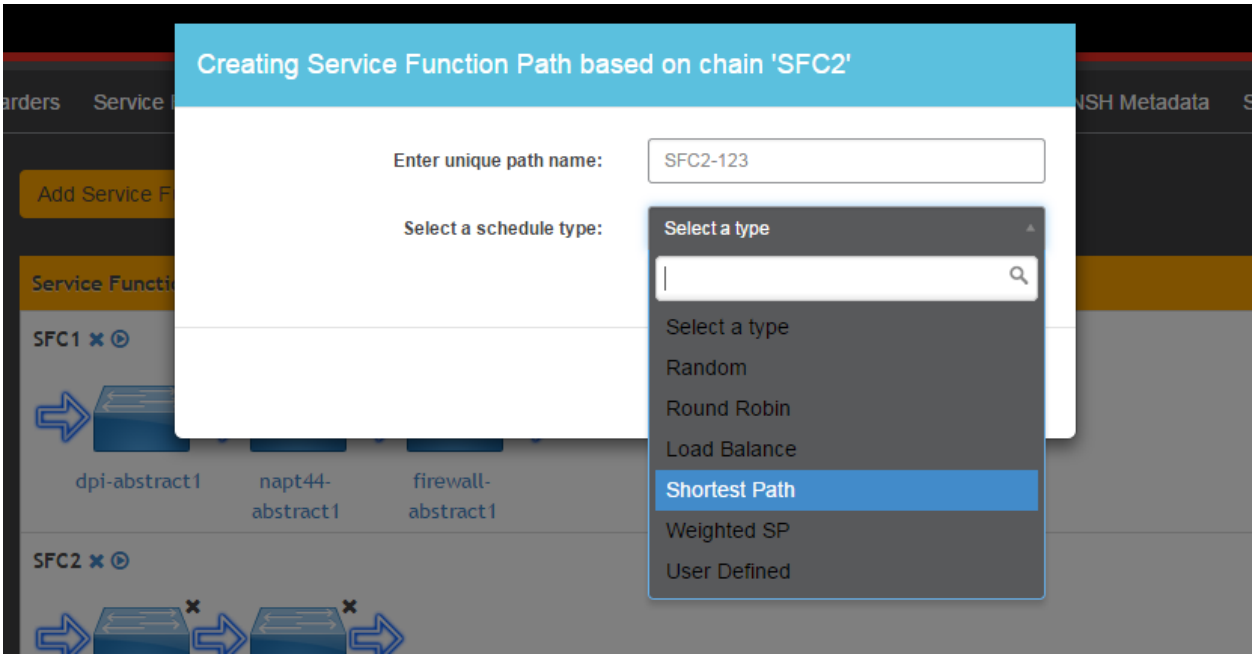


Fig. 1.104: select schedule type

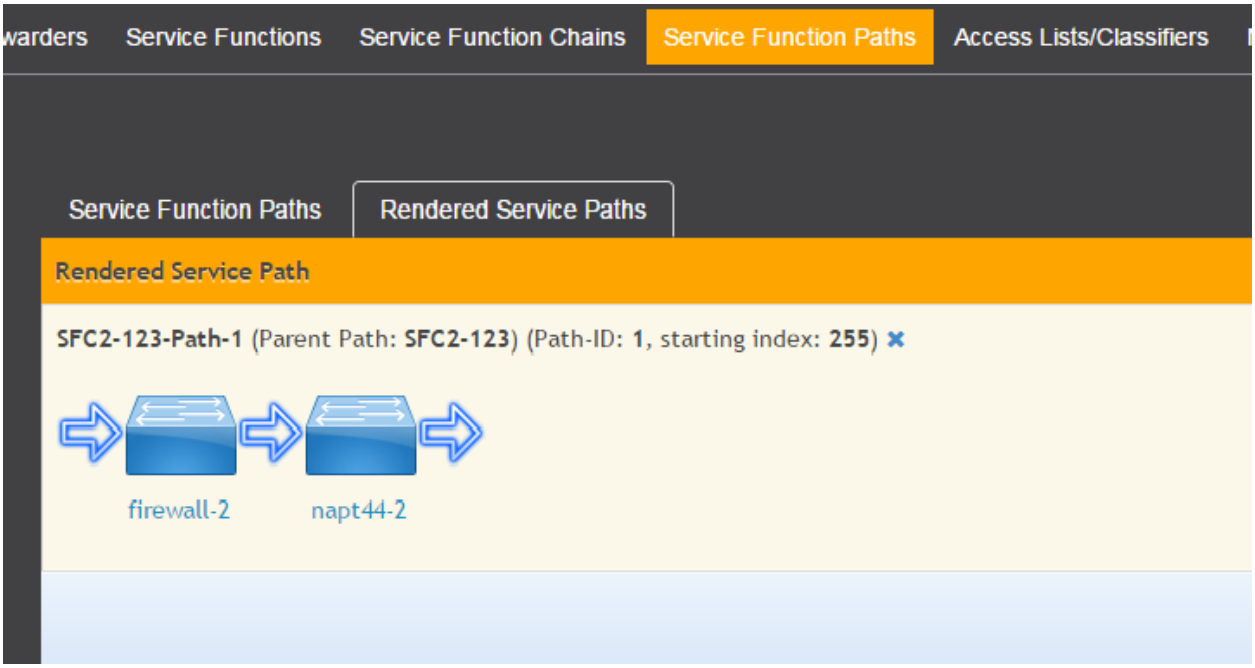


Fig. 1.105: rendered service path

Load Balancing Architecture

Service Function Groups (SFG) can replace Service Functions (SF) in the Rendered Path model. A Service Path can only be defined using SFGs or SFs, but not a combination of both.

Relevant objects in the YANG model are as follows:

1. Service-Function-Group-Algorithm:

```
Service-Function-Group-Algorithms {
    Service-Function-Group-Algorithm {
        String name
        String type
    }
}
```

```
Available types: ALL, SELECT, INDIRECT, FAST_FAILURE
```

2. Service-Function-Group:

```
Service-Function-Groups {
    Service-Function-Group {
        String name
        String serviceFunctionGroupAlgorithmName
        String type
        String groupId
        Service-Function-Group-Element {
            String service-function-name
            int index
        }
    }
}
```

3. ServiceFunctionHop: holds a reference to a name of SFG (or SF)

Tutorials

This tutorial will explain how to create a simple SFC configuration, with SFG instead of SF. In this example, the SFG will include two existing SF.

Setup SFC

For general SFC setup and scenarios, please see the SFC wiki page: https://wiki.opendaylight.org/view/Service_Function_Chaining:Main#SFC_101

Create an algorithm

POST - http://127.0.0.1:8181/restconf/config/service-function-group-algorithm:service-function-group-algorithms

```
{
    "service-function-group-algorithm": [
      {
        "name": "alg1"
        "type": "ALL"
```

```
        }
    ]
}
```

(Header "content-type": application/json)

Verify: get all algorithms

GET - http://127.0.0.1:8181/restconf/config/service-function-group-algorithm:service-function-group-algorithms

In order to delete all algorithms: DELETE - http://127.0.0.1:8181/restconf/config/service-function-group-algorithm:service-function-group-algorithms

Create a group

POST - http://127.0.0.1:8181/restconf/config/service-function-group:service-function-groups

```
{
    "service-function-group": [
    {
        "rest-uri": "http://localhost:10002",
        "ip-mgmt-address": "10.3.1.103",
        "algorithm": "alg1",
        "name": "SFG1",
        "type": "napt44",
        "sfc-service-function": [
            {
                "name":"napt44-104"
            },
            {
                "name":"napt44-103-1"
            }
        ]
      }
    ]
}
```

Verify: get all SFG's

GET - http://127.0.0.1:8181/restconf/config/service-function-group:service-function-groups

SFC Proof of Transit User Guide

Overview

Several deployments use traffic engineering, policy routing, segment routing or service function chaining (SFC) to steer packets through a specific set of nodes. In certain cases regulatory obligations or a compliance policy require to prove that all packets that are supposed to follow a specific path are indeed being forwarded across the exact set of nodes specified. I.e. if a packet flow is supposed to go through a series of service functions or network nodes, it has to be proven that all packets of the flow actually went through the service chain or collection of nodes specified by the policy. In case the packets of a flow weren't appropriately processed, a proof of transit egress device would be

required to identify the policy violation and take corresponding actions (e.g. drop or redirect the packet, send an alert etc.) corresponding to the policy.

Service Function Chaining (SFC) Proof of Transit (SFC PoT) implements Service Chaining Proof of Transit functionality on capable network devices. Proof of Transit defines mechanisms to securely prove that traffic transited the defined path. After the creation of an Rendered Service Path (RSP), a user can configure to enable SFC proof of transit on the selected RSP to effect the proof of transit.

To ensure that the data traffic follows a specified path or a function chain, meta-data is added to user traffic in the form of a header. The meta-data is based on a 'share of a secret' and provisioned by the SFC PoT configuration from ODL over a secure channel to each of the nodes in the SFC. This meta-data is updated at each of the service-hop while a designated node called the verifier checks whether the collected meta-data allows the retrieval of the secret.

The following diagram shows the overview and essentially utilizes Shamir's secret sharing algorithm, where each service is given a point on the curve and when the packet travels through each service, it collects these points (meta-data) and a verifier node tries to re-construct the curve using the collected points, thus verifying that the packet traversed through all the service functions along the chain.

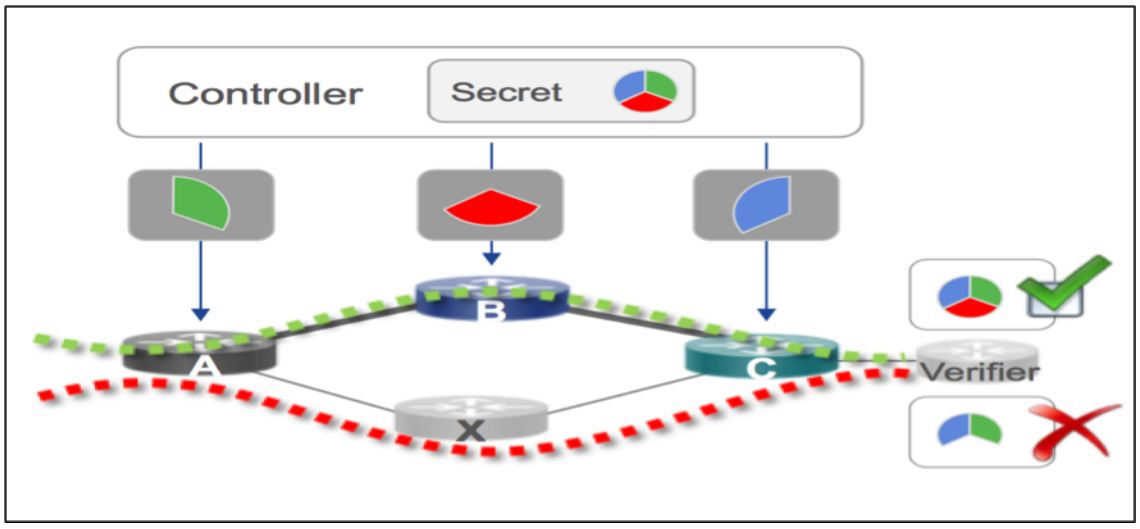


Fig. 1.106: SFC Proof of Transit overview

Transport options for different protocols includes a new TLV in SR header for Segment Routing, NSH Type-2 meta-data, IPv6 extension headers, IPv4 variants and for VXLAN-GPE. More details are captured in the following link.

In-situ OAM: https://github.com/CiscoDevNet/iOAM

Common acronyms used in the following sections:

- SF - Service Function
- SFF - Service Function Forwarder
- SFC - Service Function Chain
- SFP - Service Function Path
- RSP - Rendered Service Path
- SFC PoT - Service Function Chain Proof of Transit

SFC Proof of Transit Architecture

SFC PoT feature is implemented as a two-part implementation with a north-bound handler that augments the RSP while a south-bound renderer auto-generates the required parameters and passes it on to the nodes that belong to the SFC.

The north-bound feature is enabled via odl-sfc-pot feature while the south-bound renderer is enabled via the odl-sfc-pot-netconf-renderer feature. For the purposes of SFC PoT handling, both features must be installed.

RPC handlers to augment the RSP are part of `SfcPotRpc` while the RSP augmentation to enable or disable SFC PoT feature is done via `SfcPotRspProcessor`.

SFC Proof of Transit entities

In order to implement SFC Proof of Transit for a service function chain, an RSP is a pre-requisite to identify the SFC to enable SFC PoT on. SFC Proof of Transit for a particular RSP is enabled by an RPC request to the controller along with necessary parameters to control some of the aspects of the SFC Proof of Transit process.

The RPC handler identifies the RSP and adds PoT feature meta-data like enable/disable, number of PoT profiles, profiles refresh parameters etc., that directs the south-bound renderer appropriately when RSP changes are noticed via call-backs in the renderer handlers.

Administering SFC Proof of Transit

To use the SFC Proof of Transit Karaf, at least the following Karaf features must be installed:

- odl-sfc-model
- odl-sfc-provider
- odl-sfc-netconf
- odl-restconf
- odl-netconf-topology
- odl-netconf-connector-all
- odl-sfc-pot

Please note that the odl-sfc-pot-netconf-renderer or other renderers in future must be installed for the feature to take full-effect. The details of the renderer features are described in other parts of this document.

SFC Proof of Transit Tutorial

Overview

This tutorial is a simple example how to configure Service Function Chain Proof of Transit using SFC POT feature.

Preconditions

To enable a device to handle SFC Proof of Transit, it is expected that the NETCONF node device advertise capability as under ioam-sb-pot.yang present under sfc-model/src/main/yang folder. It is also expected that base NETCONF support be enabled and its support capability advertised as capabilities.

NETCONF support:`urn:ietf:params:netconf:base:1.0`

PoT support: `(urn:cisco:params:xml:ns:yang:sfc-ioam-sb-pot?revision=2017-01-12)sfc-ioam-sb-pot`

It is also expected that the devices are netconf mounted and available in the topology-netconf store.

Instructions

When SFC Proof of Transit is installed, all netconf nodes in topology-netconf are processed and all capable nodes with accessible mountpoints are cached.

First step is to create the required RSP as is usually done using RSP creation steps in SFC main.

Once RSP name is available it is used to send a POST RPC to the controller similar to below:

POST - http://ODL-IP:8181/restconf/operations/sfc-ioam-nb-pot:enable-sfc-ioam-pot-rendered-path/

```
{
    "input":
    {
        "sfc-ioam-pot-rsp-name": "sfc-path-3sf3sff",
        "ioam-pot-enable":true,
        "ioam-pot-num-profiles":2,
        "ioam-pot-bit-mask":"bits32",
        "refresh-period-time-units":"milliseconds",
        "refresh-period-value":5000
    }
}
```

The following can be used to disable the SFC Proof of Transit on an RSP which disables the PoT feature.

POST - http://ODL-IP:8181/restconf/operations/sfc-ioam-nb-pot:disable-sfc-ioam-pot-rendered-path/

```
{
    "input":
    {
        "sfc-ioam-pot-rsp-name": "sfc-path-3sf3sff",
    }
}
```

SFC PoT NETCONF Renderer User Guide

Overview

The SFC Proof of Transit (PoT) NETCONF renderer implements SFC Proof of Transit functionality on NETCONF-capable devices, that have advertised support for in-situ OAM (iOAM) support.

It listens for an update to an existing RSP with enable or disable proof of transit support and adds the auto-generated SFC PoT configuration parameters to all the SFC hop nodes. The last node in the SFC is configured as a verifier node to allow SFC PoT process to be completed.

Common acronyms are used as below:

- SF - Service Function
- SFC - Service Function Chain

- RSP - Rendered Service Path
- SFF - Service Function Forwarder

Mapping to SFC entities

The renderer module listens to RSP updates in `SfcPotNetconfRSPListener` and triggers configuration generation in `SfcPotNetconfIoam` class. Node arrival and leaving are managed via `SfcPotNetconfNodeManager` and `SfcPotNetconfNodeListener`. In addition there is a timer thread that runs to generate configuration periodically to refresh the profiles in the nodes that are part of the SFC.

Administering SFC PoT NETCONF Renderer

To use the SFC Proof of Transit Karaf, the following Karaf features must be installed:

- odl-sfc-model
- odl-sfc-provider
- odl-sfc-netconf
- odl-restconf-all
- odl-netconf-topology
- odl-netconf-connector-all
- odl-sfc-pot
- odl-sfc-pot-netconf-renderer

SFC PoT NETCONF Renderer Tutorial

Overview

This tutorial is a simple example how to enable SFC PoT on NETCONF-capable devices.

Preconditions

The NETCONF-capable device will have to support sfc-ioam-sb-pot.yang file.

It is expected that a NETCONF-capable VPP device has Honeycomb (Hc2vpp) Java-based agent that helps to translate between NETCONF and VPP internal APIs.

More details are here: In-situ OAM: https://github.com/CiscoDevNet/iOAM

Steps

When the SFC PoT NETCONF renderer module is installed, all NETCONF nodes in topology-netconf are processed and all sfc-ioam-sb-pot yang capable nodes with accessible mountpoints are cached.

The first step is to create RSP for the SFC as per SFC guidelines above.

Enable SFC PoT is done on the RSP via RESTCONF to the ODL as outlined above.

Internally, the NETCONF renderer will act on the callback to a modified RSP that has PoT enabled.

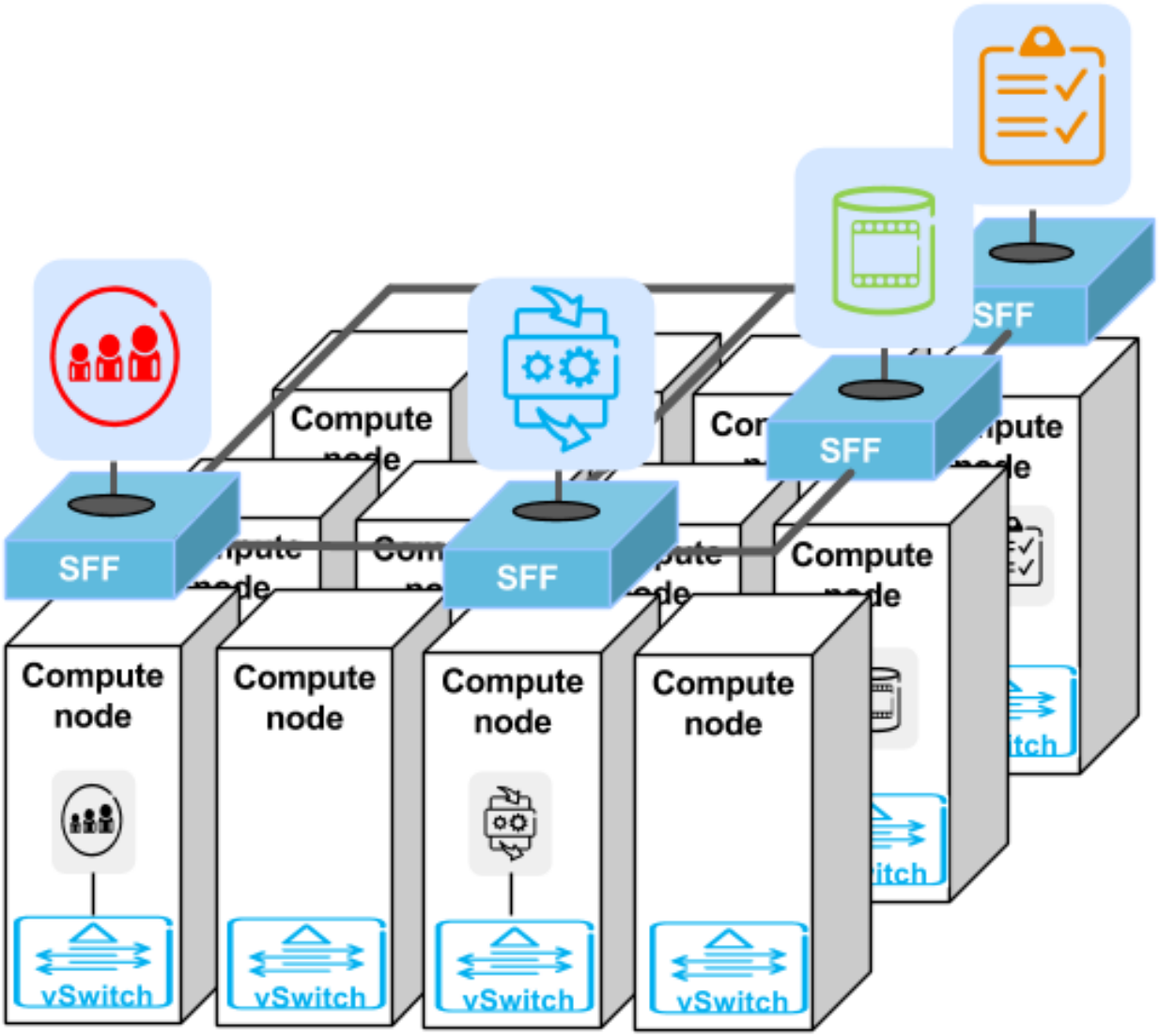
In-situ OAM algorithms for auto-generation of SFC PoT parameters are generated automatically and sent to these nodes via NETCONF.

Logical Service Function Forwarder

Overview

Rationale

When the current SFC is deployed in a cloud environment, it is assumed that each switch connected to a Service Function is configured as a Service Function Forwarder and each Service Function is connected to its Service Function Forwarder depending on the Compute Node where the Virtual Machine is located.

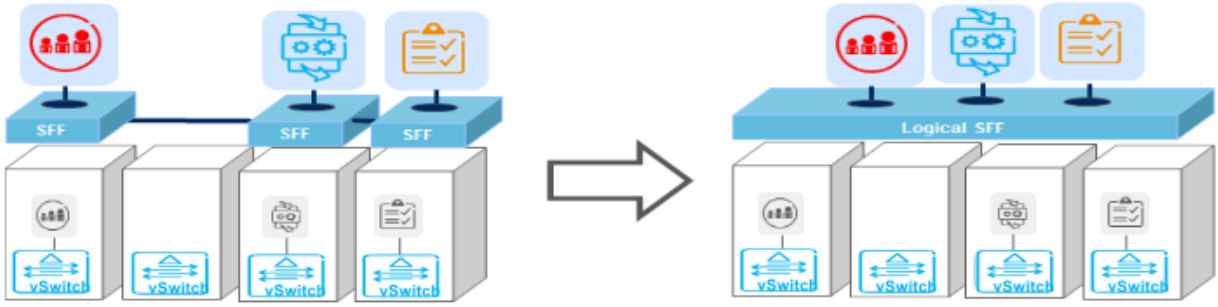


As shown in the picture above, this solution allows the basic cloud use cases to be fulfilled, as for example, the ones

required in OPNFV Brahmaputra, however, some advanced use cases like the transparent migration of VMs can not be implemented. The Logical Service Function Forwarder enables the following advanced use cases:

1. Service Function mobility without service disruption
2. Service Functions load balancing and failover

As shown in the picture below, the Logical Service Function Forwarder concept extends the current SFC northbound API to provide an abstraction of the underlying Data Center infrastructure. The Data Center underlaying network can be abstracted by a single SFF. This single SFF uses the logical port UUID as data plane locator to connect SFs globally and in a location-transparent manner. SFC makes use of Genius project to track the location of the SF's logical ports.



The SFC internally distributes the necessary flow state over the relevant switches based on the internal Data Center topology and the deployment of SFs.

Changes in data model

The Logical Service Function Forwarder concept extends the current SFC northbound API to provide an abstraction of the underlying Data Center infrastructure.

The Logical SFF simplifies the configuration of the current SFC data model by reducing the number of parameters to be be configured in every SFF, since the controller will discover those parameters by interacting with the services offered by the Genius project.

The following picture shows the Logical SFF data model. The model gets simplified as most of the configuration parameters of the current SFC data model are discovered in runtime. The complete YANG model can be found here logical SFF model.

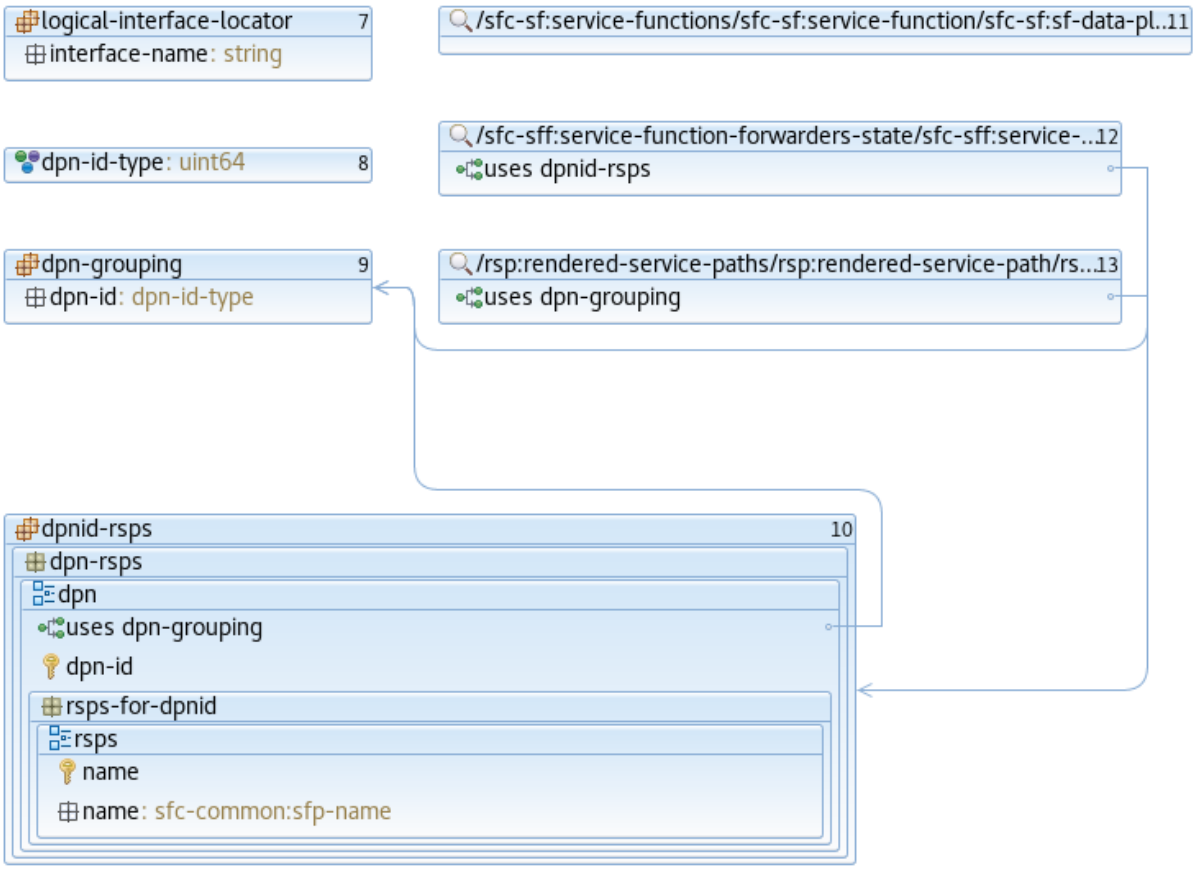
How to configure the Logical SFF

The following are examples to configure the Logical SFF:

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪{JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/config/restconf/
↪config/service-function:service-functions/
```

Service Functions JSON.

```
{
"service-functions": {
    "service-function": [
        {
            "name": "firewall-1",
            "type": "firewall",
```


logical-interface-locator 7
interface-name: string
/sfc-sf:service-functions/sfc-sf:service-function/sfc-sf:sf-data-pl..11
dpn-id-type: uint64 8
/sfc-sff:service-function-forwarders-state/sfc-sff:service-...12
uses dpnid-rsps
dpn-grouping 9
dpn-id: dpn-id-type
/rsp:rendered-service-paths/rsp:rendered-service-path/rs...13
uses dpn-grouping
dpnid-rsps 10
dpn-rsps
dpn
uses dpn-grouping
dpn-id
rsps-for-dpnid
rsps
name
name: sfc-common:sfp-name

```
            "sf-data-plane-locator": [
                {
                    "name": "firewall-dpl",
                    "interface-name": "eccb57ae-5a2e-467f-823e-45d7bb2a6a9a",
                    "transport": "service-locator:eth-nsh",
                    "service-function-forwarder": "sfflogical1"

                }
            ]
        },
        {
            "name": "dpi-1",
            "type": "dpi",
            "sf-data-plane-locator": [
                {
                    "name": "dpi-dpl",
                    "interface-name": "df15ac52-e8ef-4e9a-8340-ae0738aba0c0",
                    "transport": "service-locator:eth-nsh",
                    "service-function-forwarder": "sfflogical1"
                }
            ]
        }
    ]
}
}
```

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪{JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/config/service-
↪function-forwarder:service-function-forwarders/
```

Service Function Forwarders JSON.

```
{
"service-function-forwarders": {
    "service-function-forwarder": [
        {
            "name": "sfflogical1"
        }
    ]
}
}
```

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪{JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/config/service-
↪function-chain:service-function-chains/
```

Service Function Chains JSON.

```
{
"service-function-chains": {
    "service-function-chain": [
        {
            "name": "SFC1",
            "sfc-service-function": [
                {
                    "name": "dpi-abstract1",
                    "type": "dpi"
```

```
                },
                {
                    "name": "firewall-abstract1",
                    "type": "firewall"
                }
            ]
        },
        {
            "name": "SFC2",
            "sfc-service-function": [
                {
                    "name": "dpi-abstract1",
                    "type": "dpi"
                }
            ]
        }
    ]
}
}
```

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪{JSON}' -X PUT --user admin:admin http://localhost:8182/restconf/config/service-
↪function-chain:service-function-paths/
```

Service Function Paths JSON.

```
{
"service-function-paths": {
    "service-function-path": [
        {
            "name": "SFP1",
            "service-chain-name": "SFC1",
            "starting-index": 255,
            "symmetric": "true",
            "context-metadata": "NSH1",
            "transport-type": "service-locator:vxlan-gpe"

        }
    ]
}
}
```

As a result of above configuration, OpenDaylight renders the needed flows in all involved SFFs. Those flows implement:

- Two Rendered Service Paths:
  - dpi-1 (SF1), firewall-1 (SF2)
  - firewall-1 (SF2), dpi-1 (SF1)
- The communication between SFFs and SFs based on eth-nsh
- The communication between SFFs based on vxlan-gpe

The following picture shows a topology and traffic flow (in green) which corresponds to the above configuration.

The Logical SFF functionality allows OpenDaylight to find out the SFFs holding the SFs involved in a path. In this example the SFFs affected are Node3 and Node4 thus the controller renders the flows containing NSH parameters just in those SFFs.

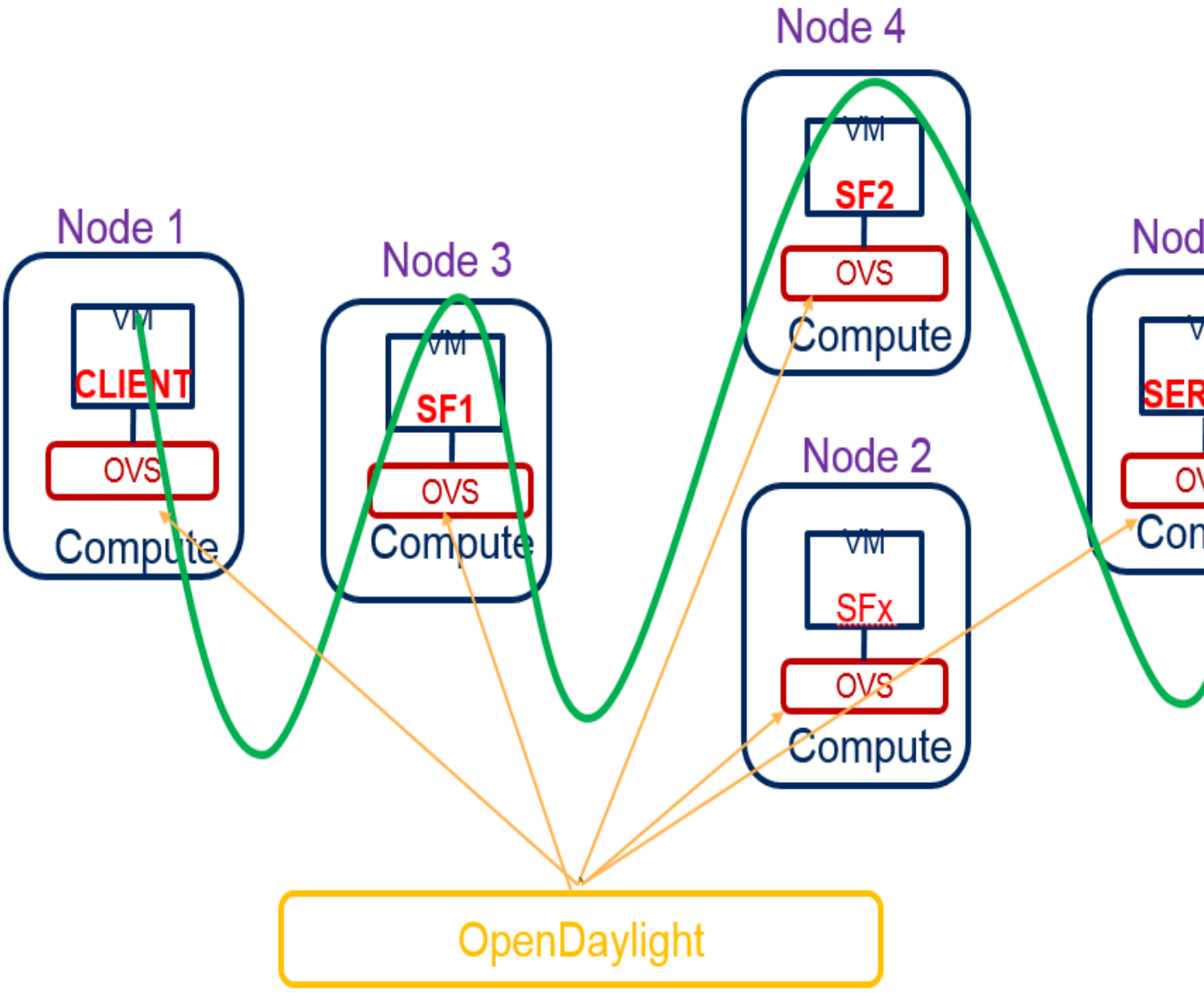


Fig. 1.107: Logical SFF Example

Here you have the new flows rendered in Node3 and Node4 which implement the NSH protocol. Every Rendered Service Path is represented by an NSP value. We provisioned a symmetric RSP so we get two NSPs: 8388613 and 5. Node3 holds the first SF of NSP 8388613 and the last SF of NSP 5. Node 4 holds the first SF of NSP 5 and the last SF of NSP 8388613. Both Node3 and Node4 will pop the NSH header when the received packet has gone through the last SF of its path.

Rendered flows Node 3

```
cookie=0x14, duration=59.264s, table=83, n_packets=0, n_bytes=0, priority=250,nsp=5␣
↪actions=goto_table:86
cookie=0x14, duration=59.194s, table=83, n_packets=0, n_bytes=0, priority=250,
↪nsp=8388613 actions=goto_table:86
cookie=0x14, duration=59.257s, table=86, n_packets=0, n_bytes=0, priority=550,nsi=254,
↪nsp=5 actions=load:0x8e0a37cc9094->NXM_NX_ENCAP_ETH_SRC[],load:0x6ee006b4c51e->NXM_
↪NX_ENCAP_ETH_DST[],goto_table:87
cookie=0x14, duration=59.189s, table=86, n_packets=0, n_bytes=0, priority=550,nsi=255,
↪nsp=8388613 actions=load:0x8e0a37cc9094->NXM_NX_ENCAP_ETH_SRC[],load:0x6ee006b4c51e-
↪>NXM_NX_ENCAP_ETH_DST[],goto_table:87
cookie=0xba5eba1100000203, duration=59.213s, table=87, n_packets=0, n_bytes=0,␣
↪priority=650,nsi=253,nsp=5 actions=pop_nsh,set_field:6e:e0:06:b4:c5:1e->eth_src,
↪resubmit(,17)
cookie=0xba5eba1100000201, duration=59.213s, table=87, n_packets=0, n_bytes=0,␣
↪priority=650,nsi=254,nsp=5 actions=load:0x800->NXM_NX_REG6[],resubmit(,220)
cookie=0xba5eba1100000201, duration=59.188s, table=87, n_packets=0, n_bytes=0,␣
↪priority=650,nsi=255,nsp=8388613 actions=load:0x800->NXM_NX_REG6[],resubmit(,220)
cookie=0xba5eba1100000201, duration=59.182s, table=87, n_packets=0, n_bytes=0,␣
↪priority=650,nsi=254,nsp=8388613 actions=set_field:0->tun_id,output:6
```

Rendered Flows Node 4

```
cookie=0x14, duration=69.040s, table=83, n_packets=0, n_bytes=0, priority=250,nsp=5␣
↪actions=goto_table:86
cookie=0x14, duration=69.008s, table=83, n_packets=0, n_bytes=0, priority=250,
↪nsp=8388613 actions=goto_table:86
cookie=0x14, duration=69.040s, table=86, n_packets=0, n_bytes=0, priority=550,nsi=255,
↪nsp=5 actions=load:0xbea93873f4fa->NXM_NX_ENCAP_ETH_SRC[],load:0x214845ea85d->NXM_
↪NX_ENCAP_ETH_DST[],goto_table:87
cookie=0x14, duration=69.005s, table=86, n_packets=0, n_bytes=0, priority=550,nsi=254,
↪nsp=8388613 actions=load:0xbea93873f4fa->NXM_NX_ENCAP_ETH_SRC[],load:0x214845ea85d->
↪NXM_NX_ENCAP_ETH_DST[],goto_table:87
cookie=0xba5eba1100000201, duration=69.029s, table=87, n_packets=0, n_bytes=0,␣
↪priority=650,nsi=255,nsp=5 actions=load:0x1100->NXM_NX_REG6[],resubmit(,220)
cookie=0xba5eba1100000201, duration=69.029s, table=87, n_packets=0, n_bytes=0,␣
↪priority=650,nsi=254,nsp=5 actions=set_field:0->tun_id,output:1
cookie=0xba5eba1100000201, duration=68.999s, table=87, n_packets=0, n_bytes=0,␣
↪priority=650,nsi=254,nsp=8388613 actions=load:0x1100->NXM_NX_REG6[],resubmit(,220)
cookie=0xba5eba1100000203, duration=68.996s, table=87, n_packets=0, n_bytes=0,␣
↪priority=650,nsi=253,nsp=8388613 actions=pop_nsh,set_field:02:14:84:5e:a8:5d->eth_
↪src,resubmit(,17)
```

An interesting scenario to show the Logical SFF strength is the migration of a SF from a compute node to another. The OpenDaylight will learn the new topology by itself, then it will re-render the new flows to the new SFFs affected.

In our example, SF2 is moved from Node4 to Node2 then OpenDaylight removes NSH specific flows from Node4 and puts them in Node2. Check below flows showing this effect. Now Node3 keeps holding the first SF of NSP 8388613 and the last SF of NSP 5; but Node2 becomes the new holder of the first SF of NSP 5 and the last SF of NSP 8388613.

Rendered Flows Node 3 After Migration

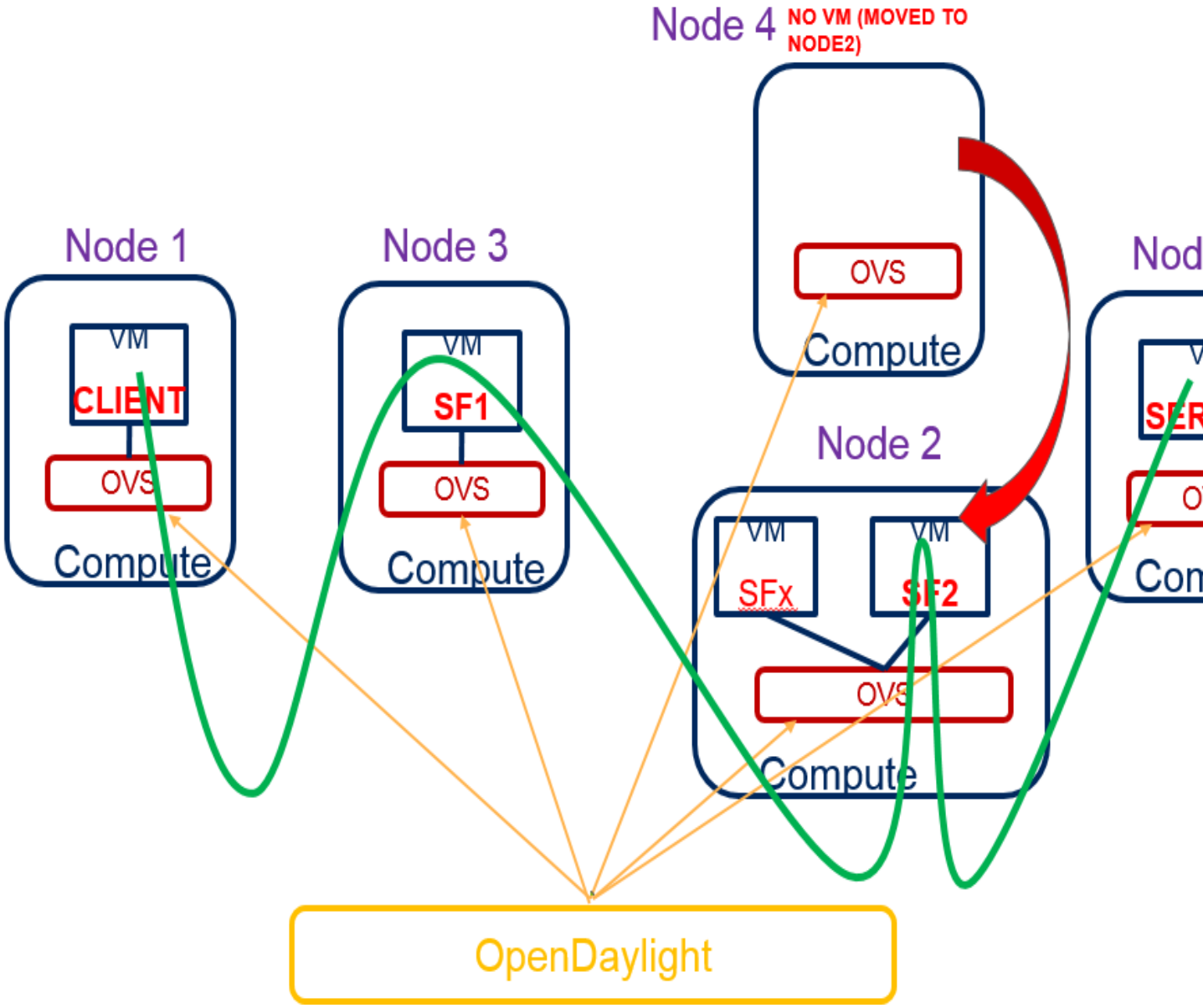


Fig. 1.108: Logical SFF - SF Migration Example

```
cookie=0x14, duration=64.044s, table=83, n_packets=0, n_bytes=0, priority=250,nsp=5␣
↪actions=goto_table:86
cookie=0x14, duration=63.947s, table=83, n_packets=0, n_bytes=0, priority=250,
↪nsp=8388613 actions=goto_table:86
cookie=0x14, duration=64.044s, table=86, n_packets=0, n_bytes=0, priority=550,nsi=254,
↪nsp=5 actions=load:0x8e0a37cc9094->NXM_NX_ENCAP_ETH_SRC[],load:0x6ee006b4c51e->NXM_
↪NX_ENCAP_ETH_DST[],goto_table:87
cookie=0x14, duration=63.947s, table=86, n_packets=0, n_bytes=0, priority=550,nsi=255,
↪nsp=8388613 actions=load:0x8e0a37cc9094->NXM_NX_ENCAP_ETH_SRC[],load:0x6ee006b4c51e-
↪>NXM_NX_ENCAP_ETH_DST[],goto_table:87
cookie=0xba5eba1100000201, duration=64.034s, table=87, n_packets=0, n_bytes=0,␣
↪priority=650,nsi=254,nsp=5 actions=load:0x800->NXM_NX_REG6[],resubmit(,220)
cookie=0xba5eba1100000203, duration=64.034s, table=87, n_packets=0, n_bytes=0,␣
↪priority=650,nsi=253,nsp=5 actions=pop_nsh,set_field:6e:e0:06:b4:c5:1e->eth_src,
↪resubmit(,17)
cookie=0xba5eba1100000201, duration=63.947s, table=87, n_packets=0, n_bytes=0,␣
↪priority=650,nsi=255,nsp=8388613 actions=load:0x800->NXM_NX_REG6[],resubmit(,220)
cookie=0xba5eba1100000201, duration=63.942s, table=87, n_packets=0, n_bytes=0,␣
↪priority=650,nsi=254,nsp=8388613 actions=set_field:0->tun_id,output:2
```

Rendered Flows Node 2 After Migration

```
cookie=0x14, duration=56.856s, table=83, n_packets=0, n_bytes=0, priority=250,nsp=5␣
↪actions=goto_table:86
cookie=0x14, duration=56.755s, table=83, n_packets=0, n_bytes=0, priority=250,
↪nsp=8388613 actions=goto_table:86
cookie=0x14, duration=56.847s, table=86, n_packets=0, n_bytes=0, priority=550,nsi=255,
↪nsp=5 actions=load:0xbea93873f4fa->NXM_NX_ENCAP_ETH_SRC[],load:0x214845ea85d->NXM_
↪NX_ENCAP_ETH_DST[],goto_table:87
cookie=0x14, duration=56.755s, table=86, n_packets=0, n_bytes=0, priority=550,nsi=254,
↪nsp=8388613 actions=load:0xbea93873f4fa->NXM_NX_ENCAP_ETH_SRC[],load:0x214845ea85d->
↪NXM_NX_ENCAP_ETH_DST[],goto_table:87
cookie=0xba5eba1100000201, duration=56.823s, table=87, n_packets=0, n_bytes=0,␣
↪priority=650,nsi=255,nsp=5 actions=load:0x1100->NXM_NX_REG6[],resubmit(,220)
cookie=0xba5eba1100000201, duration=56.823s, table=87, n_packets=0, n_bytes=0,␣
↪priority=650,nsi=254,nsp=5 actions=set_field:0->tun_id,output:4
cookie=0xba5eba1100000201, duration=56.755s, table=87, n_packets=0, n_bytes=0,␣
↪priority=650,nsi=254,nsp=8388613 actions=load:0x1100->NXM_NX_REG6[],resubmit(,220)
cookie=0xba5eba1100000203, duration=56.750s, table=87, n_packets=0, n_bytes=0,␣
↪priority=650,nsi=253,nsp=8388613 actions=pop_nsh,set_field:02:14:84:5e:a8:5d->eth_
↪src,resubmit(,17)
```

Rendered Flows Node 4 After Migration

```
-- No flows for NSH processing --
```

Classifier impacts

As previously mentioned, in the *Logical SFF rationale*, the Logical SFF feature relies on Genius to get the dataplane IDs of the OpenFlow switches, in order to properly steer the traffic through the chain.

Since one of the classifier's objectives is to steer the packets *into* the SFC domain, the classifier has to be aware of where the first Service Function is located - if it migrates somewhere else, the classifier table has to be updated accordingly, thus enabling the seemless migration of Service Functions.

For this feature, mobility of the client VM is out of scope, and should be managed by its high-availability module, or VNF manager.

Keep in mind that classification *always* occur in the compute-node where the client VM (i.e. traffic origin) is running.

How to attach the classifier to a Logical SFF

In order to leverage this functionality, the classifier has to be configured using a Logical SFF as an attachment-point, specifying within it the neutron port to classify.

The following examples show how to configure an ACL, and a classifier having a Logical SFF as an attachment-point:

Configure an ACL

The following ACL enables traffic intended for port 80 within the subnetwork 192.168.2.0/24, for RSP1 and RSP1-Reverse.

```
{
  "access-lists": {
    "acl": [
      {
        "acl-name": "ACL1",
        "acl-type": "ietf-access-control-list:ipv4-acl",
        "access-list-entries": {
          "ace": [
            {
              "rule-name": "ACE1",
              "actions": {
                "service-function-acl:rendered-service-path": "RSP1"
              },
              "matches": {
                "destination-ipv4-network": "192.168.2.0/24",
                "source-ipv4-network": "192.168.2.0/24",
                "protocol": "6",
                "source-port-range": {
                    "lower-port": 0
                },
                "destination-port-range": {
                    "lower-port": 80
                }
              }
            }
          ]
        }
      },
      {
        "acl-name": "ACL2",
        "acl-type": "ietf-access-control-list:ipv4-acl",
        "access-list-entries": {
          "ace": [
            {
              "rule-name": "ACE2",
              "actions": {
                "service-function-acl:rendered-service-path": "RSP1-Reverse"
              },
              "matches": {
                "destination-ipv4-network": "192.168.2.0/24",
                "source-ipv4-network": "192.168.2.0/24",
                "protocol": "6",
                "source-port-range": {
                    "lower-port": 80
```

```
                },
                "destination-port-range": {
                    "lower-port": 0
                }
              }
            }
          ]
        }
      }
    ]
  }
}
```

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪{JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/config/ietf-access-
↪control-list:access-lists/
```

Configure a classifier JSON

The following JSON provisions a classifier, having a Logical SFF as an attachment point. The value of the field 'interface' is where you indicate the neutron ports of the VMs you want to classify.

```
{
  "service-function-classifiers": {
    "service-function-classifier": [
      {
        "name": "Classifier1",
        "scl-service-function-forwarder": [
          {
            "name": "sfflogical1",
            "interface": "09a78ba3-78ba-40f5-a3ea-1ce708367f2b"
          }
        ],
        "acl": {
            "name": "ACL1",
            "type": "ietf-access-control-list:ipv4-acl"
         }
      }
    ]
  }
}
```

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$
↪{JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/config/service-
↪function-classifier:service-function-classifiers/
```

SFC pipeline impacts

After binding SFC service with a particular interface by means of Genius, as explained in the *Genius User Guide*, the entry point in the SFC pipeline will be table 82 (SFC_TRANSPORT_CLASSIFIER_TABLE), and from that point, packet processing will be similar to the *SFC OpenFlow pipeline*, just with another set of specific tables for the SFC service.

This picture shows the SFC pipeline after service integration with Genius:

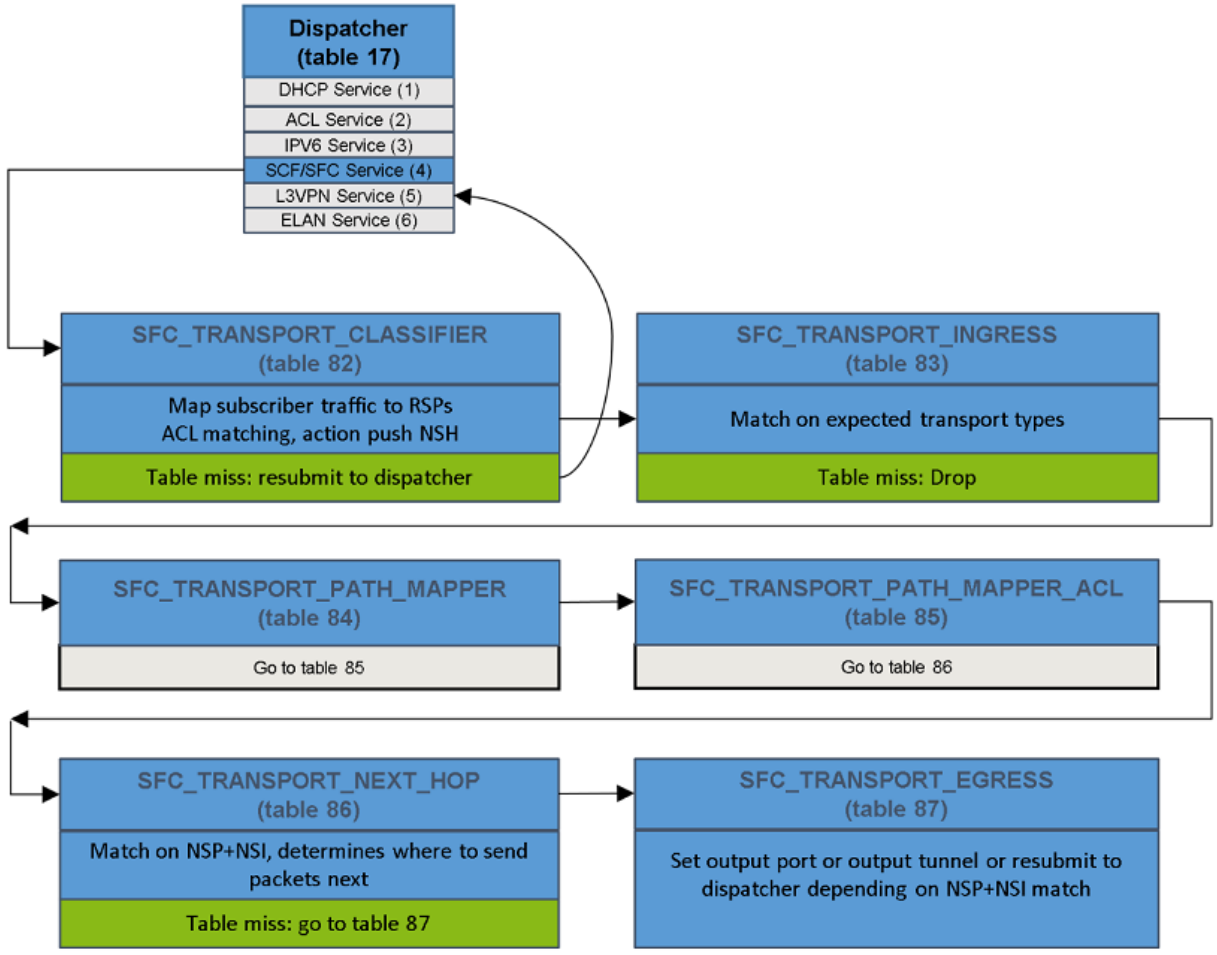


Fig. 1.109: SFC Logical SFF OpenFlow pipeline

SNMP Plugin User Guide

Installing Feature

The SNMP Plugin can be installed using a single karaf feature: **odl-snmp-plugin**

After starting Karaf:

- Install the feature: **feature:install odl-snmp-plugin**
- Expose the northbound API: **feature:install odl-restconf**

Northbound APIs

There are two exposed northbound APIs: snmp-get & snmp-set

SNMP GET

Default URL: http://localhost:8181/restconf/operations/snmp:snmp-get

POST Input

Field Name	Type	Description	Example	Required?
ip-address	Ipv4 Address	The IPv4 Address of the desired network node	10.86.3.13	Yes
oid	String	The Object Identifier of the desired MIB table/object	1.3.6.1.2.1.1.1	Yes
get-type	ENUM (GET, GET-NEXT, GET-BULK, GET-WALK)	The type of get request to send	GET-BULK	Yes
community	String	The community string to use for the SNMP request	private	No. (Default: public)

Example.

```
{
    "input": {
        "ip-address": "10.86.3.13",
        "oid" : "1.3.6.1.2.1.1.1",
        "get-type" : "GET-BULK",
        "community" : "private"
    }
}
```

POST Output

Field Name	Type	Description
results	List of { “value” : String } pairs	The results of the SNMP query

Example.

```
{
    "snmp:results": [
        {
            "value": "Ethernet0/0/0",
            "oid": "1.3.6.1.2.1.2.2.1.2.1"
        },
        {
            "value": "FastEthernet0/0/0",
            "oid": "1.3.6.1.2.1.2.2.1.2.2"
        },
        {
            "value": "GigabitEthernet0/0/0",
            "oid": "1.3.6.1.2.1.2.2.1.2.3"
        }
    ]
}
```

SNMP SET

Default URL: http://localhost:8181/restconf/operations/snmp:snmp-set

POST Input

Field Name	Type	Description	Example	Required?
ip-address	Ipv4 Address	The Ipv4 address of the desired network node	10.86.3.13	Yes
oid	String	The Object Identifier of the desired MIB object	1.3.6.2.1.1.1	Yes
value	String	The value to set on the network device	“Hello World”	Yes
community	String	The community string to use for the SNMP request	private	No. (Default: public)

Example.

```
{
    "input": {
        "ip-address": "10.86.3.13",
        "oid" : "1.3.6.1.2.1.1.1.0",
        "value" : "Sample description",
        "community" : "private"
    }
}
```

POST Output

On a successful SNMP-SET, no output is presented, just a HTTP status of 200.

Errors

If any errors happen in the set request, you will be presented with an error message in the output.

For example, on a failed set request you may see an error like:

```
{
    "errors": {
        "error": [
            {
                "error-type": "application",
                "error-tag": "operation-failed",
                "error-message": "SnmpSET failed with error status: 17, error index:␣
↪1. StatusText: Not writable"
            }
        ]
    }
}
```

which corresponds to Error status 17 in the SNMPv2 RFC: https://tools.ietf.org/html/rfc1905.

SNMP4SDN User Guide

Overview

We propose a southbound plugin that can control the off-the-shelf commodity Ethernet switches for the purpose of building SDN using Ethernet switches. For Ethernet switches, forwarding table, VLAN table, and ACL are where one can install flow configuration on, and this is done via SNMP and CLI in the proposed plugin. In addition, some settings required for Ethernet switches in SDN, e.g., disabling STP and flooding, are proposed.

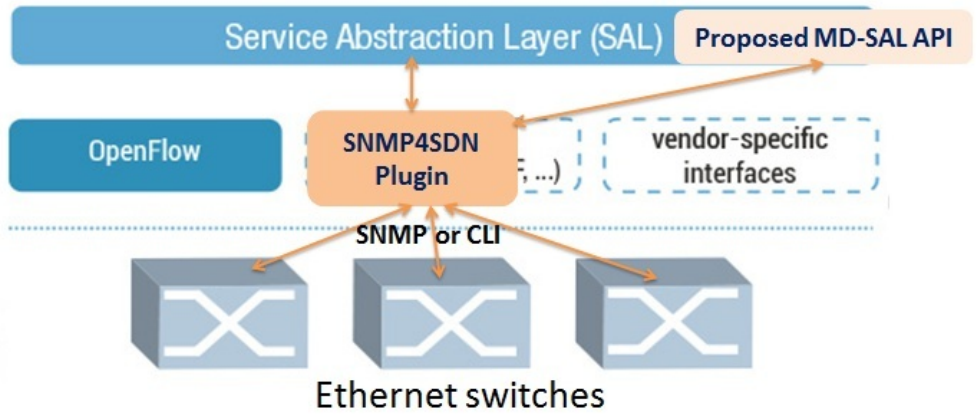


Fig. 1.110: SNMP4SDN as an OpenDaylight southbound plugin

Configuration

Just follow the steps:

Prepare the switch list database file

A sample is here, and we suggest to save it as */etc/snmp4sdn_swdb.csv* so that SNMP4SDN Plugin can automatically load this file. Note that the first line is title and should not be removed.

Prepare the vendor-specific configuration file

A sample is here, and we suggest to save it as */etc/snmp4sdn_VendorSpecificSwitchConfig.xml* so that SNMP4SDN Plugin can automatically load this file.

Install SNMP4SDN Plugin

If using SNMP4SDN Plugin provided in OpenDaylight release, just do the following from the Karaf CLI:

```
feature:install odl-snmp4sdn-all
```

Troubleshooting

Installation Troubleshooting

Feature installation failure

When trying to install a feature, if the following failure occurs:

```
Error executing command: Could not start bundle ...
Reason: Missing Constraint: Require-Capability: osgi.ee; filter="(&(osgi.
↪ee=JavaSE)(version=1.7))"
```

A workaround: exit Karaf, and edit the file <karaf_directory>/etc/config.properties, remove the line *${services-${karaf.framework}}* and the ", \" in the line above.

Runtime Troubleshooting

Problem starting SNMP Trap Interface

It is possible to get the following exception during controller startup. (The error would not be printed in Karaf console, one may see it in <karaf_directory>/data/log/karaf.log)

```
2014-01-31 15:00:44.688 CET [fileinstall-./plugins] WARN  o.o.snmp4sdn.internal.
↪SNMPListener - Problem starting SNMP Trap Interface: {}
 java.net.BindException: Permission denied
        at java.net.PlainDatagramSocketImpl.bind0(Native Method) ~[na:1.7.0_51]
        at java.net.AbstractPlainDatagramSocketImpl.
↪bind(AbstractPlainDatagramSocketImpl.java:95) ~[na:1.7.0_51]
        at java.net.DatagramSocket.bind(DatagramSocket.java:376) ~[na:1.7.0_51]
        at java.net.DatagramSocket.<init>(DatagramSocket.java:231) ~[na:1.7.0_51]
        at java.net.DatagramSocket.<init>(DatagramSocket.java:284) ~[na:1.7.0_51]
        at java.net.DatagramSocket.<init>(DatagramSocket.java:256) ~[na:1.7.0_51]
        at org.snmpj.SNMPTrapReceiverInterface.<init>(SNMPTrapReceiverInterface.
↪java:126) ~[org.snmpj-1.4.3.jar:na]
```

```
        at org.snmpj.SNMPTrapReceiverInterface.<init>(SNMPTrapReceiverInterface.
↪java:99) ~[org.snmpj-1.4.3.jar:na]
        at org.opendaylight.snmp4sdn.internal.SNMPListener.<init>(SNMPListener.
↪java:75) ~[bundlefile:na]
        at org.opendaylight.snmp4sdn.core.internal.Controller.start(Controller.
↪java:174) [bundlefile:na]
...
```

This indicates that the controller is being run as a user which does not have sufficient OS privileges to bind the SNMPTRAP port (162/UDP)

Switch list file missing

The SNMP4SDN Plugin needs a switch list file, which is necessary for topology discovery and should be provided by the administrator (so please prepare one for the first time using SNMP4SDN Plugin, here is the sample). The default file path is /etc/snmp4sdn_swdb.csv. SNMP4SDN Plugin would automatically load this file and start topology discovery. If this file is not ready there, the following message like this will pop up:

```
2016-02-02 04:21:52,476 | INFO| Event Dispatcher | CmethUtil                          |␣
↪466 - org.opendaylight.snmp4sdn - 0.3.0.SNAPSHOT | CmethUtil.readDB() err: {}
java.io.FileNotFoundException: /etc/snmp4sdn_swdb.csv (No such file or directory)
    at java.io.FileInputStream.open0(Native Method)[:1.8.0_65]
    at java.io.FileInputStream.open(FileInputStream.java:195)[:1.8.0_65]
    at java.io.FileInputStream.<init>(FileInputStream.java:138)[:1.8.0_65]
    at java.io.FileInputStream.<init>(FileInputStream.java:93)[:1.8.0_65]
    at java.io.FileReader.<init>(FileReader.java:58)[:1.8.0_65]
    at org.opendaylight.snmp4sdn.internal.util.CmethUtil.readDB(CmethUtil.java:66)
    at org.opendaylight.snmp4sdn.internal.util.CmethUtil.<init>(CmethUtil.java:43)
...
```

Configuration

Just follow the steps:

1. Prepare the switch list database file

A sample is here, and we suggest to save it as */etc/snmp4sdn_swdb.csv* so that SNMP4SDN Plugin can automatically load this file.

Note: The first line is title and should not be removed.

---

2. Prepare the vendor-specific configuration file

A sample is here, and we suggest to save it as */etc/snmp4sdn_VendorSpecificSwitchConfig.xml* so that SNMP4SDN Plugin can automatically load this file.

3. Install SNMP4SDN Plugin

If using SNMP4SDN Plugin provided in OpenDaylight release, just do the following:

Launch Karaf in Linux console:

```
cd <Boron_controller_directory>/bin
(for example, cd distribution-karaf-x.x.x-Boron/bin)
```

```
./karaf
```

Then in Karaf console, execute:

```
feature:install odl-snmp4sdn-all
```

4. Load switch list

For initialization, we need to feed SNMP4SDN Plugin the switch list. Actually SNMP4SDN Plugin automatically try to load the switch list at /etc/snmp4sdn_swdb.csv if there is. If so, this step could be skipped. In Karaf console, execute:

```
snmp4sdn:ReadDB <switch_list_path>
(For example, snmp4sdn:ReadDB /etc/snmp4sdn_swdb.csv)
(in Windows OS, For example, snmp4sdn:ReadDB D://snmp4sdn_swdb.csv)
```

A sample is here, and we suggest to save it as */etc/snmp4sdn_swdb.csv* so that SNMP4SDN Plugin can automatically load this file.

Note: The first line is title and should not be removed.

---

5. Show switch list

```
snmp4sdn:PrintDB
```

Tutorial

Topology Service

Execute topology discovery

The SNMP4SDN Plugin automatically executes topology discovery on startup. One may use the following commands to invoke topology discovery manually. Note that you may need to wait for seconds for itto complete.

Note: Currently, one needs to manually execute *snmp4sdn:TopoDiscover* first (just once), then later the automatic topology discovery can be successful. If switches change (switch added or removed), *snmp4sdn:TopoDiscover* is also required. A future version will fix it to eliminate these requirements.

```
snmp4sdn:TopoDiscover
```

If one like to discover all inventory (i.e. switches and their ports) but not edges, just execute "TopoDiscoverSwitches":

```
snmp4sdn:TopoDiscoverSwitches
```

If one like to only discover all edges but not inventory, just execute "TopoDiscoverEdges":

```
snmp4sdn:TopoDiscoverEdges
```

You can also trigger topology discovery via the REST API by using `curl` from the Linux console (or any other REST client):

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/
↪topology:rediscover
```

You can change the periodic topology discovery interval via a REST API:

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/topology:set-
↪discovery-interval -d "{"input":{"interval-second":'<interval_time>'}}"
For example, set the interval as 15 seconds:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/topology:set-
↪discovery-interval -d "{"input":{"interval-second":'15'}}"
```

Show the topology

SNMP4SDN Plugin supports to show topology via REST API:

- Get topology

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/
↪topology:get-edge-list
```

- Get switch list

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/
↪topology:get-node-list
```

- Get switches' ports list

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/
↪topology:get-node-connector-list
```

- The three commands above are just for user to get the latest topology discovery result, it does not trigger SNMP4SDN Plugin to do topology discovery.
- To trigger SNMP4SDN Plugin to do topology discover, as described in aforementioned *Execute topology discovery*.

Flow configuration

FDB configuration

SNMP4SDN supports to add entry on FDB table via REST API:

- Get FDB table

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/fdb:get-fdb-table -d "{input:{"node-id":<switch-mac-address-in-
↪number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/fdb:get-fdb-
↪table -d "{input:{"node-id":158969157063648}}"
```

- Get FDB table entry

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/fdb:get-fdb-entry -d "{input:{"node-id":<switch-mac-address-in-
↪number>, "vlan-id":<vlan-id-in-number>, "dest-mac-addr":<destination-mac-
↪address-in-number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/fdb:get-fdb-
↪entry -d "{input:{"node-id":158969157063648, "vlan-id":1, "dest-mac-addr
↪":158969157063648}}"
```

- Set FDB table entry

  (Notice invalid value: (1) non unicast mac (2) port not in the VLAN)

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/fdb:set-fdb-entry -d "{input:{"node-id":<switch-mac-address-in-
↪number>, "vlan-id":<vlan-id-in-number>, "dest-mac-addr":<destination-mac-
↪address-in-number>, "port":<port-in-number>, "type":'<type>'}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/fdb:set-fdb-
↪entry -d "{input:{"node-id":158969157063648, "vlan-id":1, "dest-mac-addr
↪":187649984473770, "port":23, "type":'MGMT'}}"
```

- Delete FDB table entry

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/fdb:del-fdb-entry -d "{input:{"node-id":<switch-mac-address-in-
↪number>, "vlan-id":<vlan-id-in-number>, "dest-mac-addr":<destination-mac-
↪address-in-number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/fdb:del-fdb-
↪entry -d "{input:{"node-id":158969157063648, "vlan-id":1, "dest-mac-addr
↪":187649984473770}}"
```

VLAN configuration

SNMP4SDN supports to add entry on VLAN table via REST API:

- Get VLAN table

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/vlan:get-vlan-table -d "{input:{node-id:<switch-mac-address-in-
↪number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/vlan:get-
↪vlan-table -d "{input:{node-id:158969157063648}}"
```

- Add VLAN

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/vlan:add-vlan -d "{"input":{"node-id":<switch-mac-address-in-number>,
↪ "vlan-id":<vlan-id-in-number>, "vlan-name":'<vlan-name>'}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/vlan:add-
↪vlan -d "{"input":{"node-id":158969157063648, "vlan-id":123, "vlan-name":'v123'}
↪}"
```

- Delete VLAN

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/vlan:delete-vlan -d "{"input":{"node-id":<switch-mac-address-in-
↪number>, "vlan-id":<vlan-id-in-number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/vlan:delete-
↪vlan -d "{"input":{"node-id":158969157063648, "vlan-id":123}}"
```

- Add VLAN and set ports

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/vlan:add-vlan-and-set-ports -d "{"input":{"node-id":<switch-mac-
↪address-in-number>, "vlan-id":<vlan-id-in-number>, "vlan-name":'<vlan-name>',
↪"tagged-port-list":'<tagged-ports-separated-by-comma>', "untagged-port-list":'
↪<untagged-ports-separated-by-comma>'}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/vlan:add-
↪vlan-and-set-ports -d "{"input":{"node-id":158969157063648, "vlan-id":123,
↪"vlan-name":'v123', "tagged-port-list":'1,2,3', "untagged-port-list":'4,5,6'}}"
```

- Set VLAN ports

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/vlan:set-vlan-ports -d "{"input":{"node-id":<switch-mac-address-in-
↪number>, "vlan-id":<vlan-id-in-number>, "tagged-port-list":'<tagged-ports-
↪separated-by-comma>', "untagged-port-list":'<untagged-ports-separated-by-comma>
↪'}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/vlan:set-
↪vlan-ports -d "{"input":{"node-id":"158969157063648", "vlan-id":"123", "tagged-
↪port-list":'4,5', "untagged-port-list":'2,3'}}"
```

ACL configuration

SNMP4SDN supports to add flow on ACL table via REST API. However, it is so far only implemented for the D-Link DGS-3120 switch.

ACL configuration via CLI is vendor-specific, and SNMP4SDN will support configuration with vendor-specific CLI in future release.

To do ACL configuration using the REST APIs, use commands like the following:

- Clear ACL table

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/acl:clear-acl-table -d "{"input":{"nodeId":<switch-mac-address-in-
↪number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/acl:clear-
↪acl-table -d "{"input":{"nodeId":158969157063648}}"
```

- Create ACL profile (IP layer)

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/acl:create-acl-profile -d "{input:{"nodeId":<switch-mac-address-in-
↪number>,"profile-id":<profile_id_in_number>,"profile-name":'<profile_name>',
↪"acl-layer":'IP',"vlan-mask":<vlan_mask_in_number>,"src-ip-mask":'<src_ip_mask>
↪','dst-ip-mask":"<destination_ip_mask>"}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/acl:create-
↪acl-profile -d "{input:{"nodeId":158969157063648,"profile-id":1,"profile-name":
↪'profile_1','acl-layer":'IP','vlan-mask":1,"src-ip-mask":'255.255.0.0','dst-ip-
↪mask":'255.255.255.255'}}"
```

- Create ACL profile (MAC layer)

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/acl:create-acl-profile -d "{input:{"nodeId":<switch-mac-address-in-
↪number>,"profile-id":<profile_id_in_number>,"profile-name":'<profile_name>',
↪"acl-layer":'ETHERNET',"vlan-mask":<vlan_mask_in_number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/acl:create-
↪acl-profile -d "{input:{"nodeId":158969157063648,"profile-id":2,"profile-name":
↪'profile_2',"acl-layer":'ETHERNET',"vlan-mask":4095}}"
```

- Delete ACL profile

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/acl:del-acl-
↪profile -d "{input:{"nodeId":<switch-mac-address-in-number>,"profile-id":
↪<profile_id_in_number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/acl:del-acl-
↪profile -d "{input:{"nodeId":158969157063648,"profile-id":1}}"
```

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/acl:del-acl-profile -d "{input:{"nodeId":<switch-mac-address-in-
↪number>,"profile-name":"<profile_name>"}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/acl:del-acl-
↪profile -d "{input:{"nodeId":158969157063648,"profile-name":'profile_2'}}"
```

- Set ACL rule

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/acl:set-acl-rule -d "{input:{"nodeId":<switch-mac-address-in-number>,
↪"profile-id":<profile_id_in_number>,"profile-name":'<profile_name>',"rule-id":
↪<rule_id_in_number>,"port-list":[<port_number>,<port_number>,...],"acl-layer":'
↪<acl_layer>','vlan-id":<vlan_id_in_number>,"src-ip":"<src_ip_address>","dst-ip":
↪'<dst_ip_address>',"acl-action":'<acl_action>'}}"
(<acl_layer>: IP or ETHERNET)
(<acl_action>: PERMIT as permit, DENY as deny)

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/acl:set-acl-
↪rule -d "{input:{"nodeId":158969157063648,"profile-id":1,"profile-name":
↪'profile_1',"rule-id":1,"port-list":[1,2,3],"acl-layer":'IP',"vlan-id":2,"src-ip
↪":'1.1.1.1',"dst-ip":'2.2.2.2',"acl-action":'PERMIT'}}"
```

- Delete ACL rule

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/acl:del-acl-rule -d "{input:{"nodeId":<switch-mac-address-in-number>,
↪"profile-id":<profile_id_in_number>,"profile-name":'<profile_name>',"rule-id":
↪<rule_id_in_number>}}"
```

```
For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/acl:del-acl-
↪rule -d "{input:{"nodeId":158969157063648,"profile-id":1,"profile-name":
↪'profile_1',"rule-id":1}}"
```

Special configuration

SNMP4SDN supports setting the following special configurations via REST API:

- Set STP port state

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:set-stp-port-state -d "{input:{"node-id":<switch-mac-address-
↪in-number>, "port":<port_number>, enable:<true_or_false>}}"
(true: enable, false: disable)

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/config:set-
↪stp-port-state -d "{input:{"node-id":158969157063648, "port":2, enable:false}}"
```

- Get STP port state

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:get-stp-port-state -d "{input:{"node-id":<switch-mac-address-
↪in-number>, "port":<port_number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/config:get-
↪stp-port-state -d "{input:{"node-id":158969157063648, "port":2}}"
```

- Get STP port root

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:get-stp-port-root -d "{input:{"node-id":<switch-mac-address-
↪in-number>, "port":<port_number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/config:get-
↪stp-port-root -d "{input:{"node-id":158969157063648, "port":2}}"
```

- Enable STP

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:enable-stp -d "{input:{"node-id":<switch-mac-address-in-
↪number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/
↪config:enable-stp -d "{input:{"node-id":158969157063648}}"
```

- Disable STP

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:disable-stp -d "{input:{"node-id":<switch-mac-address-in-
↪number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/
↪config:disable-stp -d "{input:{"node-id":158969157063648}}"
```

- Get ARP table

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:get-arp-table -d "{input:{"node-id":<switch-mac-address-in-
↪number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/config:get-
↪arp-table -d "{input:{"node-id":158969157063648}}"
```

- Set ARP entry

  (Notice to give IP address with subnet prefix)

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:set-arp-entry -d "{input:{"node-id":<switch-mac-address-in-
↪number>, "ip-address":'<ip_address>', "mac-address":<mac_address_in_number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/config:set-
↪arp-entry -d "{input:{"node-id":158969157063648, "ip-address":'10.217.9.9',
↪"mac-address":1}}"
```

- Get ARP entry

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:get-arp-entry -d "{input:{"node-id":<switch-mac-address-in-
↪number>, "ip-address":'<ip_address>'}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/config:get-
↪arp-entry -d "{input:{"node-id":158969157063648, "ip-address":'10.217.9.9'}}"
```

- Delete ARP entry

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:delete-arp-entry -d "{input:{"node-id":<switch-mac-address-in-
↪number>, "ip-address":'<ip_address>'}}"
```

```
For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/
↪config:delete-arp-entry -d "{input:{"node-id":158969157063648, "ip-address":'10.
↪217.9.9'}}"
```

Using Postman to invoke REST API

Besides using the curl tool to invoke REST API, like the examples aforementioned, one can also use GUI tool like Postman for better data display.

- Install Postman: Install Postman in the Chrome browser
- In the chrome browser bar enter

```
chrome://apps/
```

- Click on Postman.

Example: Get VLAN table using Postman

As the screenshot shown below, one needs to fill in required fields.

```
URL:
http://<controller_ip_address>:8181/restconf/operations/vlan:get-vlan-table

Accept header:
application/json

Content-type:
application/json

Body:
{input:{"node-id":<node_id>}}
for example:
{input:{"node-id":158969157063648}}
```

Multi-vendor support

So far the supported vendor-specific configurations:

- Add VLAN and set ports
- (More functions are TBD)

The SNMP4SDN Plugin would examine whether the configuration is described in the vendor-specific configuration file. If yes, the configuration description would be adopted, otherwise just use the default configuration. For example, adding VLAN and setting the ports is supported via SNMP standard MIB. However we found some special cases, for example, certain Accton switch requires to add VLAN first and then allows to set the ports. So one may describe this in the vendor-specific configuration file.

A vendor-specific configuration file sample is here, and we suggest to save it as */etc/snmp4sdn_VendorSpecificSwitchConfig.xml* so that SNMP4SDN Plugin can automatically load it.

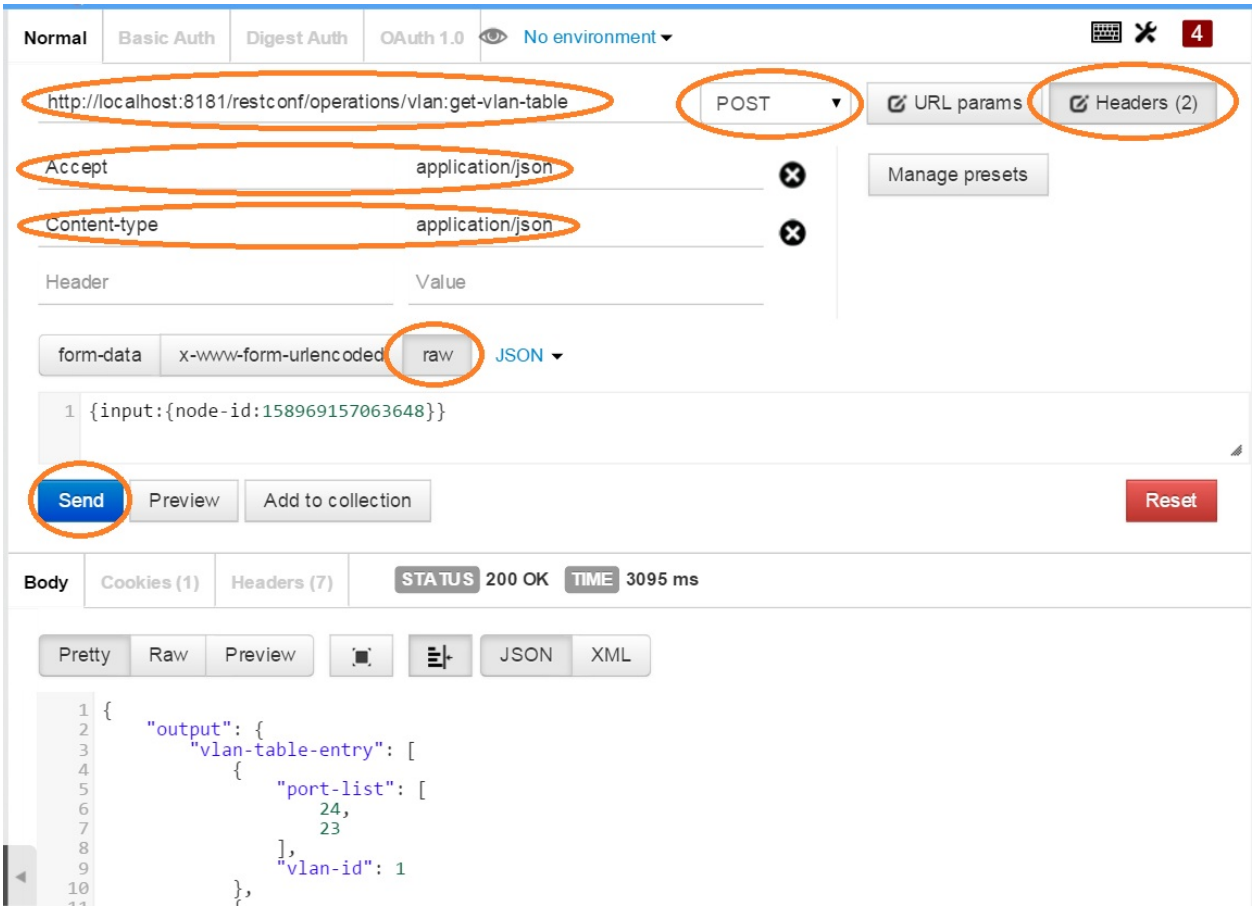


Fig. 1.111: Example: Get VLAN table using Postman

Help

- SNMP4SDN Wiki
- SNMP4SDN Mailing Lists: (user, developer)
- Latest troubleshooting in Wiki

SXP User Guide

Overview

SXP (Scalable-Group Tag eXchange Protocol) project is an effort to enhance OpenDaylight platform with IP-SGT (IP Address to Source Group Tag) bindings that can be learned from connected SXP-aware network nodes. The current implementation supports SXP protocol version 4 according to the Smith, Kandula - SXP IETF draft and grouping of peers and creating filters based on ACL/Prefix-list syntax for filtering outbound and inbound IP-SGT bindings. All protocol legacy versions 1-3 are supported as well. Additionally, version 4 adds bidirectional connection type as an extension of a unidirectional one.

SXP Architecture

The SXP Server manages all connected clients in separate threads and a common SXP protocol agreement is used between connected peers. Each SXP network peer is modelled with its pertaining class, e.g., SXP Server represents the SXP Speaker, SXP Listener the Client. The server program creates the ServerSocket object on a specified port and waits until a client starts up and requests connect on the IP address and port of the server. The client program opens a Socket that is connected to the server running on the specified host IP address and port.

The SXP Listener maintains connection with its speaker peer. From an opened channel pipeline, all incoming SXP messages are processed by various handlers. Message must be decoded, parsed and validated.

The SXP Speaker is a counterpart to the SXP Listener. It maintains a connection with its listener peer and sends composed messages.

The SXP Binding Handler extracts the IP-SGT binding from a message and pulls it into the SXP-Database. If an error is detected during the IP-SGT extraction, an appropriate error code and sub-code is selected and an error message is sent back to the connected peer. All transitive messages are routed directly to the output queue of SXP Binding Dispatcher.

The SXP Binding Dispatcher represents a selector that will decides how many data from the SXP-database will be sent and when. It is responsible for message content composition based on maximum message length.

The SXP Binding Filters handles filtering of outgoing and incoming IP-SGT bindings according to BGP filtering using ACL and Prefix List syntax for specifying filter or based on Peer-sequence length.

The SXP Domains feature provides isolation of SXP peers and bindings learned between them, also exchange of Bindings is possible across SXP-Domains by ACL, Prefix List or Peer-Sequence filters

Configuring SXP

The OpenDaylight Karaf distribution comes pre-configured with baseline SXP configuration. Configuration of SXP Nodes is also possible via NETCONF.

- **22-sxp-controller-one-node.xml** (defines the basic parameters)

Administering or Managing SXP

By RPC (response is XML document containing requested data or operation status):

- Get Connections POST http://127.0.0.1:8181/restconf/operations/sxp-controller:get-connections

```
<input xmlns:xsi="urn:opendaylight:sxp:controller">
 <domain-name>global</domain-name>
 <requested-node>0.0.0.100</requested-node>
</input>
```

- Add Connection POST http://127.0.0.1:8181/restconf/operations/sxp-controller:add-connection

```
<input xmlns:xsi="urn:opendaylight:sxp:controller">
 <requested-node>0.0.0.100</requested-node>
 <domain-name>global</domain-name>
 <connections>
  <connection>
   <peer-address>172.20.161.50</peer-address>
   <tcp-port>64999</tcp-port>
   <!-- Password setup: default | none leave empty -->
   <password>default</password>
   <!-- Mode: speaker/listener/both -->
   <mode>speaker</mode>
   <version>version4</version>
   <description>Connection to ASR1K</description>
   <!-- Timers setup: 0 to disable specific timer usability, the default value will␣
↪be used -->
   <connection-timers>
    <!-- Speaker -->
    <hold-time-min-acceptable>45</hold-time-min-acceptable>
    <keep-alive-time>30</keep-alive-time>
   </connection-timers>
  </connection>
  <connection>
   <peer-address>172.20.161.178</peer-address>
   <tcp-port>64999</tcp-port>
   <!-- Password setup: default | none leave empty-->
   <password>default</password>
   <!-- Mode: speaker/listener/both -->
   <mode>listener</mode>
   <version>version4</version>
   <description>Connection to ISR</description>
   <!-- Timers setup: 0 to disable specific timer usability, the default value will␣
↪be used -->
   <connection-timers>
    <!-- Listener -->
    <reconciliation-time>120</reconciliation-time>
    <hold-time>90</hold-time>
    <hold-time-min>90</hold-time-min>
    <hold-time-max>180</hold-time-max>
   </connection-timers>
  </connection>
 </connections>
</input>
```

- Delete Connection POST http://127.0.0.1:8181/restconf/operations/sxp-controller:delete-connection

```
<input xmlns:xsi="urn:opendaylight:sxp:controller">
 <requested-node>0.0.0.100</requested-node>
 <domain-name>global</domain-name>
 <peer-address>172.20.161.50</peer-address>
</input>
```

- Add Binding Entry POST http://127.0.0.1:8181/restconf/operations/sxp-controller:add-entry

```
<input xmlns:xsi="urn:opendaylight:sxp:controller">
 <requested-node>0.0.0.100</requested-node>
 <domain-name>global</domain-name>
 <ip-prefix>192.168.2.1/32</ip-prefix>
 <sgt>20</sgt >
</input>
```

- Update Binding Entry POST http://127.0.0.1:8181/restconf/operations/sxp-controller:update-entry

```
<input xmlns:xsi="urn:opendaylight:sxp:controller">
 <requested-node>0.0.0.100</requested-node>
 <domain-name>global</domain-name>
 <original-binding>
  <ip-prefix>192.168.2.1/32</ip-prefix>
  <sgt>20</sgt>
 </original-binding>
 <new-binding>
  <ip-prefix>192.168.3.1/32</ip-prefix>
  <sgt>30</sgt>
 </new-binding>
</input>
```

- Delete Binding Entry POST http://127.0.0.1:8181/restconf/operations/sxp-controller:delete-entry

```
<input xmlns:xsi="urn:opendaylight:sxp:controller">
 <requested-node>0.0.0.100</requested-node>
 <domain-name>global</domain-name>
 <ip-prefix>192.168.3.1/32</ip-prefix>
 <sgt>30</sgt >
</input>
```

- Get Node Bindings

  This RPC gets particular device bindings. An SXP-aware node is identified with a unique Node-ID. If a user requests bindings for a Speaker 20.0.0.2, the RPC will search for an appropriate path, which contains 20.0.0.2 Node-ID, within locally learnt SXP data in the SXP database and replies with associated bindings. POST http://127.0.0.1:8181/restconf/operations/sxp-controller:get-node-bindings

```
<input xmlns:xsi="urn:opendaylight:sxp:controller">
 <requested-node>20.0.0.2</requested-node>
 <bindings-range>all</bindings-range>
 <domain-name>global</domain-name>
</input>
```

- Get Binding SGTs POST http://127.0.0.1:8181/restconf/operations/sxp-controller:get-binding-sgts

```
<input xmlns:xsi="urn:opendaylight:sxp:controller">
 <requested-node>0.0.0.100</requested-node>
 <domain-name>global</domain-name>
```

```
 <ip-prefix>192.168.12.2/32</ip-prefix>
</input>
```

- Add PeerGroup with or without filters to node. POST http://127.0.0.1:8181/restconf/operations/sxp-controller:add-peer-group

```
<input xmlns="urn:opendaylight:sxp:controller">
 <requested-node>127.0.0.1</requested-node>
 <sxp-peer-group>
  <name>TEST</name>
  <sxp-peers>
  </sxp-peers>
  <sxp-filter>
   <filter-type>outbound</filter-type>
   <acl-entry>
    <entry-type>deny</entry-type>
    <entry-seq>1</entry-seq>
    <sgt-start>1</sgt-start>
    <sgt-end>100</sgt-end>
   </acl-entry>
   <acl-entry>
    <entry-type>permit</entry-type>
    <entry-seq>45</entry-seq>
    <matches>1</matches>
    <matches>3</matches>
    <matches>5</matches>
   </acl-entry>
  </sxp-filter>
 </sxp-peer-group>
</input>
```

- Delete PeerGroup with peer-group-name from node request-node. POST http://127.0.0.1:8181/restconf/operations/sxp-controller:delete-peer-group

```
<input xmlns="urn:opendaylight:sxp:controller">
 <requested-node>127.0.0.1</requested-node>
 <peer-group-name>TEST</peer-group-name>
</input>
```

- Get PeerGroup with peer-group-name from node request-node. POST http://127.0.0.1:8181/restconf/operations/sxp-controller:get-peer-group

```
<input xmlns="urn:opendaylight:sxp:controller">
 <requested-node>127.0.0.1</requested-node>
 <peer-group-name>TEST</peer-group-name>
</input>
```

- Add Filter to peer group on node request-node. POST http://127.0.0.1:8181/restconf/operations/sxp-controller:add-filter

```
<input xmlns="urn:opendaylight:sxp:controller">
 <requested-node>127.0.0.1</requested-node>
 <peer-group-name>TEST</peer-group-name>
 <sxp-filter>
  <filter-type>outbound</filter-type>
  <acl-entry>
   <entry-type>deny</entry-type>
   <entry-seq>1</entry-seq>
```

```
    <sgt-start>1</sgt-start>
    <sgt-end>100</sgt-end>
   </acl-entry>
   <acl-entry>
    <entry-type>permit</entry-type>
    <entry-seq>45</entry-seq>
    <matches>1</matches>
    <matches>3</matches>
    <matches>5</matches>
   </acl-entry>
  </sxp-filter>
</input>
```

- Delete Filter from peer group on node request-node. POST http://127.0.0.1:8181/restconf/operations/sxp-controller:delete-filter

```
<input xmlns="urn:opendaylight:sxp:controller">
 <requested-node>127.0.0.1</requested-node>
 <peer-group-name>TEST</peer-group-name>
 <filter-type>outbound</filter-type>
</input>
```

- Update Filter of the same type in peer group on node request-node. POST http://127.0.0.1:8181/restconf/operations/sxp-controller:update-filter

```
<input xmlns="urn:opendaylight:sxp:controller">
 <requested-node>127.0.0.1</requested-node>
 <peer-group-name>TEST</peer-group-name>
 <sxp-filter>
  <filter-type>outbound</filter-type>
  <acl-entry>
   <entry-type>deny</entry-type>
   <entry-seq>1</entry-seq>
   <sgt-start>1</sgt-start>
   <sgt-end>100</sgt-end>
  </acl-entry>
  <acl-entry>
   <entry-type>permit</entry-type>
   <entry-seq>45</entry-seq>
   <matches>1</matches>
   <matches>3</matches>
   <matches>5</matches>
  </acl-entry>
 </sxp-filter>
</input>
```

- Add new SXP aware Node POST http://127.0.0.1:8181/restconf/operations/sxp-controller:add-node

```
<input xmlns="urn:opendaylight:sxp:controller">
    <node-id>1.1.1.1</node-id>
    <source-ip>0.0.0.0</source-ip>
    <timers>
        <retry-open-time>5</retry-open-time>
        <hold-time-min-acceptable>120</hold-time-min-acceptable>
        <delete-hold-down-time>120</delete-hold-down-time>
        <hold-time-min>90</hold-time-min>
        <reconciliation-time>120</reconciliation-time>
        <hold-time>90</hold-time>
```

```
        <hold-time-max>180</hold-time-max>
        <keep-alive-time>30</keep-alive-time>
    </timers>
    <mapping-expanded>150</mapping-expanded>
    <security>
        <password>password</password>
    </security>
    <tcp-port>64999</tcp-port>
    <version>version4</version>
    <description>ODL SXP Controller</description>
    <master-database></master-database>
</input>
```

- Delete SXP aware node POST http://127.0.0.1:8181/restconf/operations/sxp-controller:delete-node

```
<input xmlns="urn:opendaylight:sxp:controller">
 <node-id>1.1.1.1</node-id>
</input>
```

- Add SXP Domain on node request-node. POST http://127.0.0.1:8181/restconf/operations/sxp-controller:add-domain

```
<input xmlns="urn:opendaylight:sxp:controller">
  <node-id>1.1.1.1</node-id>
  <domain-name>global</domain-name>
</input>
```

- Delete SXP Domain on node request-node. POST http://127.0.0.1:8181/restconf/operations/sxp-controller:delete-domain

```
<input xmlns="urn:opendaylight:sxp:controller">
 <node-id>1.1.1.1</node-id>
 <domain-name>global</domain-name>
</input>
```

- Add Route Adds route to leader Node. PUT http://127.0.0.1:8181/restconf/config/sxp-cluster-route:sxp-cluster-route/

```
<sxp-cluster-route xmlns="urn:opendaylight:sxp:cluster:route">
    <routing-definition>
        <ip-address>80.12.43.2</ip-address>
        <interface>eth1:0</interface>
        <netmask>255.255.255.0</netmask>
    </routing-definition>
</sxp-cluster-route>
```

Use cases for SXP

Cisco has a wide installed base of network devices supporting SXP. By including SXP in OpenDaylight, the binding of policy groups to IP addresses can be made available for possible further processing to a wide range of devices, and applications running on OpenDaylight. The range of applications that would be enabled is extensive. Here are just a few of them:

OpenDaylight based applications can take advantage of the IP-SGT binding information. For example, access control can be defined by an operator in terms of policy groups, while OpenDaylight can configure access control lists on network elements using IP addresses, e.g., existing technology.

Interoperability between different vendors. Vendors have different policy systems. Knowing the IP-SGT binding for Cisco makes it possible to maintain policy groups between Cisco and other vendors.

OpenDaylight can aggregate the binding information from many devices and communicate it to a network element. For example, a firewall can use the IP-SGT binding information to know how to handle IPs based on the group-based ACLs it has set. But to do this with SXP alone, the firewall has to maintain a large number of network connections to get the binding information. This incurs heavy overhead costs to maintain all of the SXP peering and protocol information. OpenDaylight can aggregate the IP-group information so that the firewall need only connect to OpenDaylight. By moving the information flow outside of the network elements to a centralized position, we reduce the overhead of the CPU consumption on the enforcement element. This is a huge savings - it allows the enforcement point to only have to make one connection rather than thousands, so it can concentrate on its primary job of forwarding and enforcing.

OpenDaylight can relay the binding information from one network element to others. Changes in group membership can be propagated more readily through a centralized model. For example, in a security application a particular host (e.g., user or IP Address) may be found to be acting suspiciously or violating established security policies. The defined response is to put the host into a different source group for remediation actions such as a lower quality of service, restricted access to critical servers, or special routing conditions to ensure deeper security enforcement (e.g., redirecting the host's traffic through an IPS with very restrictive policies). Updated group membership for this host needs to be communicated to multiple network elements as soon as possible; a very efficient and effective method of propagation can be performed using OpenDaylight as a centralized point for relaying the information.

OpenDaylight can create filters for exporting and receiving IP-SGT bindings used on specific peer groups, thus can provide more complex maintaining of policy groups.

Although the IP-SGT binding is only one specific piece of information, and although SXP is implemented widely in a single vendor's equipment, bringing the ability of OpenDaylight to process and distribute the bindings, is a very specific immediate useful implementation of policy groups. It would go a long way to develop both the usefulness of OpenDaylight and of policy groups.

TSDR User Guide

This document describes how to use HSQLDB, HBase, and Cassandra data stores to capture time series data using Time Series Data Repository (TSDR) features in OpenDaylight. This document contains configuration, administration, management, usage, and troubleshooting sections for these features.

Overview

The Time Series Data Repository (TSDR) project in OpenDaylight (ODL) creates a framework for collecting, storing, querying, and maintaining time series data. TSDR provides the framework for plugging in data collectors to collect various time series data and store the data into TSDR Data Stores. With a common data model and generic TSDR data persistence APIs, the user can choose various data stores to be plugged into the TSDR persistence framework. Currently, three types of data stores are supported: HSQLDB relational database (default installed), HBase NoSQL database and Cassandra NoSQL database.

With the capabilities of data collection, storage, query, aggregation, and purging provided by TSDR, network administrators can leverage various data driven applications built on top of TSDR for security risk detection, performance analysis, operational configuration optimization, traffic engineering and network analytics with automated intelligence.

TSDR Architecture

TSDR has the following major components:

- Data Collection Service
- Data Storage Service

- TSDR Persistence Layer with data stores as plugins
- TSDR Data Stores
- Data Query Service
- Grafana integration for time series data visualization
- Data Aggregation Service
- Data Purging Service

The Data Collection Service handles the collection of time series data into TSDR and hands it over to the Data Storage Service. The Data Storage Service stores the data into TSDR through the TSDR Persistence Layer. The TSDR Persistence Layer provides generic Service APIs allowing various data stores to be plugged in. The Data Aggregation Service aggregates time series fine-grained raw data into course-grained roll-up data to control the size of the data. The Data Purging Service periodically purges both fine-grained raw data and course-grained aggregated data according to user-defined schedules.

TSDR provides component-based services on a common data model. These services include the data collection service, data storage service and data query service. The TSDR data storage service supports HSQLDB (the default datastore), HBASE and Cassandra datastores. Between these services and components, time series data is communicated using a common TSDR data model. This data model is designed around the abstraction of time series data commonalities. With these services, TSDR is able to collect the data from the data sources and store them into one of the TSDR data stores; HSQLDB, HBase and Cassandra datastores. Data can be retrieved with the Data Query service using the default OpenDaylight RestConf interface or its ODL API interface. TSDR also has integrated support for ElasticSearch capabilities. TSDR data can also be viewed directly with Grafana for time series visualization or various chart formats.

Configuring TSDR Data Stores

To Configure HSQLDB Data Store

The HSQLDB based storage files get stored automatically in <karaf install folder>/tsdr/ directory. If you want to change the default storage location, the configuration file to change can be found in <karaf install folder>/etc directory. The filename is org.ops4j.datasource-metric.cfg. Change the last portion of the url=jdbc:hsqldb:./tsdr/metric to point to different directory.

To Configure HBase Data Store

After installing HBase Server on the same machine as OpenDaylight, if the user accepts the default configuration of the HBase Data Store, the user can directly proceed with the installation of HBase Data Store from Karaf console.

Optionally, the user can configure TSDR HBase Data Store following HBase Data Store Configuration Procedure.

- HBase Data Store Configuration Steps
  - Open the file etc/tsdr-persistence-hbase.peroperties under karaf distribution directory.
  - Edit the following parameters:
    - HBase server name
    - HBase server port
    - HBase client connection pool size
    - HBase client write buffer size

After the configuration of HBase Data Store is complete, proceed with the installation of HBase Data Store from Karaf console.

- HBase Data Store Installation Steps
  - Start Karaf Console
  - Run the following commands from Karaf Console: feature:install odl-tsdr-hbase

To Configure Cassandra Data Store

Currently, there's no configuration needed for Cassandra Data Store. The user can use Cassandra data store directly after installing the feature from Karaf console.

Additionally separate commands have been implemented to install various data collectors.

Administering or Managing TSDR Data Stores

To Administer HSQLDB Data Store

Once the TSDR default datastore feature (odl-tsdr-hsqldb-all) is enabled, the TSDR captured OpenFlow statistics metrics can be accessed from Karaf Console by executing the command

```
tsdr:list <metric-category> <starttimestamp> <endtimestamp>
```

wherein

- <metric-category> = any one of the following categories FlowGroupStats, FlowMeterStats, FlowStats, FlowTableStats, PortStats, QueueStats
- <starttimestamp> = to filter the list of metrics starting this timestamp
- <endtimestamp> = to filter the list of metrics ending this timestamp
- <starttimestamp> and <endtimestamp> are optional.
- Maximum 1000 records will be displayed.

To Administer HBase Data Store

- Using Karaf Command to retrieve data from HBase Data Store

The user first need to install hbase data store from karaf console:

feature:install odl-tsdr-hbase

The user can retrieve the data from HBase data store using the following commands from Karaf console:

```
tsdr:list
tsdr:list <CategoryName> <StartTime> <EndTime>
```

Typing tab will get the context prompt of the arguments when typeing the command in Karaf console.

To Administer Cassandra Data Store

The user first needs to install Cassandra data store from Karaf console:

```
feature:install odl-tsdr-cassandra
```

Then the user can retrieve the data from Cassandra data store using the following commands from Karaf console:

```
tsdr:list
tsdr:list <CategoryName> <StartTime> <EndTime>
```

Typing tab will get the context prompt of the arguments when typeing the command in Karaf console.

Installing TSDR Data Collectors

When the user uses HSQLDB data store and installed "odl-tsdr-hsqldb-all" feature from Karaf console, besides the HSQLDB data store, OpenFlow data collector is also installed with this command. However, if the user needs to use other collectors, such as NetFlow Collector, Syslog Collector, SNMP Collector, and Controller Metrics Collector, the user needs to install them with separate commands. If the user uses HBase or Cassandra data store, no collectors will be installed when the data store is installed. Instead, the user needs to install each collector separately using feature install command from Karaf console.

The following is the list of supported TSDR data collectors with the associated feature install commands:

- OpenFlow Data Collector

  ```
  feature:install odl-tsdr-openflow-statistics-collector
  ```

- NetFlow Data Collector

  ```
  feature:install odl-tsdr-netflow-statistics-collector
  ```

- sFlow Data Collector

  ```
  feature:install odl-tsdr-sflow-statistics-colletor
  ```

- SNMP Data Collector

  ```
  feature:install odl-tsdr-snmp-data-collector
  ```

- Syslog Data Collector

  ```
  feature:install odl-tsdr-syslog-collector
  ```

- Controller Metrics Collector

  ```
  feature:install odl-tsdr-controller-metrics-collector
  ```

- Web Activity Collector

  ```
  feature:install odl-tsdr-restconf-collector
  ```

In order to use controller metrics collector, the user needs to install Sigar library.

The following is the instructions for installing Sigar library on Ubuntu:

- Install back end library by "sudo apt-get install libhyperic-sigar-java"

- Execute "export LD_LIBRARY_PATH=/usr/lib/jni/:/usr/lib:/usr/local/lib" to set the path of the JNI (you can add this to the ".bashrc" in your home directory)
- Download the file "sigar-1.6.4.jar". It might be also in your ".m2" directory under "~/.m2/resources/org/fusesource/sigar/1.6.4"
- Create the directory "org/fusesource/sigar/1.6.4" under the "system" directory in your controller home directory and place the "sigar-1.6.4.jar" there

Configuring TSDR Data Collectors

- SNMP Data Collector Device Credential Configuration

After installing SNMP Data Collector, a configuration file under etc/ directory of ODL distribution is generated: etc/tsdr.snmp.cfg is created.

The following is a sample tsdr.snmp.cfg file:

credentials=[192.168.0.2,public],[192.168.0.3,public]

The above credentials indicate that TSDR SNMP Collector is going to connect to two devices. The IPAddress and Read community string of these two devices are (192.168.0.2, public), and (192.168.0.3) respectively.

The user can make changes to this configuration file any time during runtime. The configuration will be picked up by TSDR in the next cycle of data collection.

Polling interval configuration for SNMP Collector and OpenFlow Stats Collector

The default polling interval of SNMP Collector and OpenFlow Stats Collector is 30 seconds and 15 seconds respectively. The user can change the polling interval through restconf APIs at any time. The new polling interval will be picked up by TSDR in the next collection cycle.

- Retrieve Polling Interval API for SNMP Collector
  - URL: http://localhost:8181/restconf/config/tsdr-snmp-data-collector:TSDRSnmpDataCollectorConfig
  - Verb: GET
- Update Polling Interval API for SNMP Collector
  - URL: http://localhost:8181/restconf/operations/tsdr-snmp-data-collector:setPollingInterval
  - Verb: POST
  - Content Type: application/json
  - Input Payload:

```
{
   "input": {
       "interval": "15000"
   }
}
```

- Retrieve Polling Interval API for OpenFlowStats Collector
  - URL: http://localhost:8181/restconf/config/tsdr-openflow-statistics-collector:TSDROSCConfig
  - Verb: GET
- Update Polling Interval API for OpenFlowStats Collector

- URL: http://localhost:8181/restconf/operations/tsdr-openflow-statistics-collector:setPollingInterval
- Verb: POST
- Content Type: application/json
- Input Payload:

```
{
   "input": {
       "interval": "15000"
   }
}
```

Querying TSDR from REST APIs

TSDR provides two REST APIs for querying data stored in TSDR data stores.

- Query of TSDR Metrics
  - URL: http://localhost:8181/tsdr/metrics/query
  - Verb: GET
  - Parameters:
    - tsdrkey=[NID=][DC=][MN=][RK=]

```
The TSDRKey format indicates the NodeID(NID), DataCategory(DC),␣
↪MetricName(MN), and RecordKey(RK) of the monitored objects.
For example, the following is a valid tsdrkey:
[NID=openflow:1][DC=FLOWSTATS][MN=PacketCount][RK=Node:openflow:1,Table:0,
↪Flow:3]
The following is also a valid tsdrkey:
tsdrkey=[NID=][DC=FLOWSTATS][MN=][RK=]
In the case when the sections in the tsdrkey is empty, the query will␣
↪return all the records in the TSDR data store that matches the filled␣
↪tsdrkey. In the above example, the query will return all the data in␣
↪FLOWSTATS data category.
The query will return only the first 1000 records that match the query␣
↪criteria.
```

    - from=<time_in_seconds>
    - until=<time_in_seconds>

The following is an example curl command for querying metric data from TSDR data store:

curl -G -v -H "Accept: application/json" -H "Content-Type: application/json" "http://localhost:8181/tsdr/metrics/query" –data-urlencode "tsdrkey=[NID=][DC=FLOWSTATS][MN=][RK=]" –data-urlencode "from=0" –data-urlencode "until=240000000000"|more

- Query of TSDR Log type of data
  - URL:http://localhost:8181/tsdr/logs/query
  - Verb: GET
  - Parameters:
    - tsdrkey=tsdrkey=[NID=][DC=][RK=]

```
The TSDRKey format indicates the NodeID(NID), DataCategory(DC), and␣
↪RecordKey(RK) of the monitored objects.
For example, the following is a valid tsdrkey:
[NID=openflow:1][DC=NETFLOW][RK]
The query will return only the first 1000 records that match the query␣
↪criteria.
```

* from=<time_in_seconds>
* until=<time_in_seconds>

The following is an example curl command for querying log type of data from TSDR data store:

curl -G -v -H "Accept: application/json" -H "Content-Type: application/json" "http://localhost:8181/tsdr/logs/query" –data-urlencode "tsdrkey=[NID=][DC=NETFLOW][RK=]" –data-urlencode "from=0" –data-urlencode "until=240000000000"lmore

Grafana integration with TSDR

TSDR provides northbound integration with Grafana time series data visualization tool. All the metric type of data stored in TSDR data store can be visualized using Grafana.

For the detailed instruction about how to install and configure Grafana to work with TSDR, please refer to the following link:

https://wiki.opendaylight.org/view/Grafana_Integration_with_TSDR_Step-by-Step

Purging Service configuration

After the data stores are installed from Karaf console, the purging service will be installed as well. A configuration file called tsdr.data.purge.cfg will be generated under etc/ directory of ODL distribution.

The following is the sample default content of the tsdr.data.purge.cfg file:

host=127.0.0.1 data_purge_enabled=true data_purge_time=23:59:59 data_purge_interval_in_minutes=1440 retention_time_in_hours=168

The host indicates the IPAddress of the data store. In the case when the data store is together with ODL controller, 127.0.0.1 should be the right value for the host IP. The other attributes are self-explained. The user can change those attributes at any time. The configuration change will be picked up right away by TSDR Purging service at runtime.

How to use TSDR to collect, store, and view OpenFlow Interface Statistics

Overview

This tutorial describes an example of using TSDR to collect, store, and view one type of time series data in OpenDaylight environment.

Prerequisites

You would need to have the following as prerequisits:

- One or multiple OpenFlow enabled switches. Alternatively, you can use mininet to simulate such a switch.
- Successfully installed OpenDaylight Controller.

- Successfully installed HBase Data Store following TSDR HBase Data Store Installation Guide.
- Connect the OpenFlow enabled switch(es) to OpenDaylight Controller.

Target Environment

HBase data store is only supported in Linux operation system.

Instructions

- Start OpenDaylight.
- Connect OpenFlow enabled switch(es) to the controller.
  - If using mininet, run the following commands from mininet command line:
    - mn –topo single,3 –controller *remote,ip=172.17.252.210,port=6653* –switch ovsk,protocols=OpenFlow13
- Install TSDR hbase feature from Karaf:
  - feature:install odl-tsdr-hbase
- Install OpenFlow Statistics Collector from Karaf:
  - feature:install odl-tsdr-openflow-statistics-collector
- run the following command from Karaf console:
  - tsdr:list PORTSTATS

You should be able to see the interface statistics of the switch(es) from the HBase Data Store. If there are too many rows, you can use "tsdr:list InterfaceStats|more" to view it page by page.

By tabbing after "tsdr:list", you will see all the supported data categories. For example, "tsdr:list FlowStats" will output the Flow statistics data collected from the switch(es).

ElasticSearch

To setup and run the TSDR data store ElasticSearch feature, you need to have an ElasticSearch node (or a cluster of such nodes) running. You can use a customized ElasticSearch docker image for this purpose.

Your ElasticSearch (ES) setup must have the "Delete By Query Plugin" installed. Without this, some of the ES functionality won't work properly.

(You can skip this section if you already have an instance of ElasticSearch running)

Run the following set of commands:

```
cat << EOF > Dockerfile
FROM elasticsearch:2
RUN /usr/share/elasticsearch/bin/plugin install --batch delete-by-query
EOF
```

To build the image, run the following command in the directory where the Dockerfile was created:

```
docker build . -t elasticsearch-dd
```

You can check whether the image was properly created by running:

```
docker images
```

This should print all your container images including the elasticsearch-dd.

Now we can create and run a container from our image by typing:

```
docker run -d -p 9200:9200 -p 9300:9300 --name elasticsearch-dd elasticsearch-dd
```

To see whether the container is running, run the following command:

```
docker ps
```

The output should include a row with elasticsearch-dd in the NAMES column. To check the std out of this container use

```
docker logs elasticsearch-dd
```

Running the ElasticSearch feature

Once the features have been installed, you can change some of its properties. For example, to setup the URL where your ElasticSearch installation runs, change the *serverUrl* parameter in tsdr/persistence-elasticsearch/src/main/resources/configuration/initial/:

```
tsdr-persistence-elasticsearch.properties
```

All the data are stored into the TSDR index under a type. The metric data are stored under the metric type and the log data are store under the log type. You can modify the files in tsdr/persistence-elasticsearch/src/main/resources/configuration/initial/:

```
tsdr-persistence-elasticsearch_metric_mapping.json
tsdr-persistence-elasticsearch_log_mapping.json
```

to change or tune the mapping for those types. The changes in those files will be promoted after the feature is reloaded or the distribution is restarted.

We can now test whether the setup is correct by downloading and installing mininet, which we use to send some data to the running ElasticSearch instance.

Installing the necessary features:

```
start OpenDaylight
feature:install odl-restconf odl-l2switch-switch odl-tsdr-core odl-tsdr-openflow-
↪statistics-collector
feature:install odl-tsdr-elasticsearch
```

We can check whether the distribution is now listening on port 6653:

```
netstat -an | grep 6653
```

Run mininet

```
sudo mn --topo single,3 --controller 'remote,ip=distro_ip,port=6653' --switch ovsk,
↪protocols=OpenFlow13
```

where the distro_ip is the IP address of the machine where the OpenDaylight distribution is running. This command will create three hosts connected to one OpenFlow capable switch.

We can check if data was stored by ElasticSearch in TSDR by running the following command:

```
tsdr:list FLOWTABLESTATS
```

The output should look similar to the following:

```
[NID=openflow:1][DC=FLOWTABLESTATS][MN=ActiveFlows][RK=Node:openflow:1,
↪Table:50][TS=1473427383598][3]
[NID=openflow:1][DC=FLOWTABLESTATS][MN=PacketMatch][RK=Node:openflow:1,
↪Table:50][TS=1473427383598][12]
[NID=openflow:1][DC=FLOWTABLESTATS][MN=PacketLookup][RK=Node:openflow:1,
↪Table:50][TS=1473427383598][12]
[NID=openflow:1][DC=FLOWTABLESTATS][MN=ActiveFlows][RK=Node:openflow:1,
↪Table:80][TS=1473427383598][3]
[NID=openflow:1][DC=FLOWTABLESTATS][MN=PacketMatch][RK=Node:openflow:1,
↪Table:80][TS=1473427383598][17]
[NID=openflow:1][DC=FLOWTABLESTATS][MN=PacketMatch][RK=Node:openflow:1,
↪Table:246][TS=1473427383598][19]
...
```

Or you can query your ElasticSearch instance:

```
curl -XPOST "http://elasticseach_ip:9200/_search?pretty" -d'{ "from": 0, "size":␣
↪10000, "query": { "match_all": {} } }'
```

The elasticseach_ip is the IP address of the server where the ElasticSearch is running.

Web Activity Collector

The Web Activity Collector records the meaningful REST requests made through the OpenDaylight RESTCONF interface.

- Install some other feature that has a RESTCONF interface, for example. "odl-tsdr-syslog-collector"
- Issue a RESTCONF command that uses either POST,PUT or DELETE. For example, you could call the register-filter RPC of tsdr-syslog-collector.
- Look up data in TSDR database from Karaf.

  ```
  tsdr:list RESTCONF
  ```

- You should see the request that you have sent, along with its information (URL, HTTP method, requesting IP address and request body)
- Try to send a GET request, then check again, your request should not be registered, because the collector does not register GET requests by default.
- Open the file: "etc/tsdr.restconf.collector.cfg", and add GET to the list of METHODS_TO_LOG, so that it becomes:

  ```
  METHODS_TO_LOG=POST,PUT,DELETE,GET
  ```

  - Try again to issue your GET request, and check if it was recorded this time, it should be recorder.
  - Try manipulating the other properties (PATHS_TO_LOG (which URLs do we want to log from), REMOTE_ADDRESSES_TO_LOG (which requesting IP addresses do we want to log from) and CONTENT_TO_LOG (what should be in the request's body in order to log it)), and see if the requests are getting logged.

  - Try providing invalid properties (unknown methods for the METHODS_TO_LOG parameter, or the same method repeated multiple times, and invalid regular expressions for the other parameters), then check karaf's log using "log:display". It should tell you that the value is invalid, and that it will use the default value instead.

Troubleshooting

Karaf logs

All TSDR features and components write logging information including information messages, warnings, errors and debug messages into karaf.log.

HBase and Cassandra logs

For HBase and Cassandra data stores, the database level logs are written into HBase log and Cassandra logs.

- HBase log
  - HBase log is under <HBase-installation-directory>/logs/.
- Cassandra log
  - Cassandra log is under {cassandra.logdir}/system.log. The default {cassandra.logdir} is /var/log/cassandra/.

Security

TSDR gets the data from a variety of sources, which can be secured in different ways.

- OpenFlow Security
  - The OpenFlow data can be configured with Transport Layer Security (TLS) since the OpenFlow Plugin that TSDR depends on provides this security support.
- SNMP Security
  - The SNMP version3 has security support. However, since ODL SNMP Plugin that TSDR depends on does not support version 3, we (TSDR) will not have security support at this moment.
- NetFlow Security
  - NetFlow, which cannot be configured with security so we recommend making sure it flows only over a secured management network.
- Syslog Security
  - Syslog, which cannot be configured with security so we recommend making sure it flows only over a secured management network.

Support multiple data stores simultaneously at runtime

TSDR supports running multiple data stores simultaneously at runtim. For example, it is possible to configure TSDR to push log type of data into Cassandra data store, while pushing metrics type of data into HBase.

When you install one TSDR data store from karaf console, such as using feature:install odl-tsdr-hsqldb, a properties file will be generated under <Karaf-distribution-directory>/etc/. For example, when you install hsqldb, a file called tsdr-persistence-hsqldb.properties is generated under that directory.

By default, all the types of data are supported in the data store. For example, the default content of tsdr-persistence-hsqldb.properties is as follows:

```
metric-persistency=true
log-persistency=true
binary-persistency=true
```

When the user would like to use different data stores to support different types of data, he/she could enable or disable a particular type of data persistence in the data stores by configuring the properties file accordingly.

For example, if the user would like to store the log type of data in HBase, and store the metric and binary type of data in Cassandra, he/she needs to install both hbase and cassandra data stores from Karaf console. Then the user needs to modify the properties file under <Karaf-distribution-directory>/etc as follows:

- tsdr-persistence-hbase.properties

```
metric-persistency=false
log-persistency=true
binary-persistency=true
```

- tsdr-persistence-cassandra.properties

```
metric-psersistency=true
log-persistency=false
binary-persistency=false
```

TTP CLI Tools User Guide

Overview

Table Type Patterns are a specification developed by the Open Networking Foundation to enable the description and negotiation of subsets of the OpenFlow protocol. This is particularly useful for hardware switches that support OpenFlow as it enables the to describe what features they do (and thus also what features they do not) support. More details can be found in the full specification listed on the OpenFlow specifications page.

TTP CLI Tools Architecture

The TTP CLI Tools use the TTP Model and the YANG Tools/RESTCONF codecs to translate between the Data Transfer Objects (DTOs) and JSON/XML.

User Network Interface Manager Plug-in (Unimgr) User Guide

Overview

The User Network Interface (UNI) Manager project within OpenDaylight provides data models and APIs that enable software applications and service orchestrators to configure and provision connectivity services; in particular, Carrier Ethernet services as defined by MEF Forum, in physical and virtual network elements.

MEF has defined the Lifecycle Service Orchestration (LSO) Reference Architecture for the management and control of domains and entities that enable cooperative network services across one or more service provider networks. The

architecture also identifies LSO Reference Points, which are the logical points of interaction between specific functional management components. These LSO Reference Points are further defined by interface profiles and instantiated by APIs.

The LSO Reference Architecture is shown below. Note that this is a functional architecture that does not describe how the management components are implemented (e.g., single vs. multiple instances), but rather identifies management components that provide logical functionality as well as the points of interaction among them.

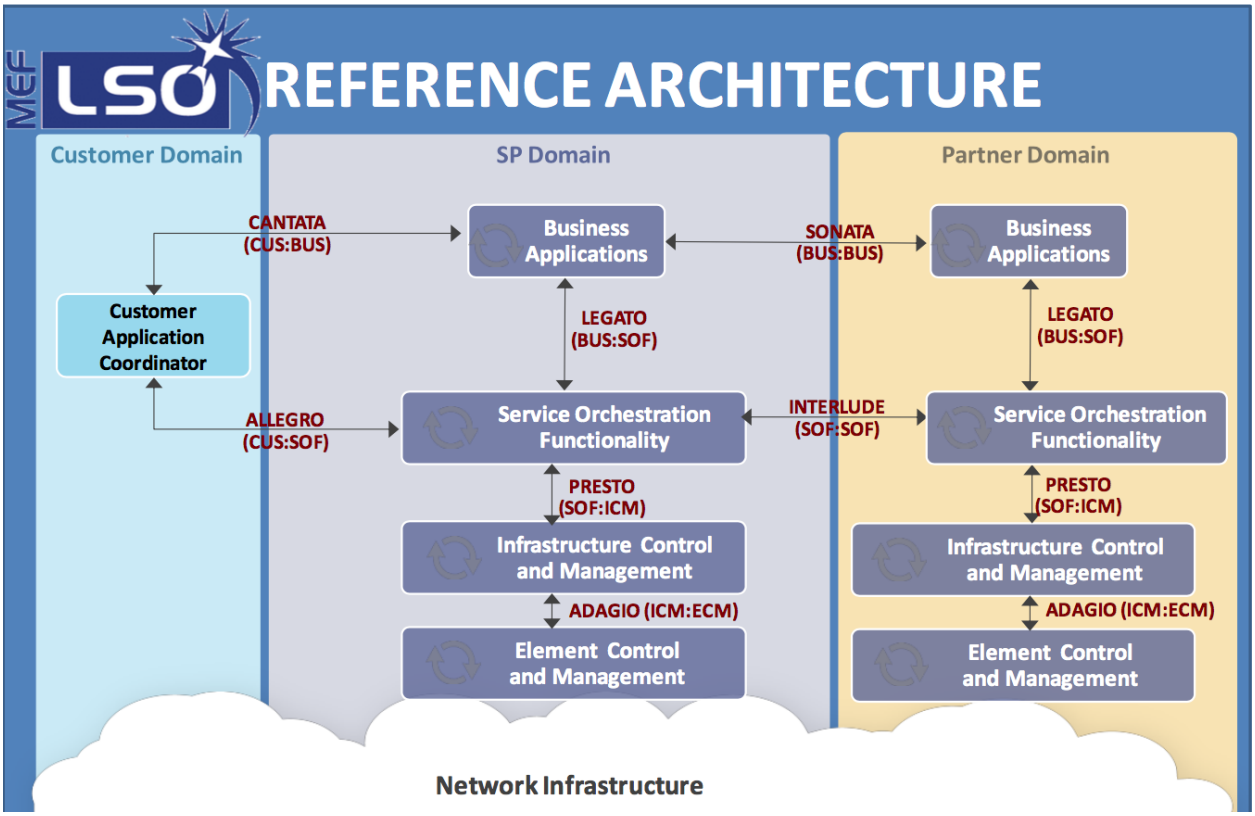


Fig. 1.112: MEF LSO Reference Architecture

Unimgr provides support for both the Legato as well as the Presto interfaces. These interfaces, and the APIs associated with them, are defined by YANG models developed within MEF in collaboration with ONF and IETF. For the Carbon release, these are as follows:

Legato YANG modules: https://git.opendaylight.org/gerrit/gitweb?p=unimgr.git;a=tree;f=legato-api/src/main/yang;hb=refs/heads/stable/carbon

Presto YANG modules: https://git.opendaylight.org/gerrit/gitweb?p=unimgr.git;a=tree;f=presto-api/src/main/yang;hb=refs/heads/stable/carbon

An application/user can interact with Unimgr at either the service orchestration layer (Legato) or the network resource provisioning layer (Presto).

Unimgr Architecture

Unimgr is comprised of the following OpenDaylight Karaf features:

odl-unimgr-api	OpenDaylight :: UniMgr :: api
odl-unimgr	OpenDaylight :: UniMgr
odl-unimgr-console	OpenDaylight :: UniMgr :: CLI
odl-unimgr-rest	OpenDaylight :: UniMgr :: REST
odl-unimgr-ui	OpenDaylight :: UniMgr :: UI

Configuring Unimgr

After launching OpenDaylight, install the feature for Unimgr. From the karaf command prompt execute the following command:

```
$ feature:install odl-unimgr-ui
```

Explore and exercise the Unimgr REST API

To see the Unimgr API, browse to this URL: http://localhost:8181/apidoc/explorer/index.html

Replace localhost with the IP address or hostname where OpenDaylight is running if you are not running OpenDaylight locally on your machine.

See also the Unimgr Developer Guide for a full listing of the API.

Unified Secure Channel

This document describes how to use the Unified Secure Channel (USC) feature in OpenDaylight. This document contains configuration, administration, and management sections for the feature.

Overview

In enterprise networks, more and more controller and network management systems are being deployed remotely, such as in the cloud. Additionally, enterprise networks are becoming more heterogeneous - branch, IoT, wireless (including cloud access control). Enterprise customers want a converged network controller and management system solution. This feature is intended for device and network administrators looking to use unified secure channels for their systems.

USC Channel Architecture

- USC Agent
  - The USC Agent provides proxy and agent functionality on top of all standard protocols supported by the device. It initiates call-home with the controller, maintains live connections with with the controller, acts as a demuxer/muxer for packets with the USC header, and authenticates the controller.
- USC Plugin
  - The USC Plugin is responsible for communication between the controller and the USC agent . It responds to call-home with the controller, maintains live connections with the devices, acts as a muxer/demuxer for packets with the USC header, and provides support for TLS/DTLS.
- USC Manager
  - The USC Manager handles configurations, high availability, security, monitoring, and clustering support for USC.

- USC UI
  - The USC UI is responsible for displaying a graphical user interface representing the state of USC in the OpenDaylight DLUX UI.

Installing USC Channel

To install USC, download OpenDaylight and use the Karaf console to install the following feature:

odl-usc-channel-ui

Configuring USC Channel

This section gives details about the configuration settings for various components in USC.

The USC configuration files for the Karaf distribution are located in distribution/karaf/target/assembly/etc/usc

- certificates
  - The certificates folder contains the client key, pem, and rootca files as is necessary for security.
- akka.conf
  - This file contains configuration related to clustering. Potential configuration properties can be found on the akka website at http://doc.akka.io
- usc.properties
  - This file contains configuration related to USC. Use this file to set the location of certificates, define the source of additional akka configurations, and assign default settings to the USC behavior.

Administering or Managing USC Channel

After installing the odl-usc-channel-ui feature from the Karaf console, users can administer and manage USC channels from the the UI or APIDOCS explorer.

Go to http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/index.html, sign in, and click on the USC side menu tab. From there, users can view the state of USC channels.

Go to http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/apidoc/explorer/index.html, sign in, and expand the usc-channel panel. From there, users can execute various API calls to test their USC deployment such as add-channel, delete-channel, and view-channel.

Tutorials

Below are tutorials for USC Channel

Viewing USC Channel

The purpose of this tutorial is to view USC Channel

Overview

This tutorial walks users through the process of viewing the USC Channel environment topology including established channels connecting the controllers and devices in the USC topology.

Prerequisites

For this tutorial, we assume that a device running a USC agent is already installed.

Instructions

- Run the OpenDaylight distribution and install odl-usc-channel-ui from the Karaf console.
- Go to http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/apidoc/explorer/index.html
- Execute add-channel with the following json data:
  - {“input”:{“channel”:{“hostname”:”127.0.0.1”,”port”:1068,”remote”:false}}}
- Go to http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/index.html
- Click on the USC side menu tab.
- The UI should display a table including the added channel from step 3.

Virtual Tenant Network (VTN)

VTN Overview

OpenDaylight Virtual Tenant Network (VTN) is an application that provides multi-tenant virtual network on an SDN controller.

Conventionally, huge investment in the network systems and operating expenses are needed because the network is configured as a silo for each department and system. So, various network appliances must be installed for each tenant and those boxes cannot be shared with others. It is a heavy work to design, implement and operate the entire complex network.

The uniqueness of VTN is a logical abstraction plane. This enables the complete separation of logical plane from physical plane. Users can design and deploy any desired network without knowing the physical network topology or bandwidth restrictions.

VTN allows the users to define the network with a look and feel of conventional L2/L3 network. Once the network is designed on VTN, it will automatically be mapped into underlying physical network, and then configured on the individual switch leveraging SDN control protocol. The definition of logical plane makes it possible not only to hide the complexity of the underlying network but also to better manage network resources. It achieves reducing reconfiguration time of network services and minimizing network configuration errors.

It is implemented as two major components

- *VTN Manager*
- *VTN Coordinator*

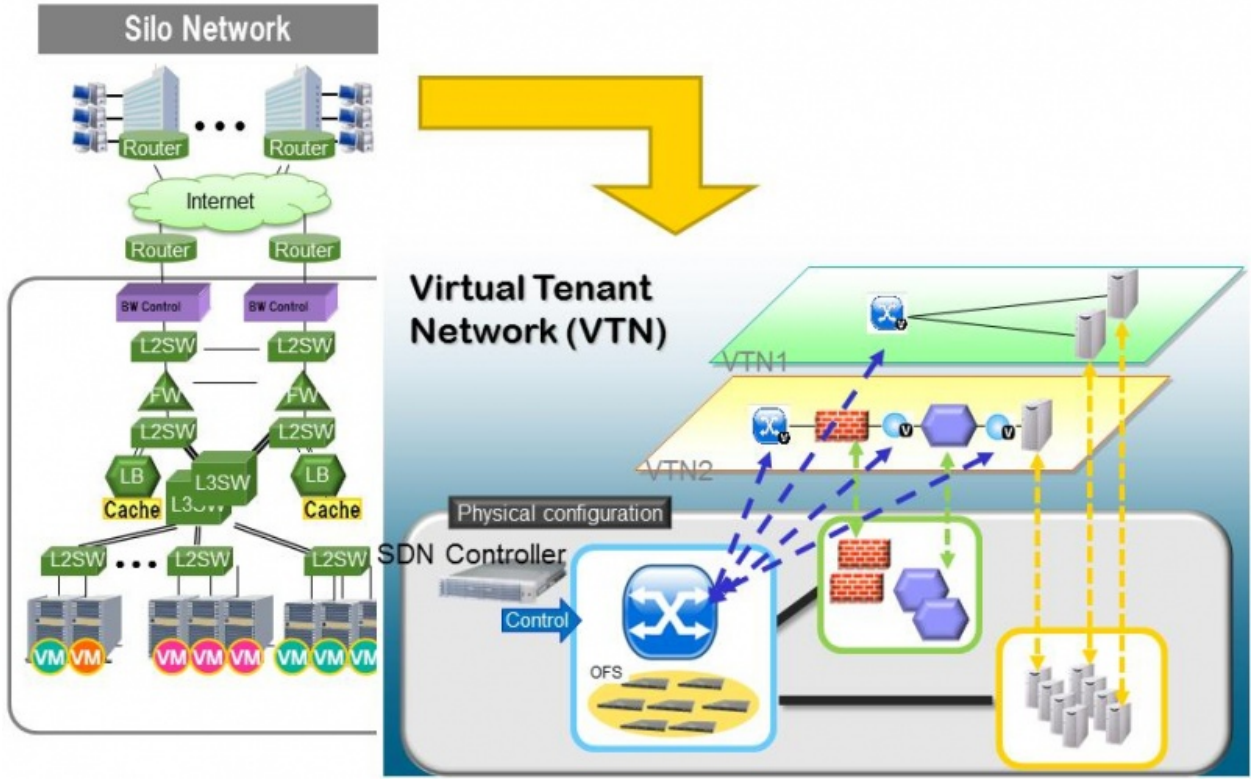


Fig. 1.113: VTN Overview

VTN Manager

An OpenDaylight Plugin that interacts with other modules to implement the components of the VTN model. It also provides a REST interface to configure VTN components in OpenDaylight. VTN Manager is implemented as one plugin to the OpenDaylight. This provides a REST interface to create/update/delete VTN components. The user command in VTN Coordinator is translated as REST API to VTN Manager by the OpenDaylight Driver component. In addition to the above mentioned role, it also provides an implementation to the OpenStack L2 Network Functions API.

Features Overview

- **odl-vtn-manager** provides VTN Manager's JAVA API.
- **odl-vtn-manager-rest** provides VTN Manager's REST API.
- **odl-vtn-manager-neutron** provides the integration with Neutron interface.

REST API

VTN Manager provides REST API for virtual network functions.

Here is an example of how to create a virtual tenant network.

```
curl --user "admin":"admin" -H "Accept: application/json" -H \
"Content-type: application/json" -X POST \
```

```
http://localhost:8181/restconf/operations/vtn:update-vtn \
-d '{"input":{"tenant-name":"vtn1"}}'
```

You can check the list of all tenants by executing the following command.

```
curl --user "admin":"admin" -H "Accept: application/json" -H \
"Content-type: application/json" -X GET \
http://localhost:8181/restconf/operational/vtn:vtns
```

REST Conf documentation for VTN Manager, please refer to: https://nexus.opendaylight.org/content/sites/site/org.opendaylight.vtn/boron/manager.model/apidocs/index.html

VTN Coordinator

The VTN Coordinator is an external application that provides a REST interface for an user to use OpenDaylight VTN Virtualization. It interacts with VTN Manager plugin to implement the user configuration. It is also capable of multiple OpenDaylight orchestration. It realizes Virtual Tenant Network (VTN) provisioning in OpenDaylight instances. In the OpenDaylight architecture VTN Coordinator is part of the network application, orchestration and services layer. VTN Coordinator will use the REST interface exposed by the VTN Manger to realize the virtual network using OpenDaylight. It uses OpenDaylight APIs (REST) to construct the virtual network in OpenDaylight instances. It provides REST APIs for northbound VTN applications and supports virtual networks spanning across multiple OpenDaylight by coordinating across OpenDaylight.

For VTN Coordinator REST API, please refer to: https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_%28VTN%29:VTN_Coordinator:RestApi

Network Virtualization Function

The user first defines a VTN. Then, the user maps the VTN to a physical network, which enables communication to take place according to the VTN definition. With the VTN definition, L2 and L3 transfer functions and flow-based traffic control functions (filtering and redirect) are possible.

Virtual Network Construction

The following table shows the elements which make up the VTN. In the VTN, a virtual network is constructed using virtual nodes (vBridge, vRouter) and virtual interfaces and links. It is possible to configure a network which has L2 and L3 transfer function, by connecting the virtual intrefaces made on virtual nodes via virtual links.

vBridge	The logical representation of L2 switch function.
vRouter	The logical representation of router function.
vTep	The logical representation of Tunnel End Point - TEP.
vTunnel	The logical representation of Tunnel.
vBypass	The logical representation of connectivity between controlled networks.
Virtual interface	The representation of end point on the virtual node.
Virtual Linkv(vLink)	The logical representation of L1 connectivity between virtual interfaces.

The following figure shows an example of a constructed virtual network. VRT is defined as the vRouter, BR1 and BR2 are defined as vBridges. interfaces of the vRouter and vBridges are connected using vLinks.

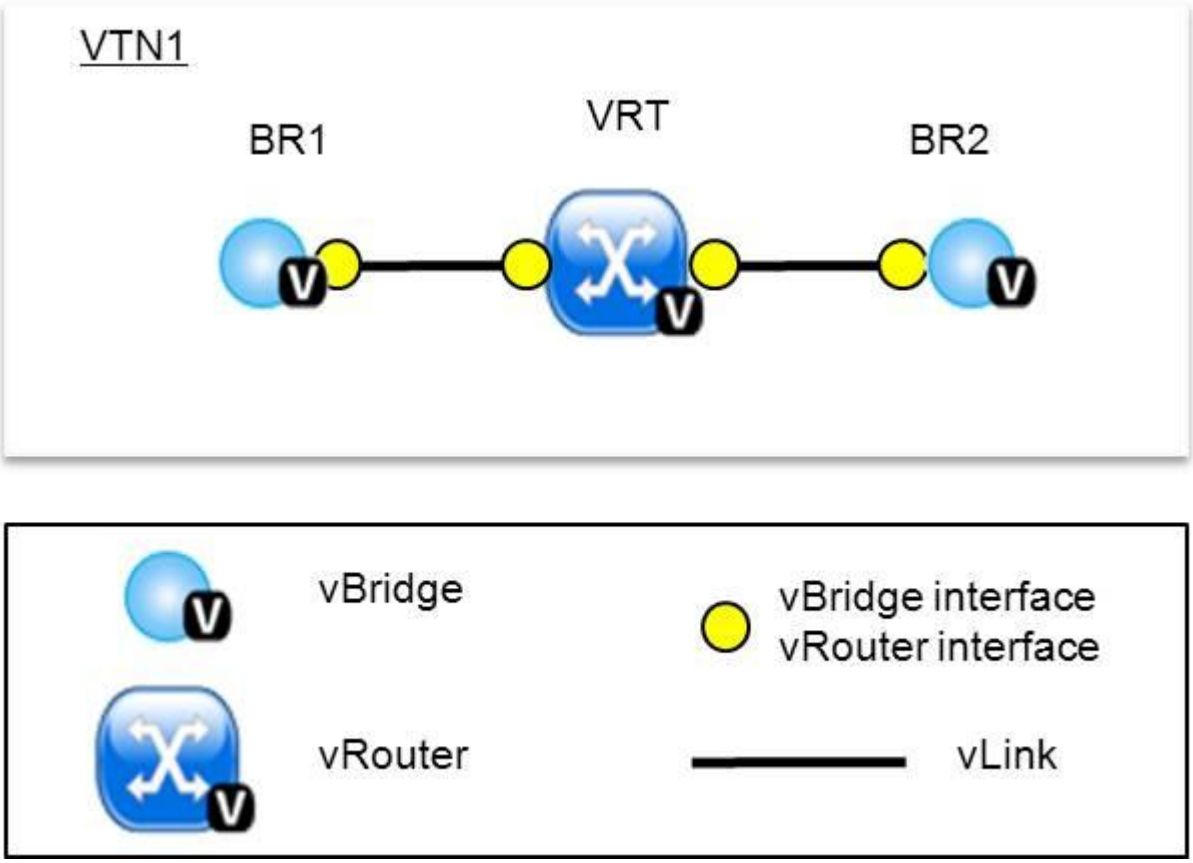


Fig. 1.114: VTN Construction

Mapping of Physical Network Resources

Map physical network resources to the constructed virtual network. Mapping identifies which virtual network each packet transmitted or received by an OpenFlow switch belongs to, as well as which interface in the OpenFlow switch transmits or receives that packet. There are two mapping methods. When a packet is received from the OFS, port mapping is first searched for the corresponding mapping definition, then VLAN mapping is searched, and the packet is mapped to the relevant vBridge according to the first matching mapping.

Port mapping	Maps physical network resources to an interface of vBridge using Switch ID, Port ID and VLAN ID of the incoming L2 frame. Untagged frame mapping is also supported.
VLAN mapping	Maps physical network resources to a vBridge using VLAN ID of the incoming L2 frame.Maps physical resources of a particular switch to a vBridge using switch ID and VLAN ID of the incoming L2 frame.
MAC mapping	Maps physical resources to an interface of vBridge using MAC address of the incoming L2 frame(The initial contribution does not include this method).

VTN can learn the terminal information from a terminal that is connected to a switch which is mapped to VTN. Further, it is possible to refer that terminal information on the VTN.

- Learning terminal information VTN learns the information of a terminal that belongs to VTN. It will store the MAC address and VLAN ID of the terminal in relation to the port of the switch.
- Aging of terminal information Terminal information, learned by the VTN, will be maintained until the packets from terminal keep flowing in VTN. If the terminal gets disconnected from the VTN, then the aging timer will start clicking and the terminal information will be maintained till timeout.

The following figure shows an example of mapping. An interface of BR1 is mapped to port GBE0/1 of OFS1 using port mapping. Packets received from GBE0/1 of OFS1 are regarded as those from the corresponding interface of BR1. BR2 is mapped to VLAN 200 using VLAN mapping. Packets with VLAN tag 200 received from any ports of any OFSs are regarded as those from an interface of BR2.

vBridge Functions

The vBridge provides the bridge function that transfers a packet to the intended virtual port according to the destination MAC address. The vBridge looks up the MAC address table and transmits the packet to the corresponding virtual interface when the destination MAC address has been learned. When the destination MAC address has not been learned, it transmits the packet to all virtual interfaces other than the receiving port (flooding). MAC addresses are learned as follows.

- MAC address learning The vBridge learns the MAC address of the connected host. The source MAC address of each received frame is mapped to the receiving virtual interface, and this MAC address is stored in the MAC address table created on a per-vBridge basis.
- MAC address aging The MAC address stored in the MAC address table is retained as long as the host returns the ARP reply. After the host is disconnected, the address is retained until the aging timer times out. To have the vBridge learn MAC addresses statically, you can register MAC addresses manually.

vRouter Functions

The vRouter transfers IPv4 packets between vBridges. The vRouter supports routing, ARP learning, and ARP aging functions. The following outlines the functions.

- Routing function When an IP address is registered with a virtual interface of the vRouter, the default routing information for that interface is registered. It is also possible to statically register routing information for a virtual interface.

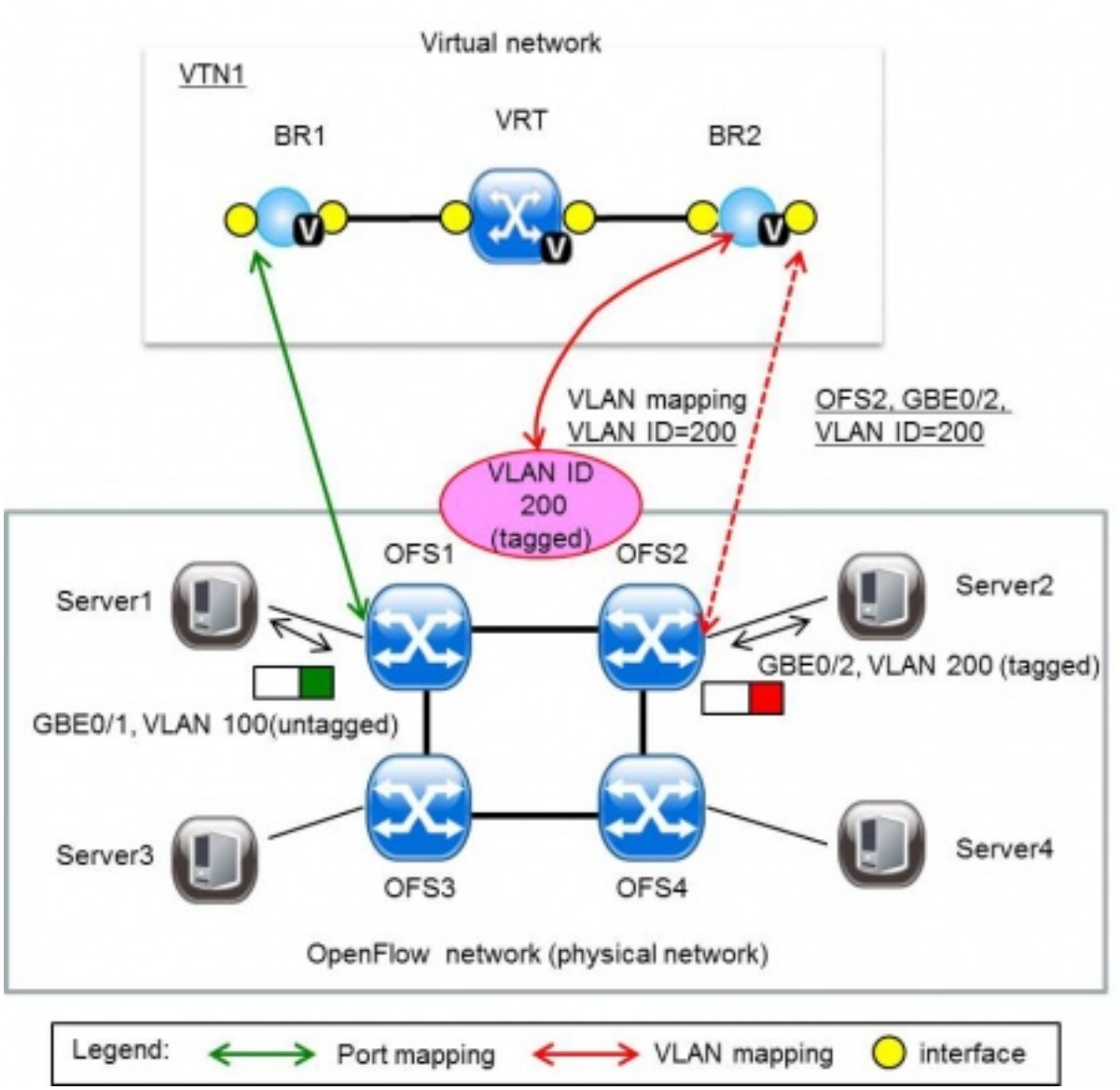


Fig. 1.115: VTN Mapping

- ARP learning function The vRouter associates a destination IP address, MAC address and a virtual interface, based on an ARP request to its host or a reply packet for an ARP request, and maintains this information in an ARP table prepared for each routing domain. The registered ARP entry is retained until the aging timer, described later, times out. The vRouter transmits an ARP request on an individual aging timer basis and deletes the associated entry from the ARP table if no reply is returned. For static ARP learning, you can register ARP entry information manually.
- DHCP relay agent function The vRouter also provides the DHCP relay agent function.

Flow Filter Functions

Flow Filter function is similar to ACL. It is possible to allow or prohibit communication with only certain kind of packets that meet a particular condition. Also, it can perform a processing called Redirection - WayPoint routing, which is different from the existing ACL. Flow Filter can be applied to any interface of a vNode within VTN, and it is possible to the control the packets that pass interface. The match conditions that could be specified in Flow Filter are as follows. It is also possible to specify a combination of multiple conditions.

- Source MAC address
- Destination MAC address
- MAC ether type
- VLAN Priority
- Source IP address
- Destination IP address
- DSCP
- IP Protocol
- TCP/UDP source port
- TCP/UDP destination port
- ICMP type
- ICMP code

The types of Action that can be applied on packets that match the Flow Filter conditions are given in the following table. It is possible to make only those packets, which match a particular condition, to pass through a particular server by specifying Redirection in Action. E.g., path of flow can be changed for each packet sent from a particular terminal, depending upon the destination IP address. VLAN priority control and DSCP marking are also supported.

Action	Function
Pass	Pass particular packets matching the specified conditions.
Drop	Discards particular packets matching the specified conditions.
Redirection	Redirects the packet to a desired virtual interface. Both Transparent Redirection (not changing MAC address) and Router Redirection (changing MAC address) are supported.

The following figure shows an example of how the flow filter function works.

If there is any matching condition specified by flow filter when a packet being transferred within a virtual network goes through a virtual interface, the function evaluates the matching condition to see whether the packet matches it. If the packet matches the condition, the function applies the matching action specified by flow filter. In the example shown in the figure, the function evaluates the matching condition at BR1 and discards the packet if it matches the condition.

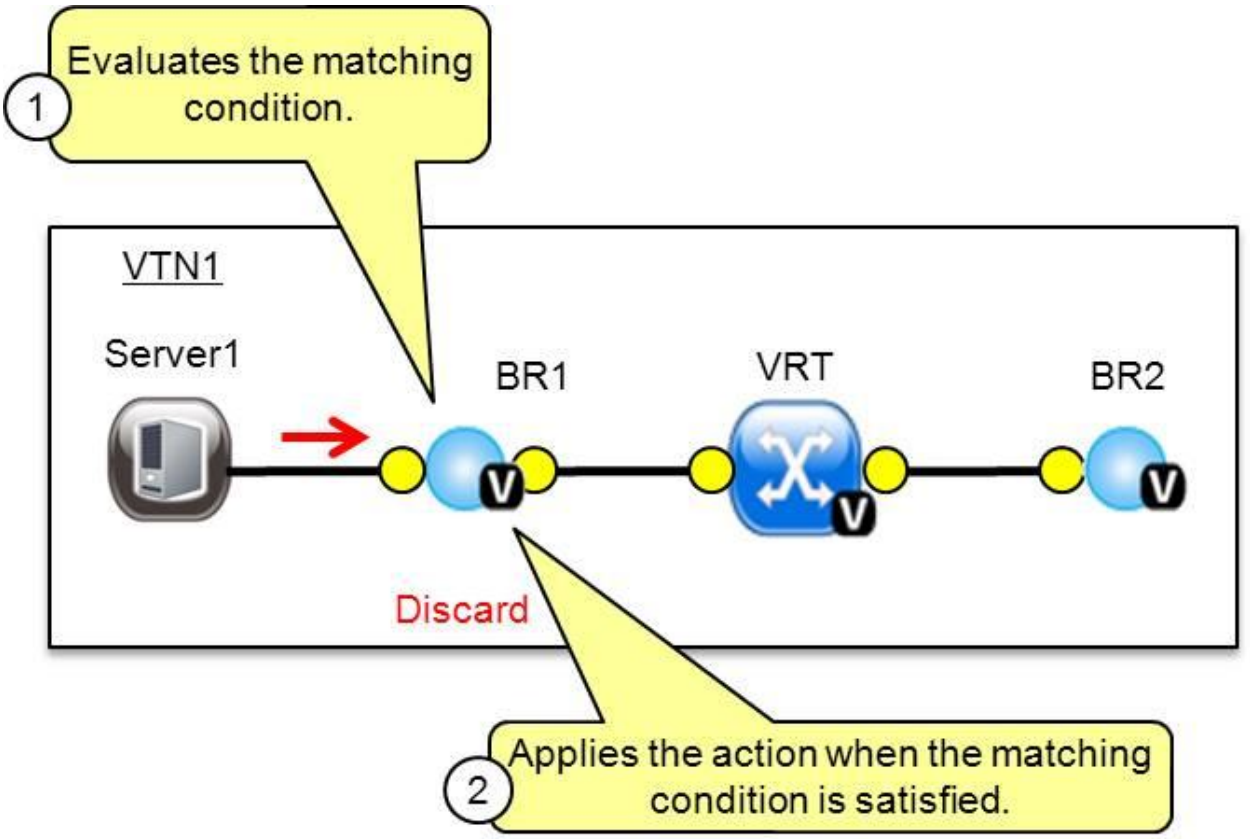


Fig. 1.116: VTN FlowFilter

Multiple SDN Controller Coordination

With the network abstractions, VTN enables to configure virtual network across multiple SDN controllers. This provides highly scalable network system.

VTN can be created on each SDN controller. If users would like to manage those multiple VTNs with one policy, those VTNs can be integrated to a single VTN.

As a use case, this feature is deployed to multi data center environment. Even if those data centers are geographically separated and controlled with different controllers, a single policy virtual network can be realized with VTN.

Also, one can easily add a new SDN Controller to an existing VTN or delete a particular SDN Controller from VTN.

In addition to this, one can define a VTN which covers both OpenFlow network and Overlay network at the same time.

Flow Filter, which is set on the VTN, will be automatically applied on the newly added SDN Controller.

Coordination between OpenFlow Network and L2/L3 Network

It is possible to configure VTN on an environment where there is mix of L2/L3 switches as well. L2/L3 switch will be shown on VTN as vBypass. Flow Filter or policing cannot be configured for a vBypass. However, it is possible to treat it as a virtual node inside VTN.

Virtual Tenant Network (VTN) API

VTN provides Web APIs. They are implemented by REST architecture and provide the access to resources within VTN that are identified by URI. User can perform the operations like GET/PUT/POST/DELETE against the virtual network resources (e.g. vBridge or vRouter) by sending a message to VTN through HTTPS communication in XML or JSON format.

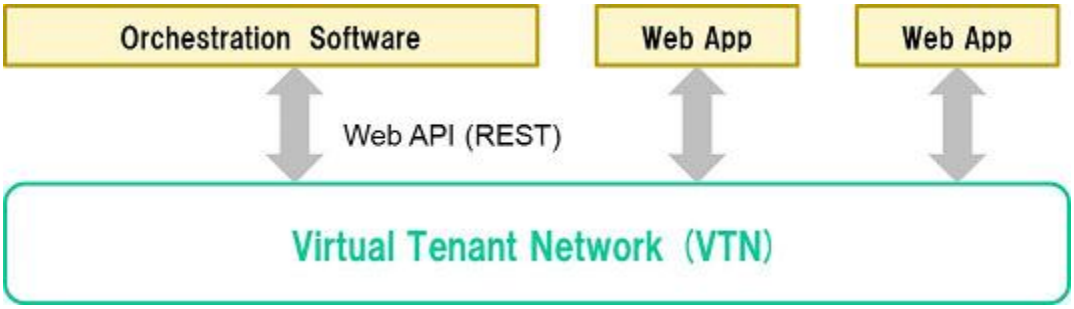


Fig. 1.117: VTN API

Function Outline

VTN provides following operations for various network resources.

Resources	GET	POST	PUT	DELETE
VTN	Yes	Yes	Yes	Yes
vBridge	Yes	Yes	Yes	Yes
vRouter	Yes	Yes	Yes	Yes
vTep	Yes	Yes	Yes	Yes
vTunnel	Yes	Yes	Yes	Yes
vBypass	Yes	Yes	Yes	Yes
vLink	Yes	Yes	Yes	Yes
Interface	Yes	Yes	Yes	Yes
Port map	Yes	No	Yes	Yes
Vlan map	Yes	Yes	Yes	Yes
Flowfilter (ACL/redirect)	Yes	Yes	Yes	Yes
Controller information	Yes	Yes	Yes	Yes
Physical topology information	Yes	No	No	No
Alarm information	Yes	No	No	No

Example usage

The following is an example of the usage to construct a virtual network.

- Create VTN

```
 curl --user admin:adminpass -X POST -H 'content-type: application/json'  \
-d '{"vtn":{"vtn_name":"VTN1"}}' http://172.1.0.1:8083/vtn-webapi/vtns.json
```

- Create Controller Information

```
 curl --user admin:adminpass -X POST -H 'content-type: application/json'  \
-d '{"controller": {"controller_id":"CONTROLLER1","ipaddr":"172.1.0.1","type":"odc",
↪"username":"admin", \
"password":"admin","version":"1.0"}}' http://172.1.0.1:8083/vtn-webapi/controllers.
↪json
```

- Create vBridge under VTN

```
curl --user admin:adminpass -X POST -H 'content-type: application/json' \
-d '{"vbridge":{"vbr_name":"VBR1","controller_id": "CONTROLLER1","domain_id":
↪"(DEFAULT)"}}' \
http://172.1.0.1:8083/vtn-webapi/vtns/VTN1/vbridges.json
```

- Create the interface under vBridge

```
curl --user admin:adminpass -X POST -H 'content-type: application/json' \
-d '{"interface":{"if_name":"IF1"}}' http://172.1.0.1:8083/vtn-webapi/vtns/VTN1/
↪vbridges/VBR1/interfaces.json
```

VTN OpenStack Configuration

This guide describes how to set up OpenStack for integration with OpenDaylight Controller.

While OpenDaylight Controller provides several ways to integrate with OpenStack, this guide focus on the way which uses VTN features available on OpenDaylight. In the integration, VTN Manager work as network service provider for OpenStack.

VTN Manager features, enable OpenStack to work in pure OpenFlow environment in which all switches in data plane are OpenFlow switch.

Requirements

- OpenDaylight Controller. (VTN features must be installed)
- OpenStack Control Node.
- OpenStack Compute Node.
- OpenFlow Switch like mininet(Not Mandatory).

The VTN features support multiple OpenStack nodes. You can deploy multiple OpenStack Compute Nodes. In management plane, OpenDaylight Controller, OpenStack nodes and OpenFlow switches should communicate with each other. In data plane, Open vSwitches running in OpenStack nodes should communicate with each other through a physical or logical OpenFlow switches. The core OpenFlow switches are not mandatory. Therefore, you can directly connect to the Open vSwitch's.

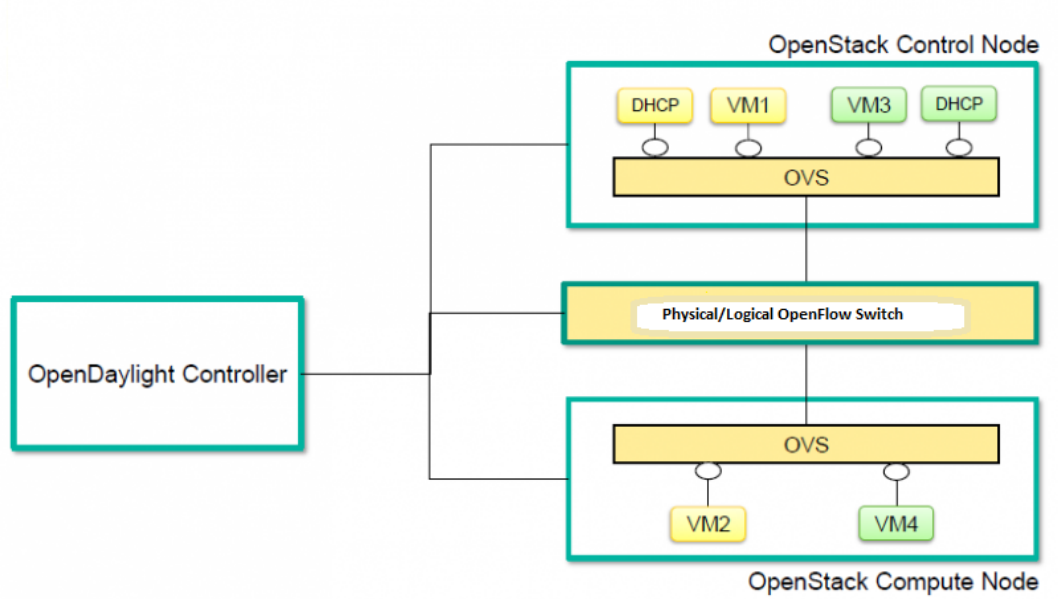


Fig. 1.118: Openstack Overview

Sample Configuration

Below steps depicts the configuration of single OpenStack Control node and OpenStack Compute node setup. Our test setup is as follows

Server Preparation

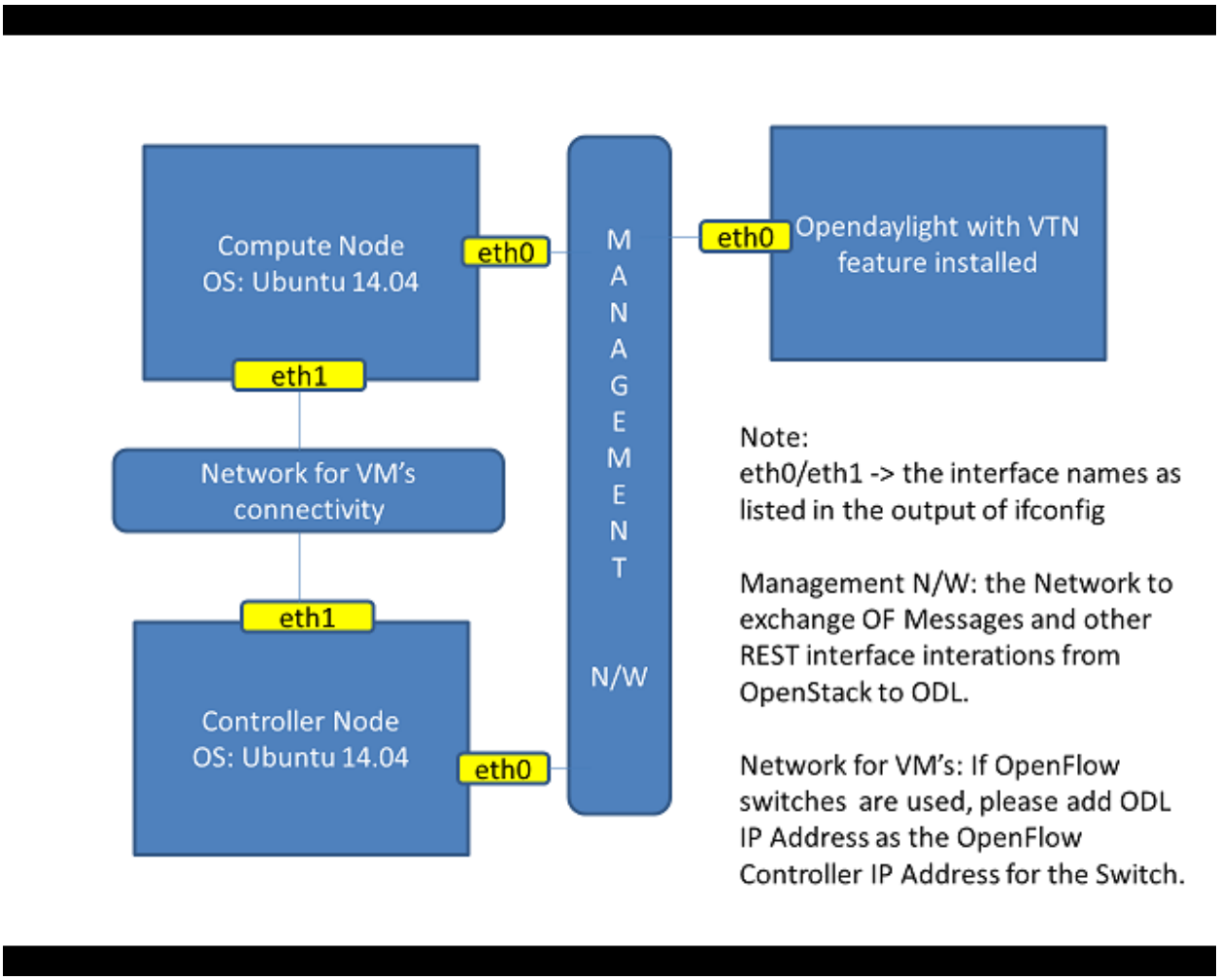


Fig. 1.119: LAB Setup

- Install Ubuntu 14.04 LTS in two servers (OpenStack Control node and Compute node respectively)
- While installing, Ubuntu mandates creation of a User, we created the user "stack"(We will use the same user for running devstack)
- Proceed with the below mentioned User Settings and Network Settings in both the Control and Compute nodes.

User Settings for devstack - Login to both servers - Disable Ubuntu Firewall

```
sudo ufw disable
```

- Install the below packages (optional, provides ifconfig and route coammnds, handy for debugging!!)

```
sudo apt-get install net-tools
```

- Edit sudo vim /etc/sudoers and add an entry as follows

```
stack ALL=(ALL) NOPASSWD: ALL
```

Network Settings - Checked the output of ifconfig -a, two interfaces were listed eth0 and eth1 as indicated in the image above. - We had connected eth0 interface to the Network where OpenDaylight is reachable. - eth1 interface in both servers were connected to a different network to act as data plane for the VM's created using the OpenStack. - Manually edited the file : sudo vim /etc/network/interfaces and made entries as follows

```
 stack@ubuntu-devstack:~/devstack$ cat /etc/network/interfaces
 # This file describes the network interfaces available on your system
 # and how to activate them. For more information, see interfaces(5).
 # The loop-back network interface
 auto lo
 iface lo inet loopback
 # The primary network interface
 auto eth0
 iface eth0 inet static
      address <IP_ADDRESS_TO_REACH_ODL>
      netmask <NET_MASK>
      broadcast <BROADCAST_IP_ADDRESS>
      gateway <GATEWAY_IP_ADDRESS>
auto eth1
iface eth1 inet static
     address <IP_ADDRESS_UNIQ>
     netmask <NETMASK>
```

Note: Please ensure that the eth0 interface is the default route and it is able to reach the ODL_IP_ADDRESS NOTE: The entries for eth1 are not mandatory, If not set, we may have to manually do "ifup eth1" after the stacking is complete to activate the interface

Finalize the user and network settings - Please reboot both nodes after the user and network settings to have the network settings applied to the network - Login again and check the output of ifconfig to ensure that both interfaces are listed

OpenDaylight Settings and Execution

VTN Configuration for OpenStack Integration:

- VTN uses the configuration parameters from "90-vtn-neutron.xml" file for the OpenStack integration.

- These values will be set for the OpenvSwitch, in all the participating OpenStack nodes.
- A configuration file "90-vtn-neutron.xml" will be generated automatically by following the below steps,
- Download the latest Boron karaf distribution from the below link,

```
http://www.opendaylight.org/software/downloads
```

- cd "distribution-karaf-0.5.0-Boron" and run karaf by using the following command "./bin/karaf".
- Install the below feature to generate "90-vtn-neutron.xml"

```
feature:install odl-vtn-manager-neutron
```

- Logout from the karaf console and Check "90-vtn-neutron.xml" file from the following path "distribution-karaf-0.5.0-Boron/etc/opendaylight/karaf/".
- The contents of "90-vtn-neutron.xml" should be as follows:

bridgename=br-int portname=eth1 protocols=OpenFlow13 failmode=secure

- The values of the configuration parameters must be changed based on the user environment.
- Especially, "portname" should be carefully configured, because if the value is wrong, OpenDaylight fails to forward packets.
- Other parameters works fine as is for general use cases.
  - bridgename
    - The name of the bridge in Open vSwitch, that will be created by OpenDaylight Controller.
    - It must be "br-int".
  - portname
    - The name of the port that will be created in the vbridge in Open vSwitch.
    - This must be the same name of the interface of OpenStack Nodes which is used for interconnecting OpenStack Nodes in data plane.(in our case:eth1)
    - By default, if 90-vtn-neutron.xml is not created, VTN uses ens33 as portname.
  - protocols
    - OpenFlow protocol through which OpenFlow Switch and Controller communicate.
    - The values can be OpenFlow13 or OpenFlow10.
  - failmode
    - The value can be "standalone" or "secure".
    - Please use "secure" for general use cases.

Start ODL Controller

- Please refer to the Installation Pages to run ODL with VTN Feature enabled.
- After running ODL Controller, please ensure ODL Controller listens to the ports:6633,6653, 6640 and 8080
- Please allow the ports in firewall for the devstack to be able to communicate with ODL Controller.

Note:

- 6633/6653 - OpenFlow Ports
- 6640 - OVS Manager Port
- 8080 - Port for REST API

Devstack Setup

Get Devstack (All nodes)

- Install git application using
  - sudo apt-get install git
- Get devstack
  - git clone https://git.openstack.org/openstack-dev/devstack;
- Switch to stable/Juno Version branch
  - cd devstack

```
git checkout stable/juno
```

Note: If you want to use stable/kilo Version branch, Please execute the below command in devstack folder

```
git checkout stable/kilo
```

Note: If you want to use stable/liberty Version branch, Please execute the below command in devstack folder

```
git checkout stable/liberty
```

Stack Control Node

- local.conf:
- cd devstack in the controller node
- Copy the contents of local.conf for juno (devstack control node) from https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_(VTN):Scripts:devstack and save it as "local.conf" in the "devstack".
- Copy the contents of local.conf for kilo and liberty (devstack control node) from https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_(VTN):Scripts:devstack_post_juno_versions and save it as "local.conf" in the "devstack".
- Please modify the IP Address values as required.
- Stack the node

```
./stack.sh
```

Verify Control Node stacking

- stack.sh prints out Horizon is now available at http://<CONTROL_NODE_IP_ADDRESS>:8080/
- Execute the command *sudo ovs-vsctl show* in the control node terminal and verify if the bridge *br-int* is created.
- Typical output of the ovs-vsctl show is indicated below:

```
e232bbd5-096b-48a3-a28d-ce4a492d4b4f
   Manager "tcp:192.168.64.73:6640"
       is_connected: true
   Bridge br-int
       Controller "tcp:192.168.64.73:6633"
           is_connected: true
       fail_mode: secure
       Port "eth1"
          Interface "eth1"
   ovs_version: "2.0.2"
```

Stack Compute Node

- local.conf:
- cd devstack in the controller node
- Copy the contents of local.conf for juno (devstack compute node) from https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_(VTN):Scripts:devstack and save it as "local.conf" in the "devstack".
- Copy the contents of local.conf file for kilo and liberty (devstack compute node) from https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_(VTN):Scripts:devstack_post_juno_versions and save it as "local.conf" in the "devstack".
- Please modify the IP Address values as required.
- Stack the node

```
./stack.sh
```

Verify Compute Node Stacking

- stack.sh prints out This is your host ip: <COMPUTE_NODE_IP_ADDRESS>
- Execute the command *sudo ovs-vsctl show* in the control node terminal and verify if the bridge *br-int* is created.
- The output of the ovs-vsctl show will be similar to the one seen in control node.

Additional Verifications

- Please visit the OpenDaylight DLUX GUI after stacking all the nodes, http://<ODL_IP_ADDRESS>:8181/index.html. The switches, topology and the ports that are currently read can be validated.

```
http://<controller-ip>:8181/index.html
```

Tip: If the interconnected between the Open vSwitch is not seen, Please bring up the interface for the dataplane manually using the below comamnd

```
ifup <interface_name>
```

- Please Accept Promiscuous mode in the networks involving the interconnect.

Create VM from Devstack Horizon GUI

- Login to http://<CONTROL_NODE_IP>:8080/ to check the horizon GUI.

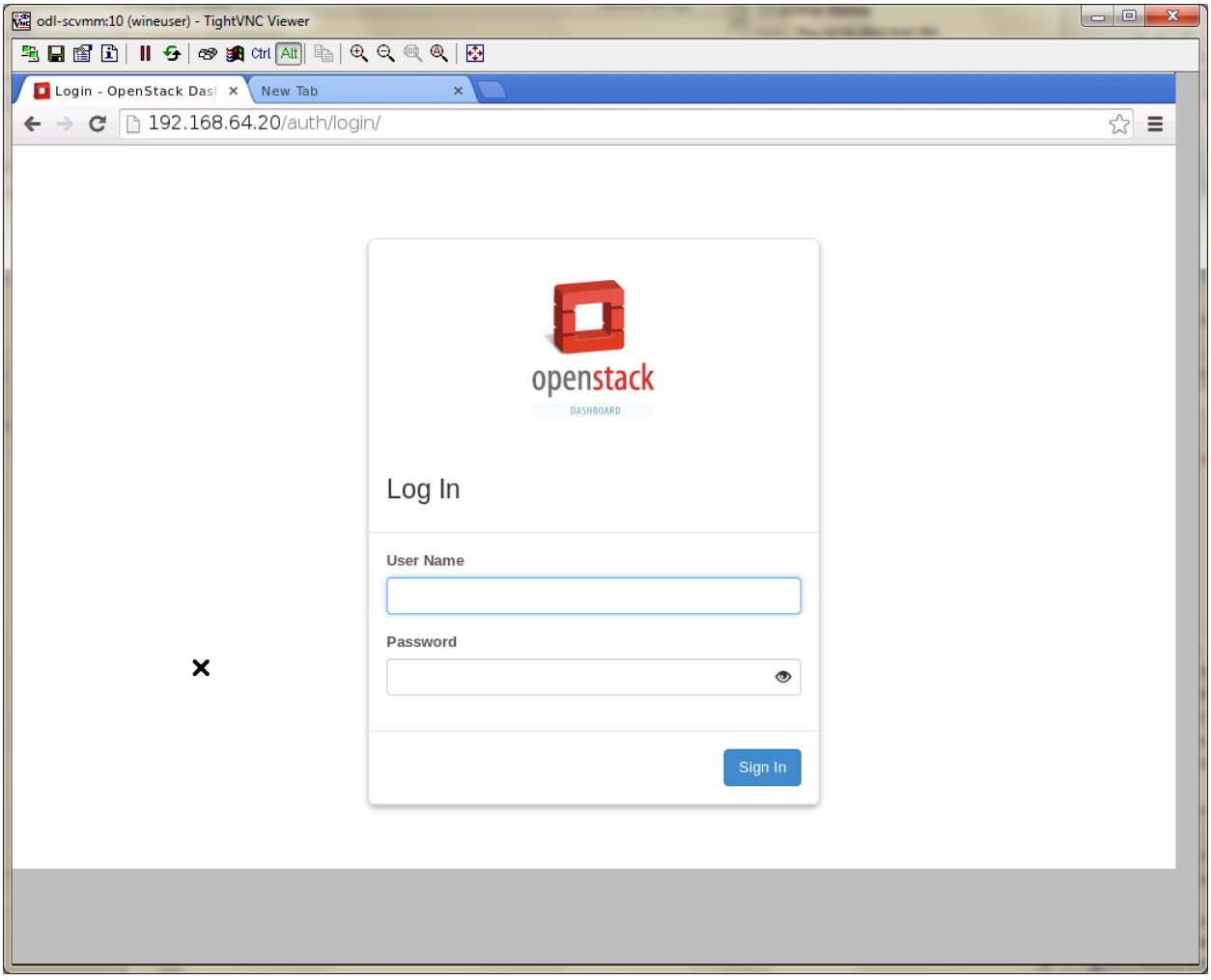


Fig. 1.120: Horizon GUI

Enter the value for User Name as admin and enter the value for Password as labstack.

- We should first ensure both the hypervisors(control node and compute node) are mapped under hypervisors by clicking on Hpervisors tab.
- Create a new Network from Horizon GUI.
- Click on Networks Tab.

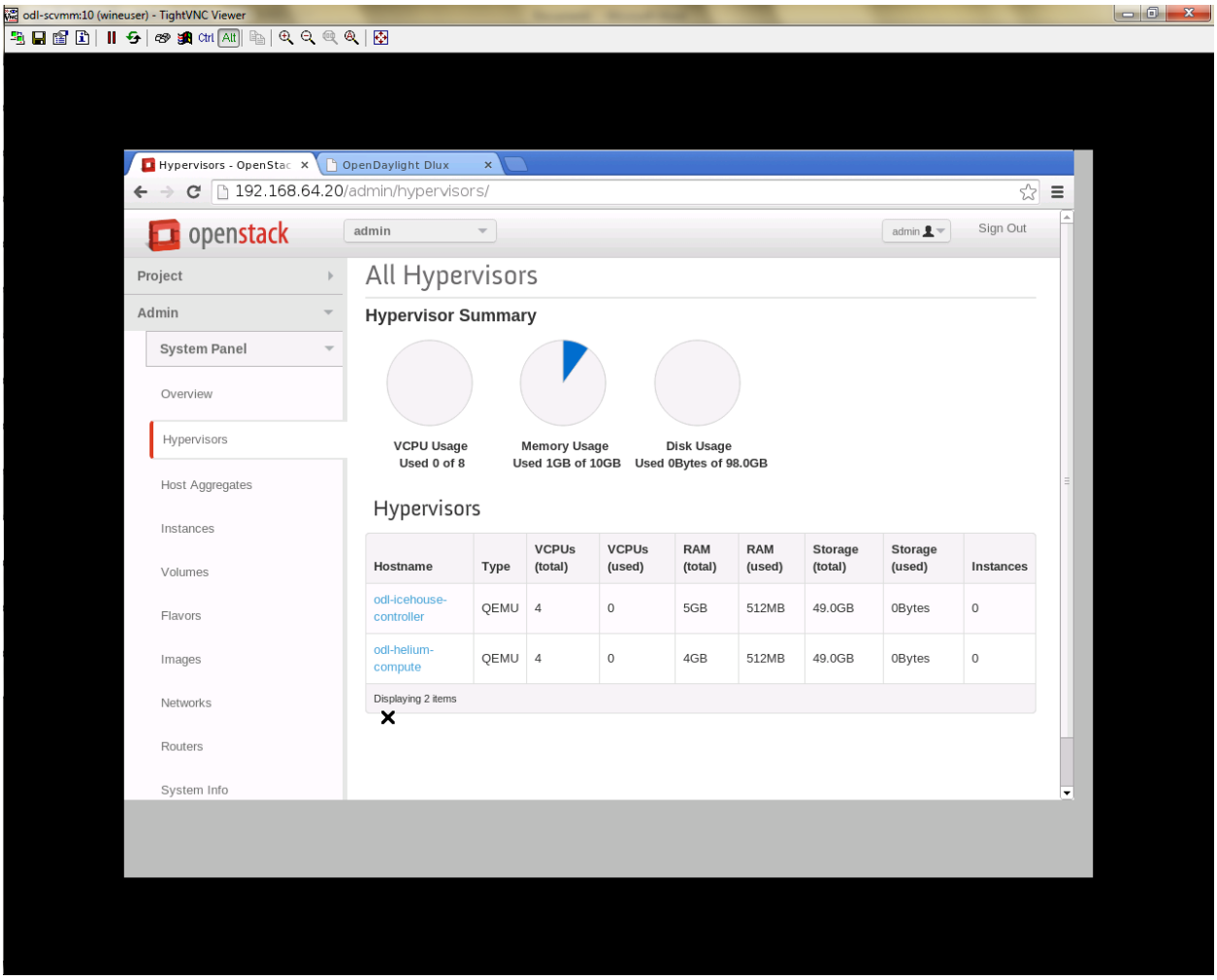


Fig. 1.121: Hypervisors

- click on the Create Network button.

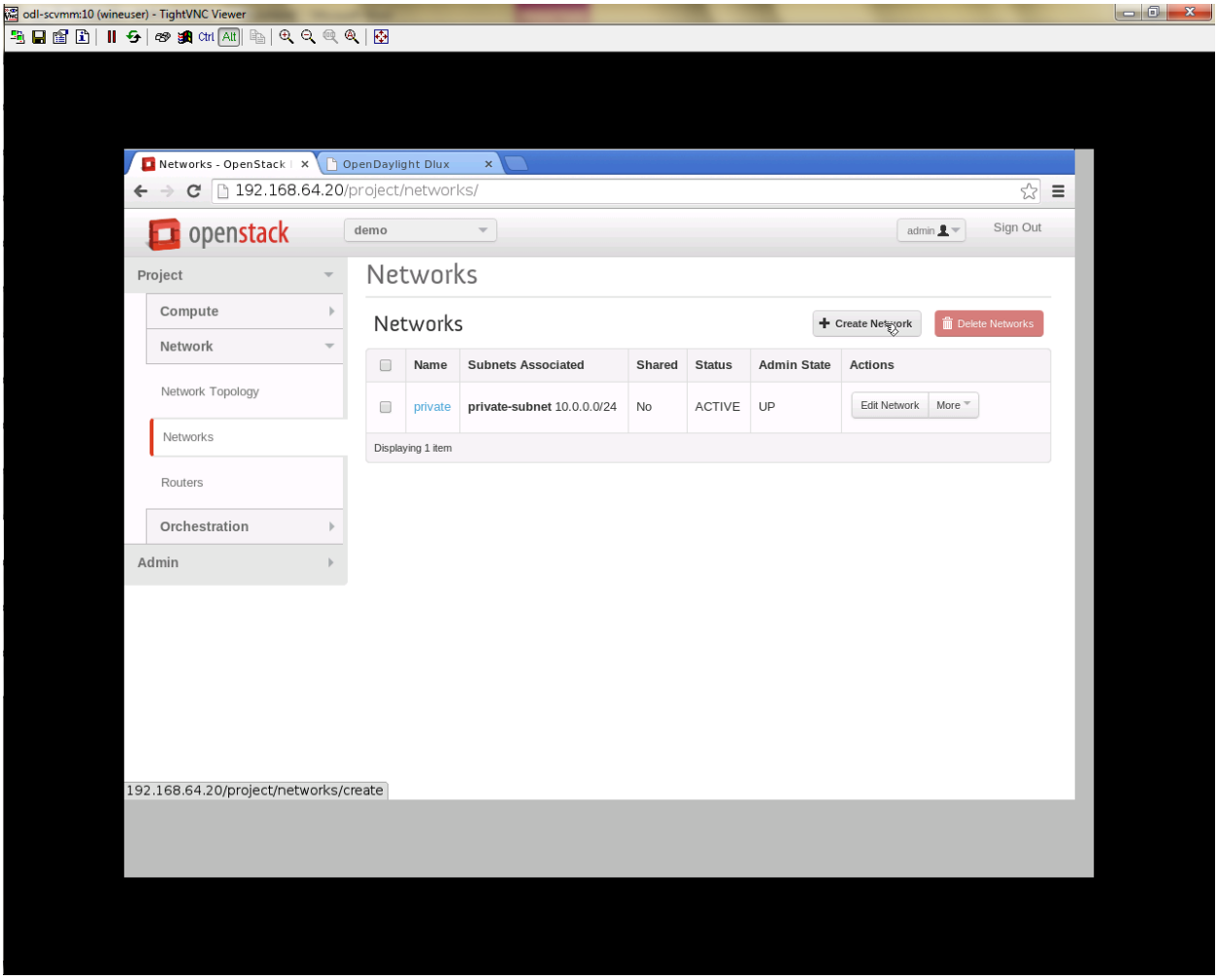


Fig. 1.122: Create Network

- A popup screen will appear.
- Enter network name and click Next button.
- Create a sub network by giving Network Address and click Next button .
- Specify the additional details for subnetwork (please refer the image for your reference).
- Click Create button
- Create VM Instance
- Navigate to Instances tab in the GUI.
- Click on Launch Instances button.
- Click on Details tab to enter the VM details.For this demo we are creating Ten VM's(instances).
- In the Networking tab, we must select the network,for this we need to drag and drop the Available networks to Selected Networks (i.e.,) Drag vtn1 we created from Available networks to Selected Networks and click Launch to create the instances.
- Ten VM's will be created.

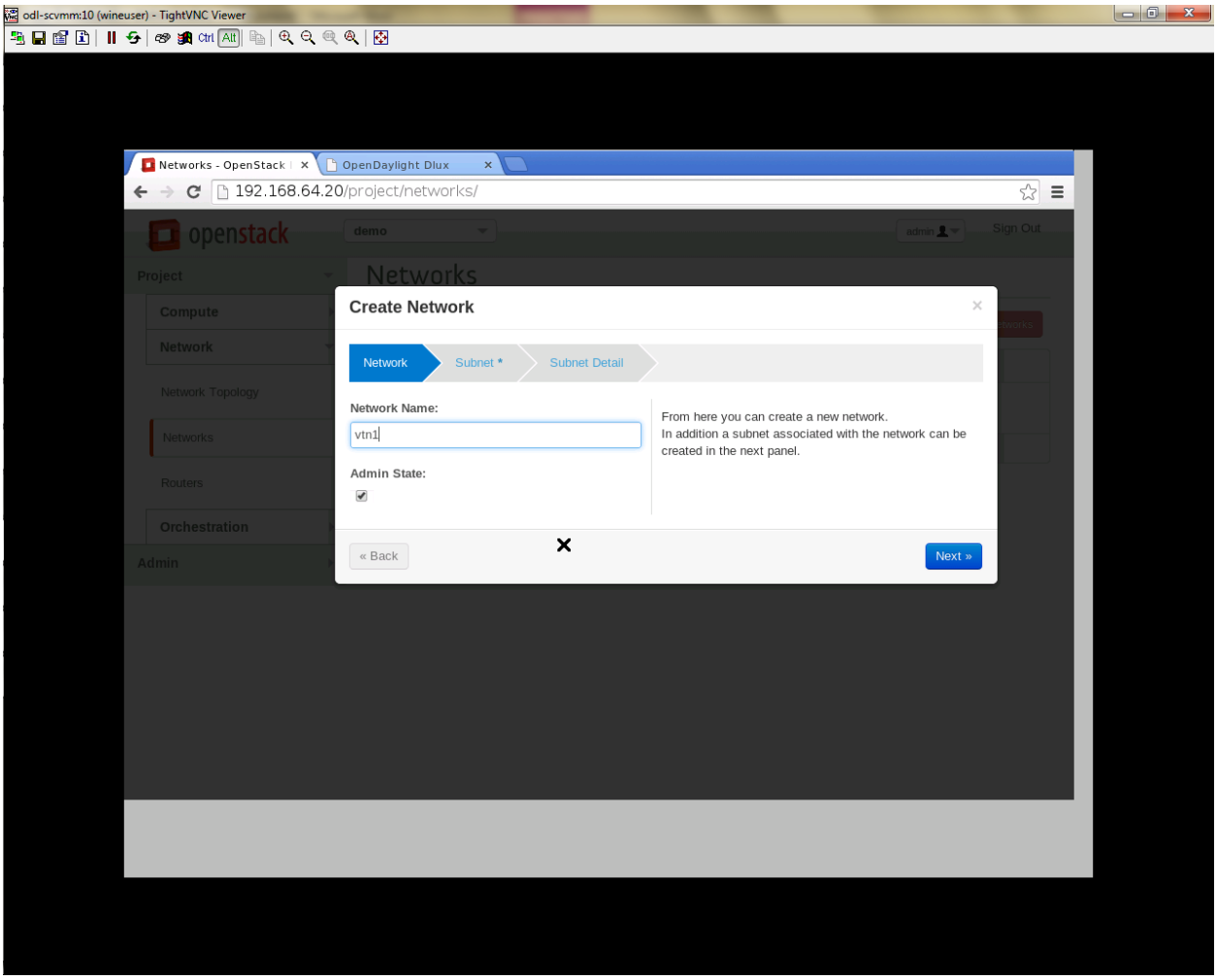


Fig. 1.123: Step 1

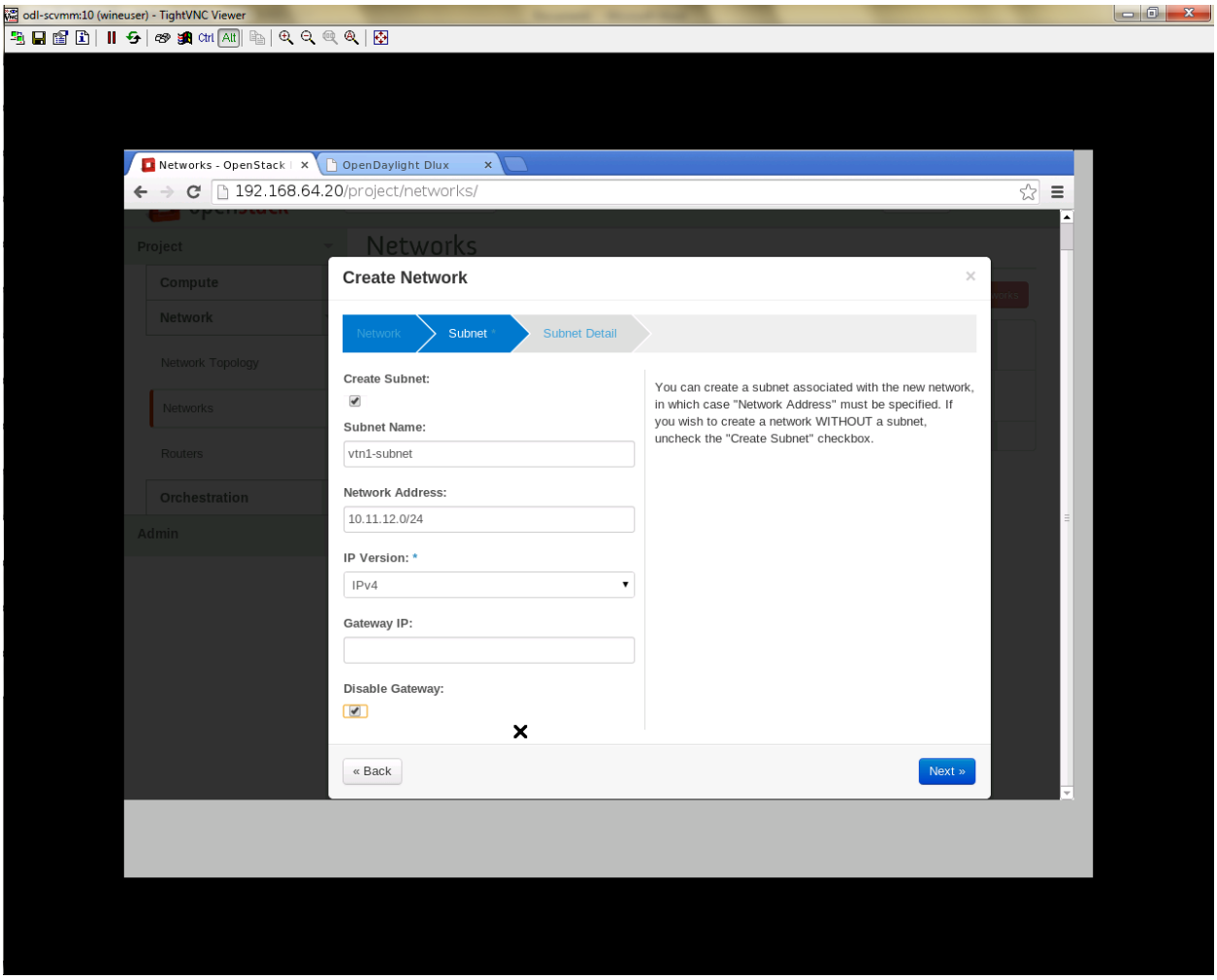


Fig. 1.124: Step 2

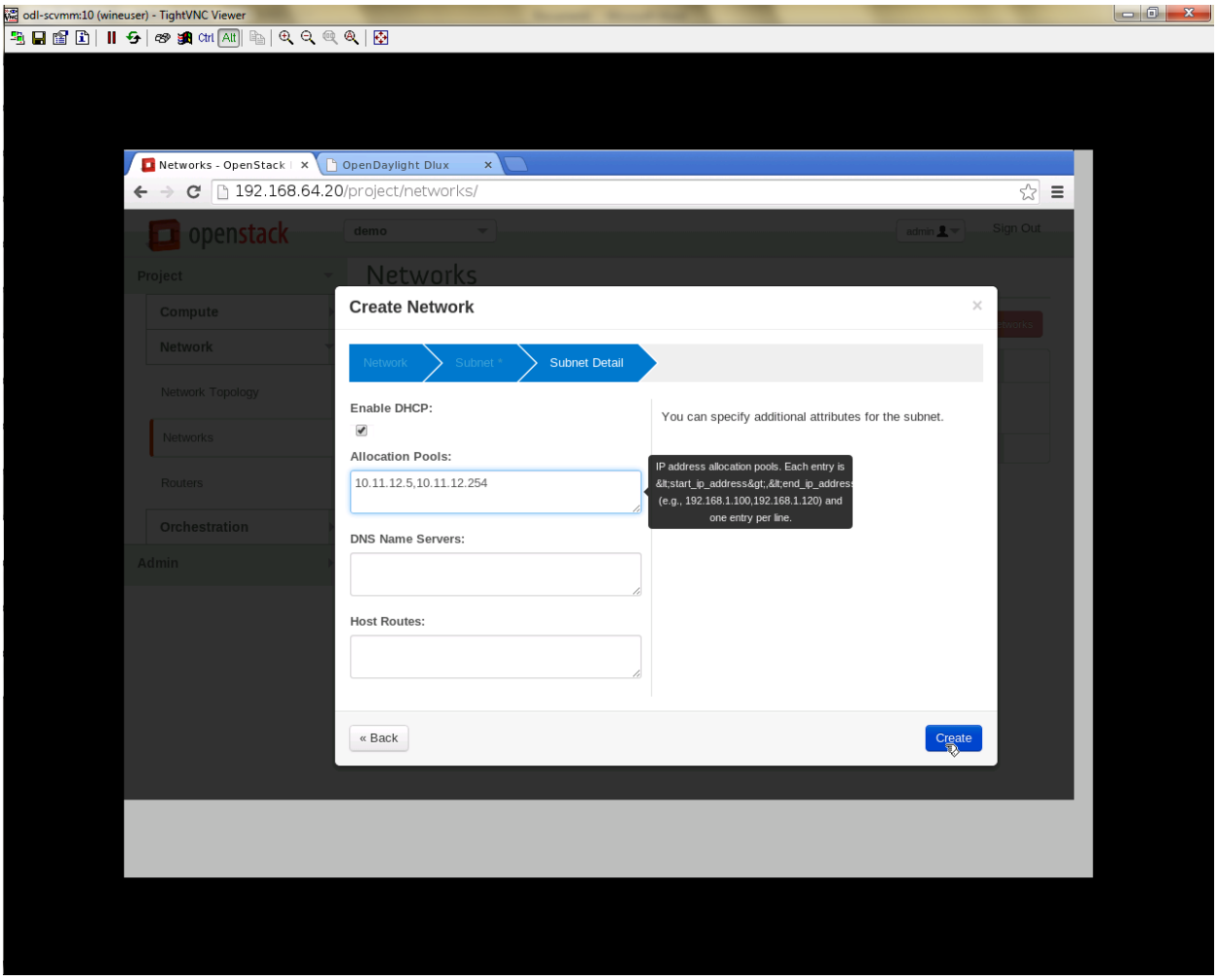


Fig. 1.125: Step 3

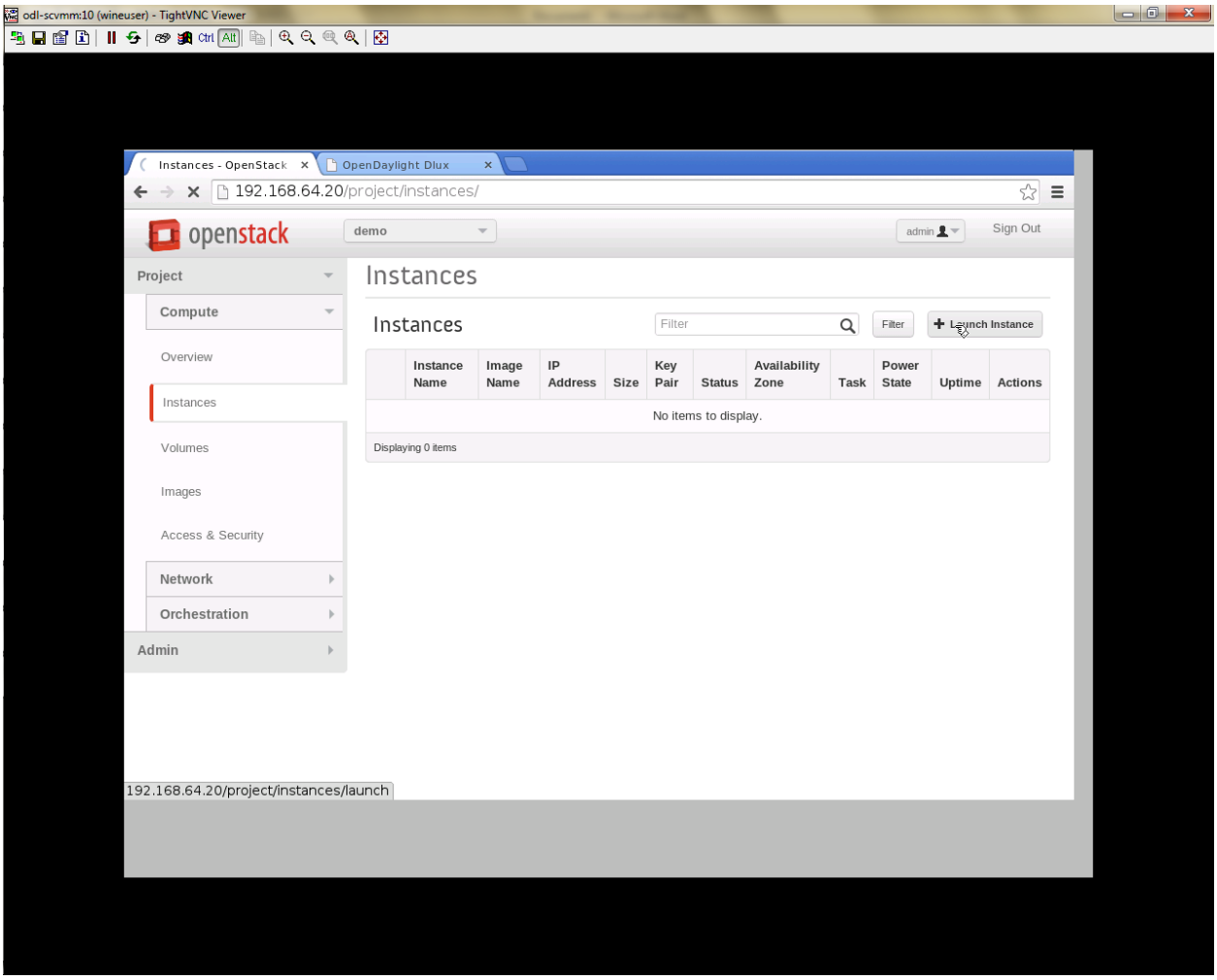


Fig. 1.126: Instance Creation

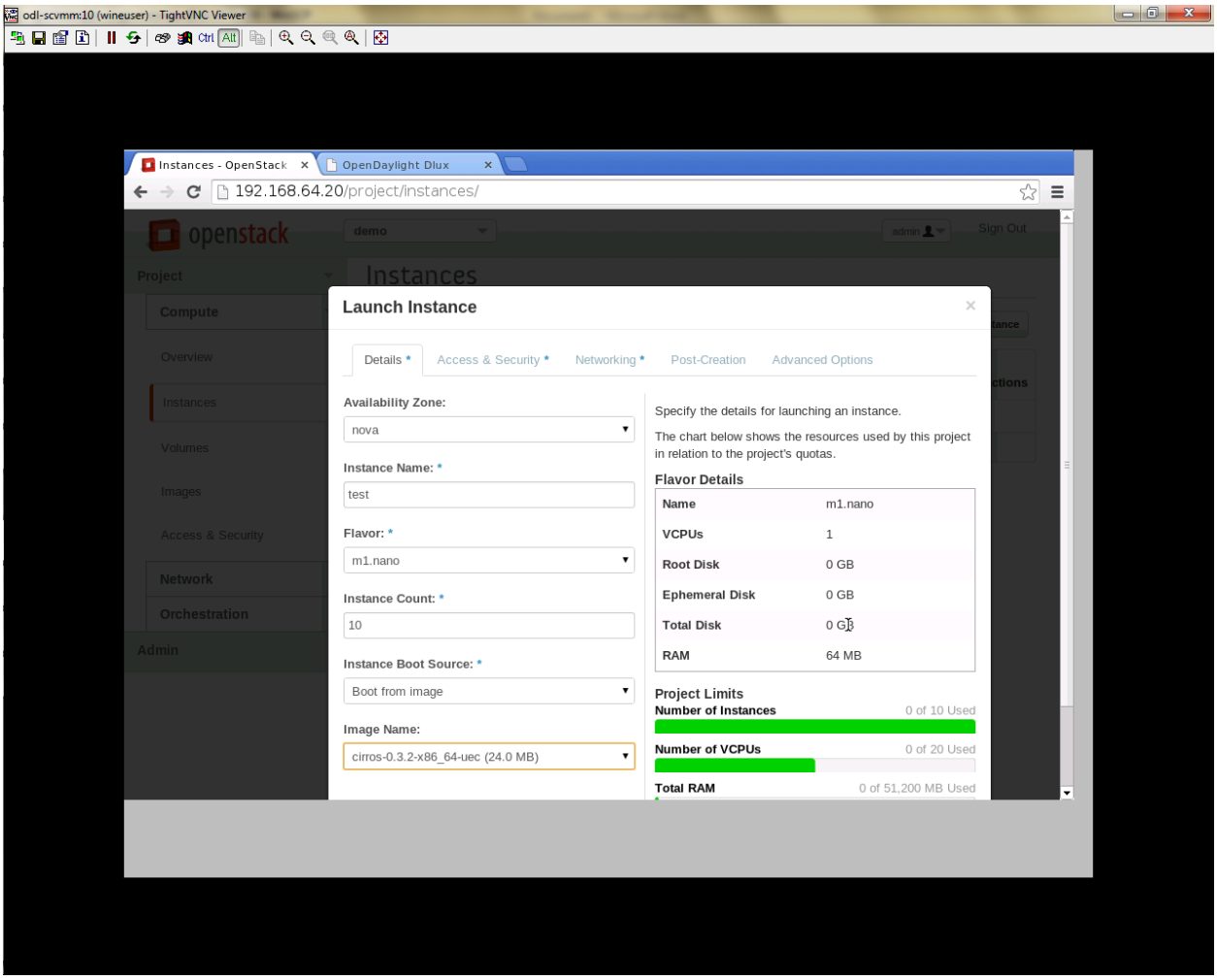


Fig. 1.127: Launch Instance

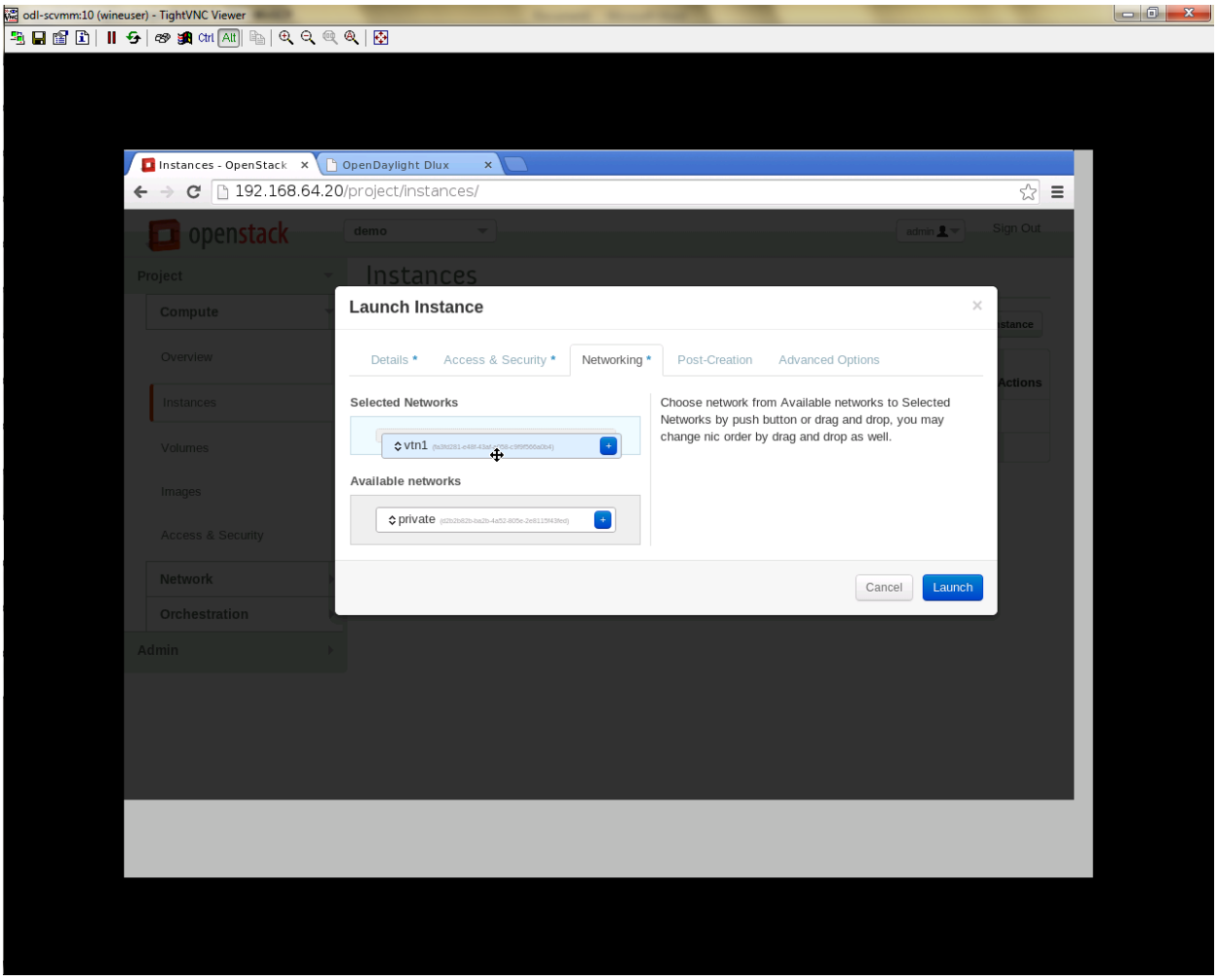


Fig. 1.128: Launch Network

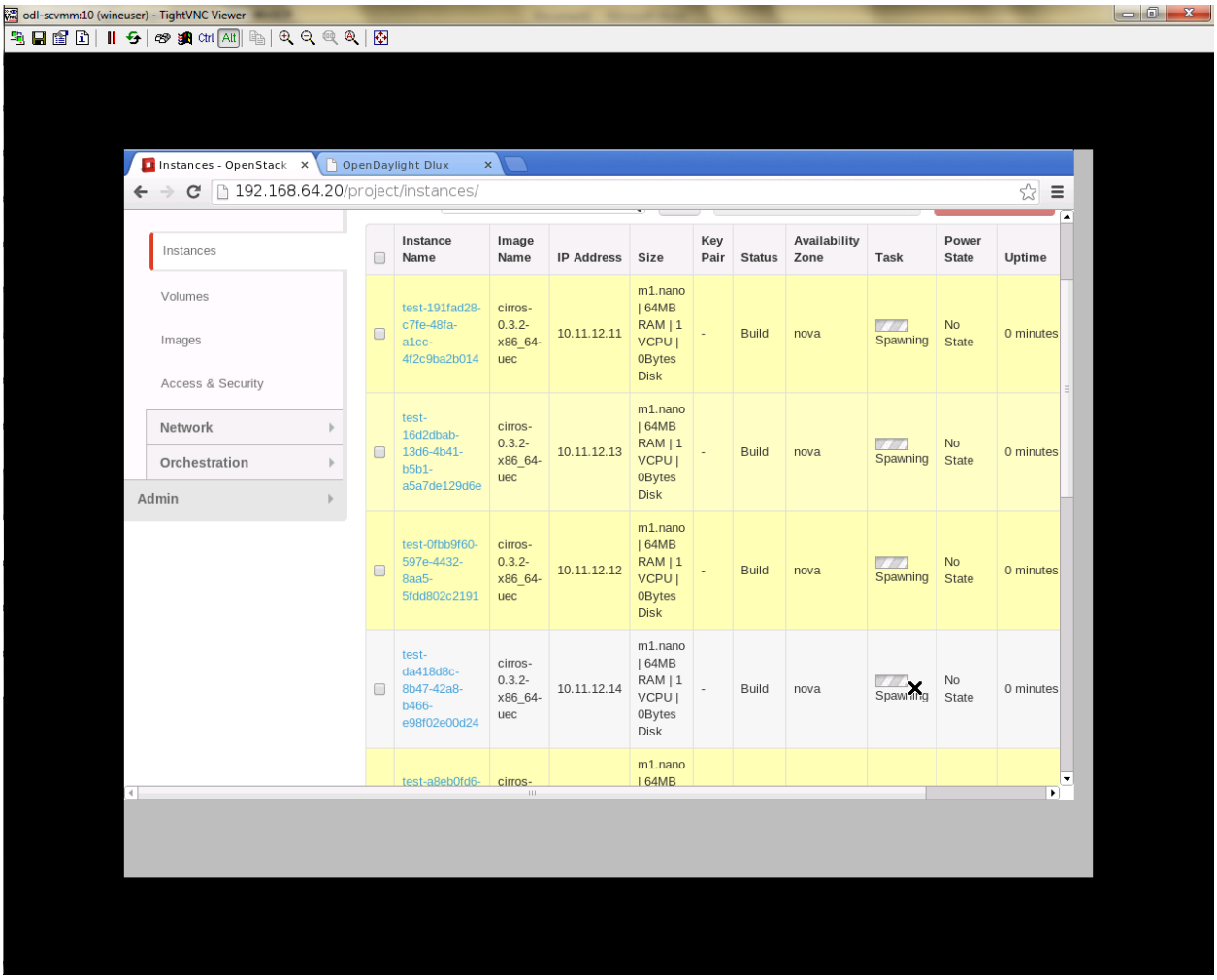


Fig. 1.129: Load All Instances

- Click on any VM displayed in the Instances tab and click the Console tab.

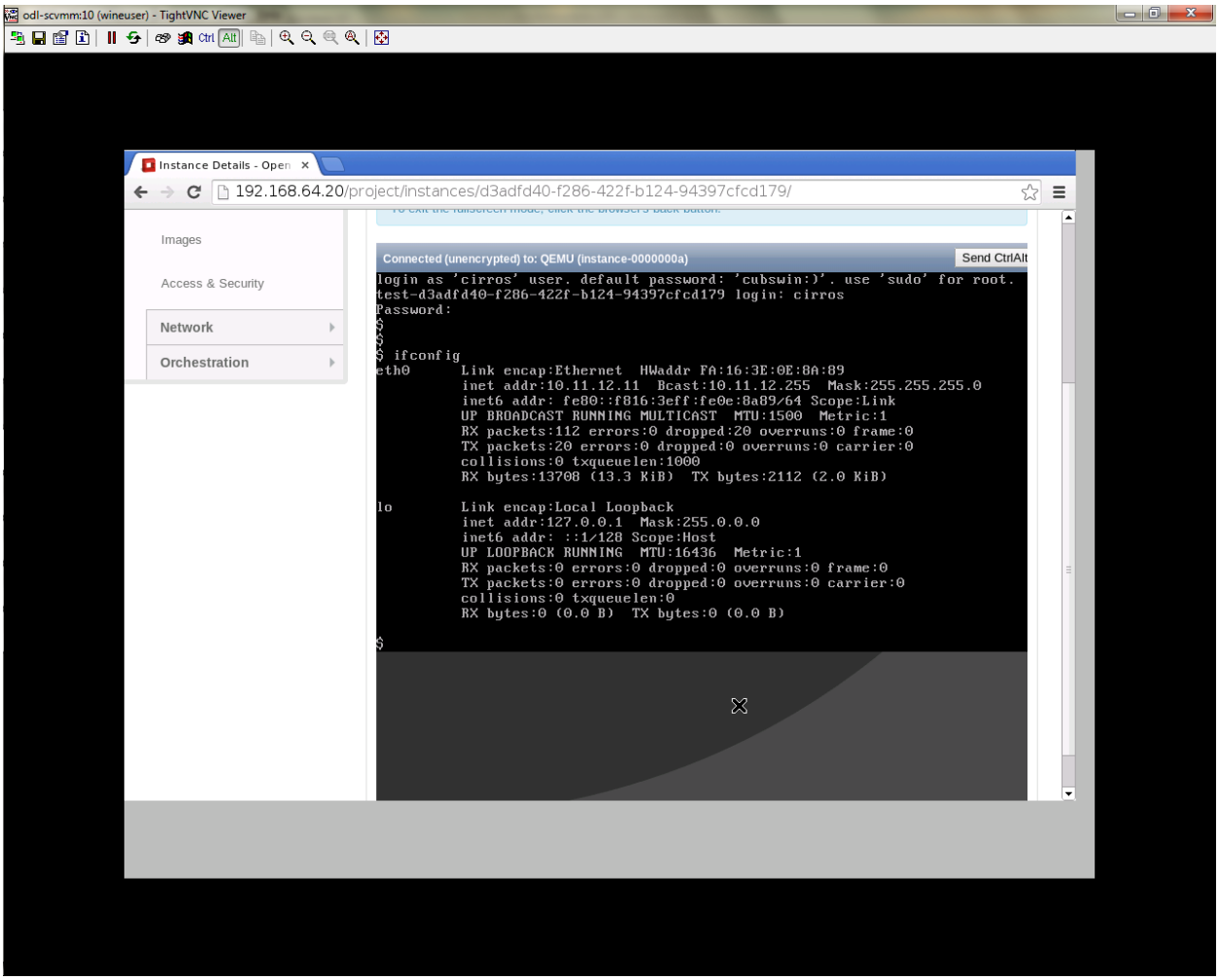


Fig. 1.130: Instance Console

- Login to the VM console and verify with a ping command.

Verification of Control and Compute Node after VM creation

- Every time a new VM is created, more interfaces are added to the br-int bridge in Open vSwitch.
- Use *sudo ovs-vsctl show* to list the number of interfaces added.
- Please visit the DLUX GUI to list the new nodes in every switch.

Getting started with DLUX

Ensure that you have created a topology and enabled MD-SAL feature in the Karaf distribution before you use DLUX for network management.

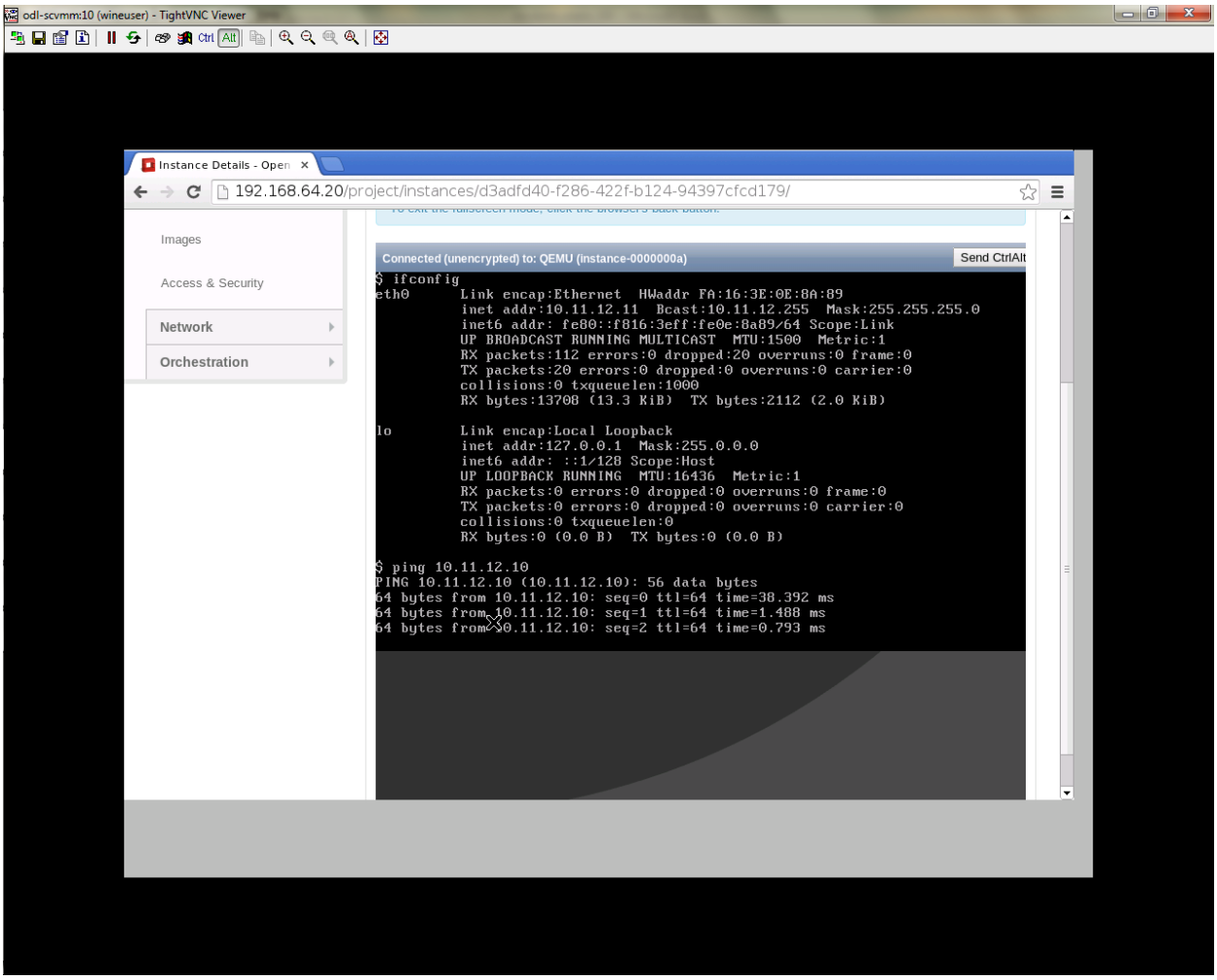


Fig. 1.131: Ping

Logging In

To log in to DLUX, after installing the application: * Open a browser and enter the login URL. If you have installed DLUX as a stand-alone, then the login URL is http://localhost:9000/DLUX/index.html. However if you have deployed DLUX with Karaf, then the login URL is http://<your IP>:8181/dlux/index.html. * Login to the application with user ID and password credentials as admin. NOTE:admin is the only user type available for DLUX in this release.

Working with DLUX

To get a complete DLUX feature list, install restconf, odl l2 switch, and switch while you start the DLUX distribution.

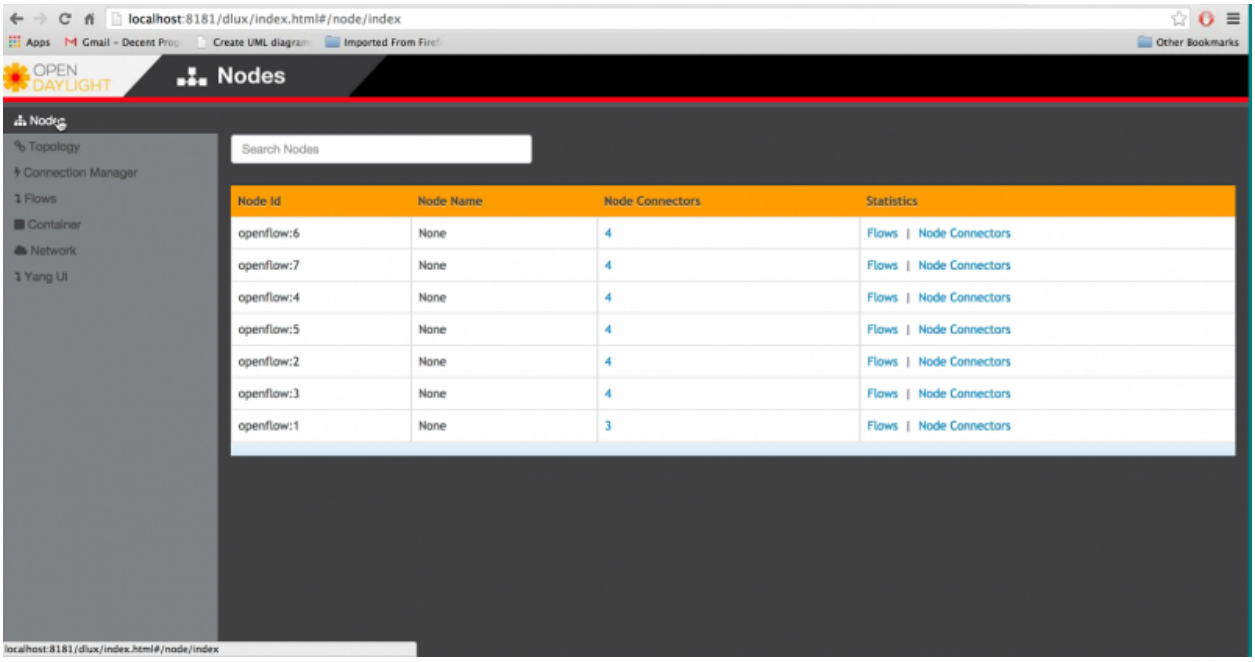


Fig. 1.132: DLUX_GUI

Note: DLUX enables only those modules, whose APIs are responding. If you enable just the MD-SAL in beginning and then start dlux, only MD-SAL related tabs will be visible. While using the GUI if you enable AD-SAL karaf features, those tabs will appear automatically.

Viewing Network Statistics

The Nodes module on the left pane enables you to view the network statistics and port information for the switches in the network. * To use the Nodes module: ** Select Nodeson the left pane.

```
The right pane displays atable that lists all the nodes, node connectors and the␣
↪statistics.
```

- Enter a node ID in the Search Nodes tab to search by node connectors.

- Click on the Node Connector number to view details such as port ID, port name, number of ports per switch, MAC Address, and so on.
- Click Flows in the Statistics column to view Flow Table Statistics for the particular node like table ID, packet match, active flows and so on.
- Click Node Connectors to view Node Connector Statistics for the particular node ID.

Viewing Network Topology

To view network topology: * Select Topology on the left pane. You will view the graphical representation on the right pane.

```
In the diagram
blue boxes represent the switches,black represents the hosts available, and lines␣
↪represents how switches are connected.
```

Note: DLUX UI does not provide ability to add topology information. The Topology should be created using an open flow plugin. Controller stores this information in the database and displays on the DLUX page, when the you connect to the controller using OpenFlow.

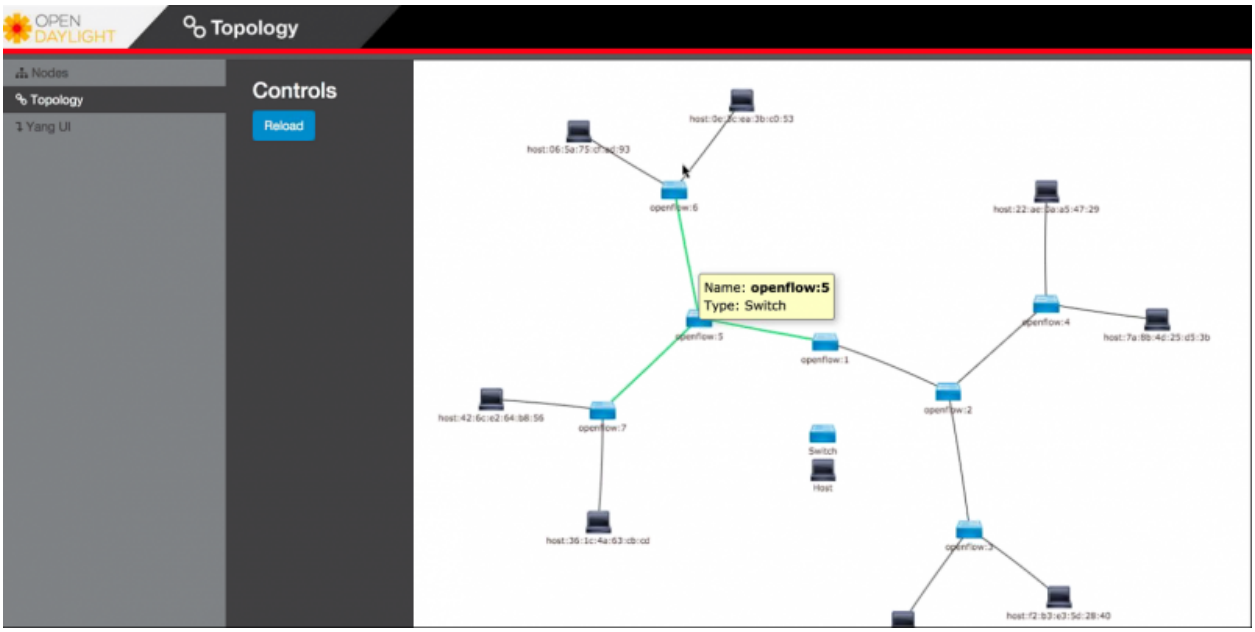


Fig. 1.133: Topology

OpenStack PackStack Installation Steps

- Please go through the below wiki page for OpenStack PackStack installation steps.
  - https://wiki.opendaylight.org/view/Release/Lithium/VTN/User_Guide/Openstack_Packstack_Support

References

- http://devstack.org/guides/multinode-lab.html
- https://wiki.opendaylight.org/view/File:Vtn_demo_hackfest_2014_march.pdf

VTN Manager Usage Examples

How to provision virtual L2 Network

Overview

This page explains how to provision virtual L2 network using VTN Manager. This page targets Boron release, so the procedure described here does not work in other releases.

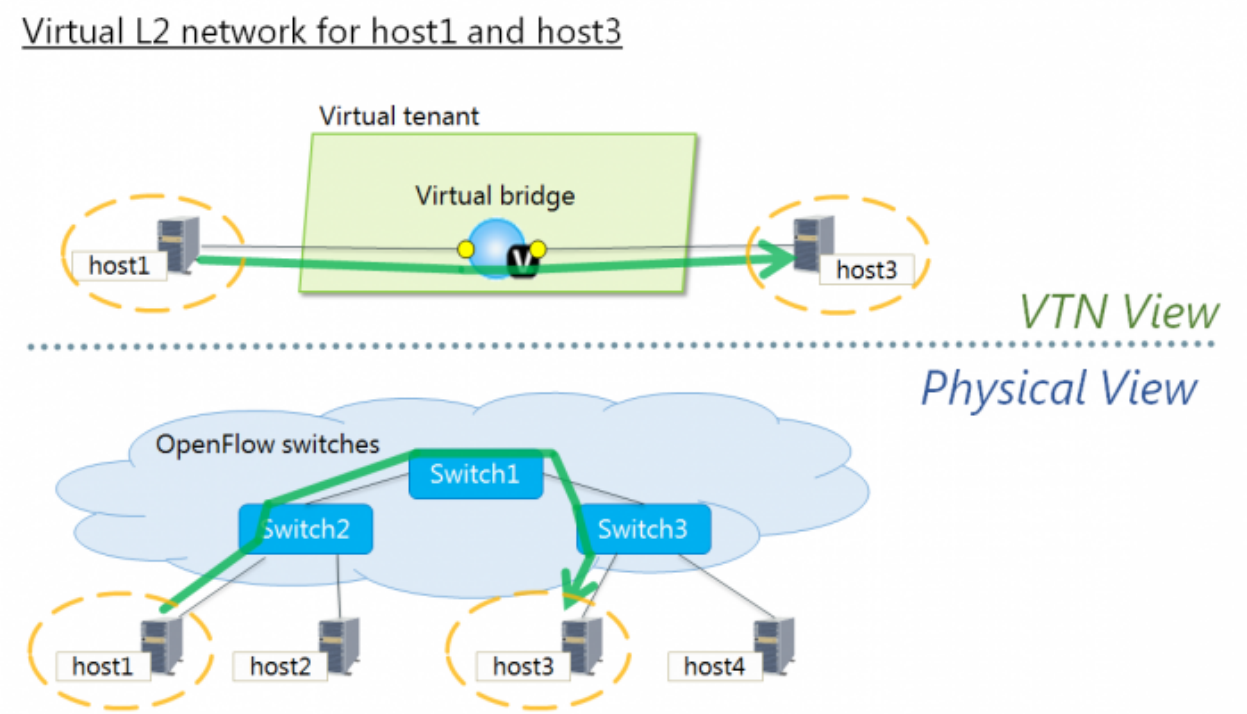


Fig. 1.134: Virtual L2 network for host1 and host3

Requirements

Mininet

- To provision OpenFlow switches, this page uses Mininet. Mininet details and set-up can be referred at the following page: https://wiki.opendaylight.org/view/OpenDaylight_Controller:Installation#Using_Mininet
- Start Mininet and create three switches(s1, s2, and s3) and four hosts(h1, h2, h3, and h4) in it.

```
mininet@mininet-vm:~$ sudo mn --controller=remote,ip=192.168.0.100 --topo tree,2
```

Note: Replace "192.168.0.100" with the IP address of OpenDaylight controller based on your environment.

- you can check the topology that you have created by executing "net" command in the Mininet console.

```
mininet> net
h1 h1-eth0:s2-eth1
h2 h2-eth0:s2-eth2
h3 h3-eth0:s3-eth1
h4 h4-eth0:s3-eth2
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s3-eth3
s2 lo:  s2-eth1:h1-eth0 s2-eth2:h2-eth0 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h3-eth0 s3-eth2:h4-eth0 s3-eth3:s1-eth2
```

- In this guide, you will provision the virtual L2 network to establish communication between h1 and h3.

Configuration

To provision the virtual L2 network for the two hosts (h1 and h3), execute REST API provided by VTN Manager as follows. It uses curl command to call the REST API.

- Create a virtual tenant named vtn1 by executing the update-vtn RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn:update-vtn -d '{"input":{"tenant-name":"vtn1
↪"}}'
```

- Create a virtual bridge named vbr1 in the tenant vtn1 by executing the update-vbridge RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vbridge:update-vbridge -d '{"input":{"tenant-
↪name":"vtn1", "bridge-name":"vbr1"}}'
```

- Create two interfaces into the virtual bridge by executing the update-vinterface RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vinterface:update-vinterface -d '{"input":{
↪"tenant-name":"vtn1", "bridge-name":"vbr1", "interface-name":"if1"}}'
```

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vinterface:update-vinterface -d '{"input":{
↪"tenant-name":"vtn1", "bridge-name":"vbr1", "interface-name":"if2"}}'
```

- Configure two mappings on the created interfaces by executing the set-port-map RPC.
  - The interface if1 of the virtual bridge will be mapped to the port "s2-eth1" of the switch "openflow:2" of the Mininet.
    - The h1 is connected to the port "s2-eth1".
  - The interface if2 of the virtual bridge will be mapped to the port "s3-eth1" of the switch "openflow:3" of the Mininet.
    - The h3 is connected to the port "s3-eth1".

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-port-map:set-port-map -d '{"input":{"tenant-
↪name":"vtn1", "bridge-name":"vbr1", "interface-name":"if1", "node":"openflow:2",
↪"port-name":"s2-eth1"}}'
```

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-port-map:set-port-map -d '{"input":{"tenant-
↪name":"vtn1", "bridge-name":"vbr1", "interface-name":"if2", "node":"openflow:3",
↪"port-name":"s3-eth1"}}'
```

Verification

- Please execute ping from h1 to h3 to verify if the virtual L2 network for h1 and h3 is provisioned successfully.

```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=243 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.341 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.078 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.079 ms
```

- You can also verify the configuration by executing the following REST API. It shows all configuration in VTN Manager.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X GET http://
↪localhost:8181/restconf/operational/vtn:vtns/
```

- The result of the command should be like this.

```
{
  "vtns": {
    "vtn": [
    {
      "name": "vtn1",
        "vtenant-config": {
          "idle-timeout": 300,
          "hard-timeout": 0
        },
        "vbridge": [
        {
          "name": "vbr1",
          "bridge-status": {
            "state": "UP",
            "path-faults": 0
          },
          "vbridge-config": {
            "age-interval": 600
          },
          "vinterface": [
          {
            "name": "if2",
            "vinterface-status": {
              "entity-state": "UP",
              "state": "UP",
              "mapped-port": "openflow:3:3"
            },
```

```
            "vinterface-config": {
              "enabled": true
            },
            "port-map-config": {
              "vlan-id": 0,
              "port-name": "s3-eth1",
              "node": "openflow:3"
            }
          },
          {
            "name": "if1",
            "vinterface-status": {
              "entity-state": "UP",
              "state": "UP",
              "mapped-port": "openflow:2:1"
            },
            "vinterface-config": {
              "enabled": true
            },
            "port-map-config": {
              "vlan-id": 0,
              "port-name": "s2-eth1",
              "node": "openflow:2"
            }
          }
          ]
        }
      ]
    }
    ]
  }
}
```

Cleaning Up

- You can delete the virtual tenant vtn1 by executing the remove-vtn RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn:remove-vtn -d '{"input":{"tenant-name":"vtn1
↪"}}'
```

How To Test Vlan-Map In Mininet Environment

Overview

This page explains how to test Vlan-map in a multi host scenario using mininet. This page targets Boron release, so the procedure described here does not work in other releases.

Requirements

Save the mininet script given below as vlan_vtn_test.py and run the mininet script in the mininet environment where Mininet is installed.

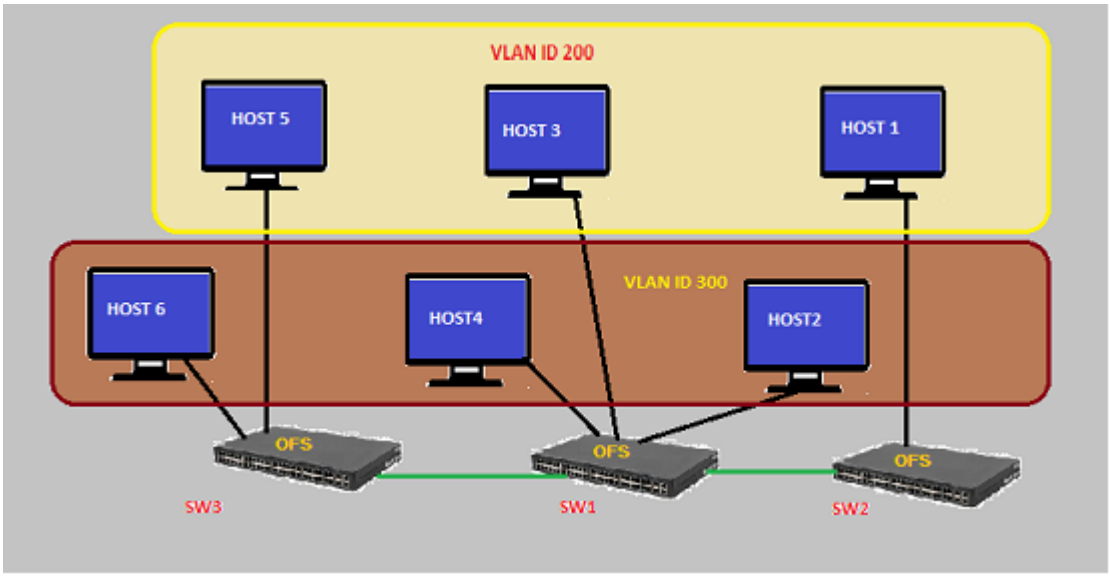


Fig. 1.135: Example that demonstrates vlanmap testing in Mininet Environment

Mininet Script

https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_(VTN):Scripts:Mininet#Network_with_hosts_in_different_vlan

- Run the mininet script

```
sudo mn --controller=remote,ip=192.168.64.13 --custom vlan_vtn_test.py --topo mytopo
```

Note: Replace "192.168.64.13" with the IP address of OpenDaylight controller based on your environment.

- You can check the topology that you have created by executing "net" command in the Mininet console.

```
mininet> net
h1 h1-eth0.200:s1-eth1
h2 h2-eth0.300:s2-eth2
h3 h3-eth0.200:s2-eth3
h4 h4-eth0.300:s2-eth4
h5 h5-eth0.200:s3-eth2
h6 h6-eth0.300:s3-eth3
s1 lo:  s1-eth1:h1-eth0.200 s1-eth2:s2-eth1 s1-eth3:s3-eth1
s2 lo:  s2-eth1:s1-eth2 s2-eth2:h2-eth0.300 s2-eth3:h3-eth0.200 s2-eth4:h4-eth0.300
s3 lo:  s3-eth1:s1-eth3 s3-eth2:h5-eth0.200 s3-eth3:h6-eth0.300
c0
```

Configuration

To test vlan-map, execute REST API provided by VTN Manager as follows.

- Create a virtual tenant named vtn1 by executing the update-vtn RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn:update-vtn -d '{"input":{"tenant-name":"vtn1
↪"}}'
```

- Create a virtual bridge named vbr1 in the tenant vtn1 by executing the update-vbridge RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vbridge:update-vbridge -d '{"input":{"tenant-
↪name":"vtn1","bridge-name":"vbr1"}}'
```

- Configure a vlan map with vlanid 200 for vBridge vbr1 by executing the add-vlan-map RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vlan-map:add-vlan-map -d '{"input":{"vlan-id
↪":200,"tenant-name":"vtn1","bridge-name":"vbr1"}}'
```

- Create a virtual bridge named vbr2 in the tenant vtn1 by executing the update-vbridge RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vbridge:update-vbridge -d '{"input":{"tenant-
↪name":"vtn1","bridge-name":"vbr2"}}'
```

- Configure a vlan map with vlanid 300 for vBridge vbr2 by executing the add-vlan-map RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vlan-map:add-vlan-map -d '{"input":{"vlan-id
↪":300,"tenant-name":"vtn1","bridge-name":"vbr2"}}'
```

Verification

- Please execute pingall in mininet environment to view the host reachability.

```
mininet> pingall
Ping: testing ping reachability
h1 -> X h3 X h5 X
h2 -> X X h4 X h6
h3 -> h1 X X h5 X
h4 -> X h2 X X h6
h5 -> h1 X h3 X X
h6 -> X h2 X h4 X
```

- You can also verify the configuration by executing the following REST API. It shows all configurations in VTN Manager.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X GET http://
↪localhost:8181/restconf/operational/vtn:vtns
```

- The result of the command should be like this.

```
{
  "vtns": {
    "vtn": [
    {
      "name": "vtn1",
        "vtenant-config": {
          "hard-timeout": 0,
```

```
          "idle-timeout": 300,
          "description": "creating vtn"
        },
        "vbridge": [
        {
          "name": "vbr2",
          "vbridge-config": {
            "age-interval": 600,
            "description": "creating vbr2"
          },
          "bridge-status": {
            "state": "UP",
            "path-faults": 0
          },
          "vlan-map": [
          {
            "map-id": "ANY.300",
            "vlan-map-config": {
              "vlan-id": 300
            },
            "vlan-map-status": {
              "active": true
            }
          }
          ]
        },
        {
          "name": "vbr1",
          "vbridge-config": {
            "age-interval": 600,
            "description": "creating vbr1"
          },
          "bridge-status": {
            "state": "UP",
            "path-faults": 0
          },
          "vlan-map": [
          {
            "map-id": "ANY.200",
            "vlan-map-config": {
              "vlan-id": 200
            },
            "vlan-map-status": {
              "active": true
            }
          }
          ]
        }
      ]
    }
    ]
  }
}
```

Cleaning Up

- You can delete the virtual tenant vtn1 by executing the remove-vtn RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn:remove-vtn -d '{"input":{"tenant-name":"vtn1
↪"}}'
```

How To Configure Service Function Chaining using VTN Manager

Overview

This page explains how to configure VTN Manager for Service Chaining. This page targets Boron release, so the procedure described here does not work in other releases.

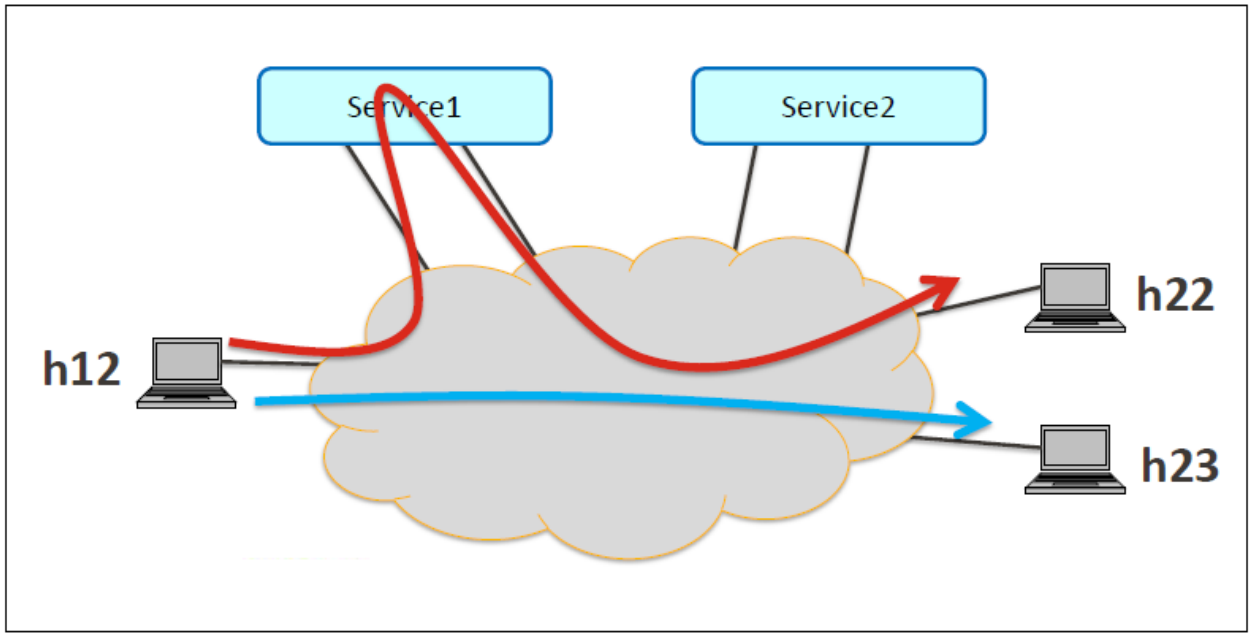


Fig. 1.136: Service Chaining With One Service

Requirements

- Please refer to the Installation Pages to run ODL with VTN Feature enabled.
- Please ensure Bridge-Utils package is installed in mininet environment before running the mininet script.
- To install Bridge-Utils package run sudo apt-get install bridge-utils (assuming Ubuntu is used to run mininet, If not then this is not required).
- Save the mininet script given below as topo_handson.py and run the mininet script in the mininet environment where Mininet is installed.

Mininet Script

- Script for emulating network with multiple hosts.
- Before executing the mininet script, please confirm Controller is up and running.
- Run the mininet script.
- Replace <path> and <Controller IP> based on your environment

```
sudo mn --controller=remote,ip=<Controller IP> --custom <path>\topo_handson.py --topo␣
↪mytopo2
```

```
mininet> net
h11 h11-eth0:s1-eth1
h12 h12-eth0:s1-eth2
h21 h21-eth0:s2-eth1
h22 h22-eth0:s2-eth2
h23 h23-eth0:s2-eth3
srvc1 srvc1-eth0:s3-eth3 srvc1-eth1:s4-eth3
srvc2 srvc2-eth0:s3-eth4 srvc2-eth1:s4-eth4
s1 lo:  s1-eth1:h11-eth0 s1-eth2:h12-eth0 s1-eth3:s2-eth4 s1-eth4:s3-eth2
s2 lo:  s2-eth1:h21-eth0 s2-eth2:h22-eth0 s2-eth3:h23-eth0 s2-eth4:s1-eth3 s2-eth5:s4-
↪eth1
s3 lo:  s3-eth1:s4-eth2 s3-eth2:s1-eth4 s3-eth3:srvc1-eth0 s3-eth4:srvc2-eth0
s4 lo:  s4-eth1:s2-eth5 s4-eth2:s3-eth1 s4-eth3:srvc1-eth1 s4-eth4:srvc2-eth1
```

Configurations

Mininet

- Please follow the below steps to configure the network in mininet as in the below image:

Configure service nodes

- Please execute the following commands in the mininet console where mininet script is executed.

```
mininet> srvc1 ip addr del 10.0.0.6/8 dev srvc1-eth0
mininet> srvc1 brctl addbr br0
mininet> srvc1 brctl addif br0 srvc1-eth0
mininet> srvc1 brctl addif br0 srvc1-eth1
mininet> srvc1 ifconfig br0 up
mininet> srvc1 tc qdisc add dev srvc1-eth1 root netem delay 200ms
mininet> srvc2 ip addr del 10.0.0.7/8 dev srvc2-eth0
mininet> srvc2 brctl addbr br0
mininet> srvc2 brctl addif br0 srvc2-eth0
mininet> srvc2 brctl addif br0 srvc2-eth1
mininet> srvc2 ifconfig br0 up
mininet> srvc2 tc qdisc add dev srvc2-eth1 root netem delay 300ms
```

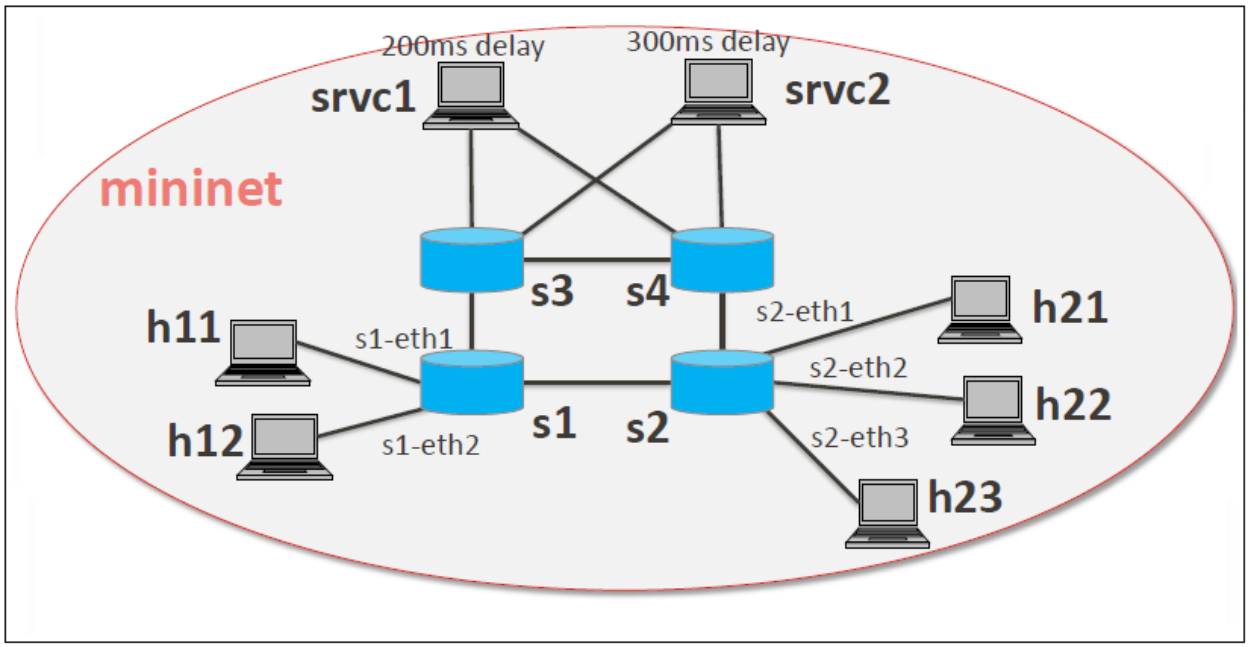


Fig. 1.137: Mininet Configuration

Controller

Multi-Tenancy

- Please execute the below commands to configure the network topology in the controller as in the below image:

Please execute the below commands in controller

Note: The below commands are for the difference in behavior of Manager in Boron topology. The Link below has the details for this bug: https://bugs.opendaylight.org/show_bug.cgi?id=3818.

```
curl --user admin:admin -H 'content-type: application/json' -H 'ipaddr:127.0.0.1' -X␣
↪PUT http://localhost:8181/restconf/config/vtn-static-topology:vtn-static-topology/
↪static-edge-ports -d '{"static-edge-ports": {"static-edge-port": [ {"port":
↪"openflow:3:3"}, {"port": "openflow:3:4"}, {"port": "openflow:4:3"}, {"port":
↪"openflow:4:4"}]}}'
```

- Create a virtual tenant named vtn1 by executing the update-vtn RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn:update-vtn -d '{"input":{"tenant-name":"vtn1
↪","update-mode":"CREATE","operation":"SET","description":"creating vtn","idle-
↪timeout":300,"hard-timeout":0}}'
```

- Create a virtual bridge named vbr1 in the tenant vtn1 by executing the update-vbridge RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vbridge:update-vbridge -d '{"input":{"update-
↪mode":"CREATE","operation":"SET","description":"creating vbr","tenant-name":"vtn1",
↪"bridge-name":"vbr1"}}'
```

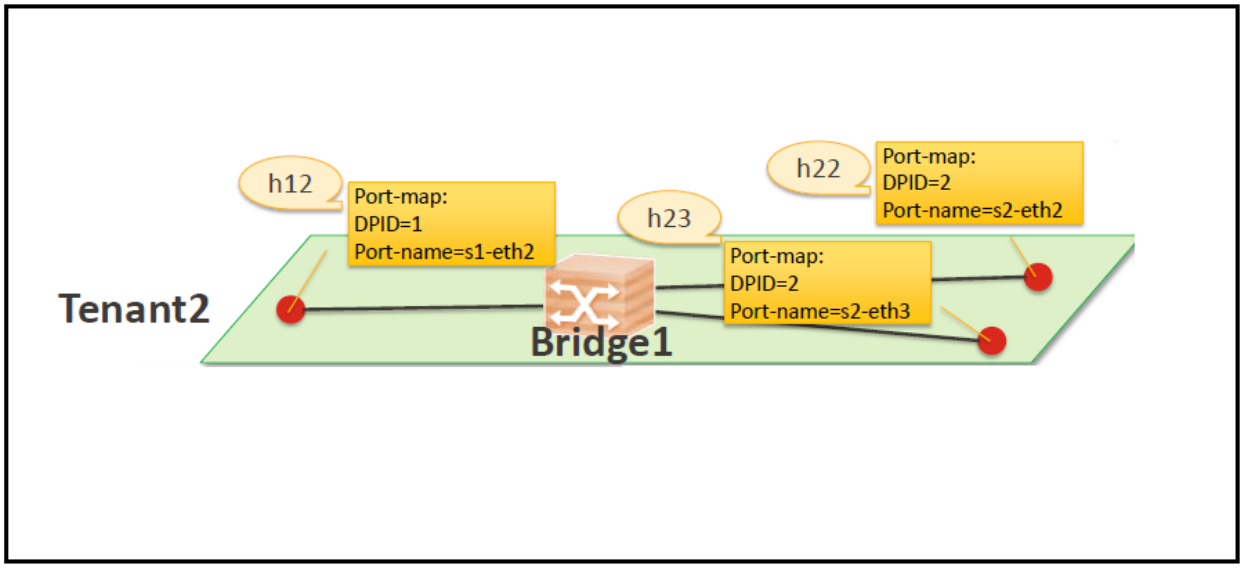


Fig. 1.138: Tenant2

- Create interface if1 into the virtual bridge vbr1 by executing the update-vinterface RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vinterface:update-vinterface -d '{"input":{
↪"update-mode":"CREATE","operation":"SET","description":"Creating vbrif1 interface",
↪"tenant-name":"vtn1","bridge-name":"vbr1","interface-name":"if1"}}'
```

- Configure port mapping on the interface by executing the set-port-map RPC.
  - The interface if1 of the virtual bridge will be mapped to the port "s1-eth2" of the switch "openflow:1" of the Mininet.
    - The h12 is connected to the port "s1-eth2".

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-port-map:set-port-map -d '{"input":{"vlan-id
↪":0,"tenant-name":"vtn1","bridge-name":"vbr1","interface-name":"if1","node":
↪"openflow:1","port-name":"s1-eth2"}}'
```

- Create interface if2 into the virtual bridge vbr1 by executing the update-vinterface RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vinterface:update-vinterface -d '{"input":{
↪"update-mode":"CREATE","operation":"SET","description":"Creating vbrif2 interface",
↪"tenant-name":"vtn1","bridge-name":"vbr1","interface-name":"if2"}}'
```

- Configure port mapping on the interface by executing the set-port-map RPC.
  - The interface if2 of the virtual bridge will be mapped to the port "s2-eth2" of the switch "openflow:2" of the Mininet.
    - The h22 is connected to the port "s2-eth2".

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-port-map:set-port-map -d '{"input":{"vlan-id
↪":0,"tenant-name":"vtn1","bridge-name":"vbr1","interface-name":"if2","node":
↪"openflow:2","port-name":"s2-eth2"}}'
```

- Create interface if3 into the virtual bridge vbr1 by executing the update-vinterface RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vinterface:update-vinterface -d '{"input":{
↪"update-mode":"CREATE","operation":"SET","description":"Creating vbrif3 interface",
↪"tenant-name":"vtn1","bridge-name":"vbr1","interface-name":"if3"}}'
```

- Configure port mapping on the interfaces by executing the set-port-map RPC.
  - The interface if3 of the virtual bridge will be mapped to the port "s2-eth3" of the switch "openflow:2" of the Mininet.
    - The h23 is connected to the port "s2-eth3".

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-port-map:set-port-map -d '{"input":{"vlan-id
↪":0,"tenant-name":"vtn1","bridge-name":"vbr1","interface-name":"if3","node":
↪"openflow:2","port-name":"s2-eth3"}}'
```

Traffic filtering

- Create flowcondition named cond_1 by executing the set-flow-condition RPC.
  - For option source and destination-network, get inet address of host h12(src) and h22(dst) from mininet.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-condition:set-flow-condition -d '{"input
↪":{"operation":"SET","present":"false","name":"cond_1","vtn-flow-match":[{"index":1,
↪"vtn-ether-match":{},"vtn-inet-match":{"source-network":"10.0.0.2/32","destination-
↪network":"10.0.0.4/32"}}]}}'
```

- Flow filter demonstration with DROP action-type. Create Flowfilter in VBR Interface if1 by executing the set-flow-filter RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-filter:set-flow-filter -d '{"input":{
↪"output":"false","tenant-name":"vtn1","bridge-name":"vbr1","interface-name":"if1",
↪"vtn-flow-filter":[{"condition":"cond_1","index":10,"vtn-drop-filter":{}}]}}'
```

Service Chaining

With One Service

- Please execute the below commands to configure the network topology which sends some specific traffic via a single service(External device) in the controller as in the below image:
- Create a virtual terminal named vt_srvc1_1 in the tenant vtn1 by executing the update-vterminal RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vterminal:update-vterminal -d '{"input":{
↪"update-mode":"CREATE","operation":"SET","tenant-name":"vtn1","terminal-name":"vt_
↪srvc1_1","description":"Creating vterminal"}}'
```

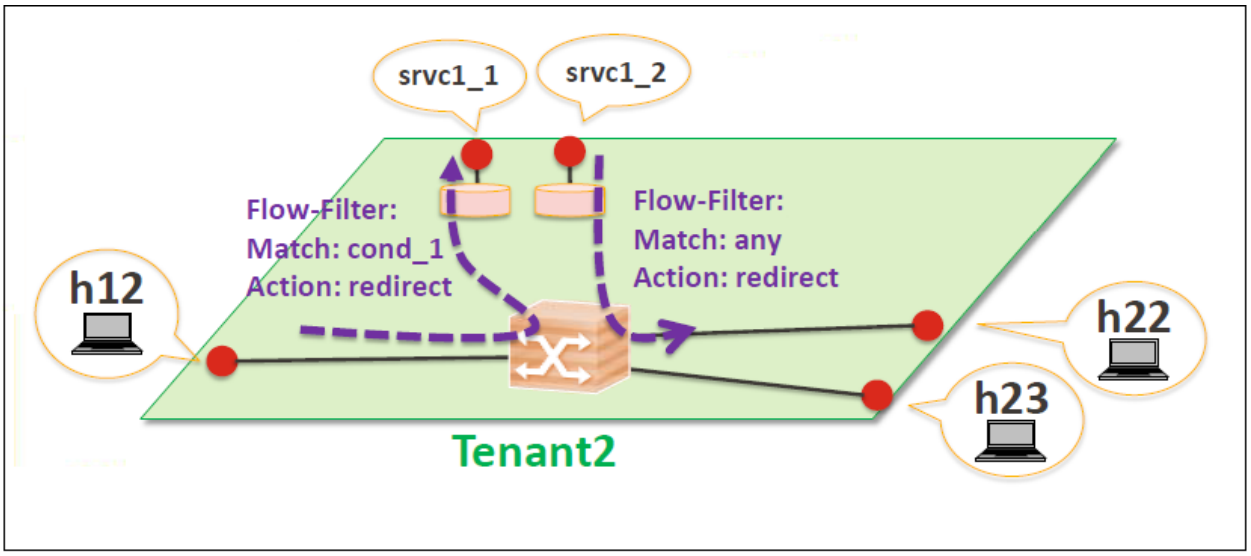


Fig. 1.139: Service Chaining With One Service LLD

- Create interface IF into the virtual terminal vt_srvc1_1 by executing the update-vinterface RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vinterface:update-vinterface -d '{"input":{
↪"update-mode":"CREATE","operation":"SET","description":"Creating vterminal IF",
↪"enabled":"true","tenant-name":"vtn1","terminal-name":"vt_srvc1_1","interface-name":
↪"IF"}}'
```

- Configure port mapping on the interfaces by executing the set-port-map RPC.
  - The interface IF of the virtual terminal will be mapped to the port "s3-eth3" of the switch "openflow:3" of the Mininet.
    - The h12 is connected to the port "s3-eth3".

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-port-map:set-port-map -d '{"input":{"tenant-
↪name":"vtn1","terminal-name":"vt_srvc1_1","interface-name":"IF","node":"openflow:3",
↪"port-name":"s3-eth3"}}'
```

- Create a virtual terminal named vt_srvc1_2 in the tenant vtn1 by executing the update-vterminal RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vterminal:update-vterminal -d '{"input":{
↪"update-mode":"CREATE","operation":"SET","tenant-name":"vtn1","terminal-name":"vt_
↪srvc1_2","description":"Creating vterminal"}}'
```

- Create interface IF into the virtual terminal vt_srvc1_2 by executing the update-vinterface RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vinterface:update-vinterface -d '{"input":{
↪"update-mode":"CREATE","operation":"SET","description":"Creating vterminal IF",
↪"enabled":"true","tenant-name":"vtn1","terminal-name":"vt_srvc1_2","interface-name":
↪"IF"}}'
```

- Configure port mapping on the interfaces by executing the set-port-map RPC.
  - The interface IF of the virtual terminal will be mapped to the port "s4-eth3" of the switch "openflow:4" of the Mininet.
    - The h22 is connected to the port "s4-eth3".

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-port-map:set-port-map -d '{"input":{"tenant-
↪name":"vtn1","terminal-name":"vt_srvc1_2","interface-name":"IF","node":"openflow:4",
↪"port-name":"s4-eth3"}}'
```

- Create flowcondition named cond_1 by executing the set-flow-condition RPC.
  - For option source and destination-network, get inet address of host h12(src) and h22(dst) from mininet.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-condition:set-flow-condition -d '{"input
↪":{"operation":"SET","present":"false","name":"cond_1","vtn-flow-match":[{"index":1,
↪"vtn-ether-match":{},"vtn-inet-match":{"source-network":"10.0.0.2/32","destination-
↪network":"10.0.0.4/32"}}]}}'
```

- Create flowcondition named cond_any by executing the set-flow-condition RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-condition:set-flow-condition -d '{"input
↪":{"operation":"SET","present":"false","name":"cond_any","vtn-flow-match":[{"index
↪":1}]}}'
```

- Flow filter demonstration with redirect action-type. Create Flowfilter in virtual terminal vt_srvc1_2 interface IF by executing the set-flow-filter RPC.
  - Flowfilter redirects vt_srvc1_2 to bridge1-IF2

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-filter:set-flow-filter -d '{"input":{
↪"output":"false","tenant-name":"vtn1","terminal-name":"vt_srvc1_2","interface-name":
↪"IF","vtn-flow-filter":[{"condition":"cond_any","index":10,"vtn-redirect-filter":{
↪"redirect-destination":{"bridge-name":"vbr1","interface-name":"if2"},"output":"true
↪"}}]}}'
```

- Flow filter demonstration with redirect action-type. Create Flowfilter in vbridge vbr1 interface if1 by executing the set-flow-filter RPC.
  - Flow filter redirects Bridge1-IF1 to vt_srvc1_1

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-filter:set-flow-filter -d '{"input":{
↪"output":"false","tenant-name":"vtn1","bridge-name":"vbr1","interface-name":"if1",
↪"vtn-flow-filter":[{"condition":"cond_1","index":10,"vtn-redirect-filter":{
↪"redirect-destination":{"terminal-name":"vt_srvc1_1","interface-name":"IF"},"output
↪":"true"}}]}}'
```

Verification

- Ping host12 to host22 to view the host rechability, a delay of 200ms will be taken to reach host22 as below.

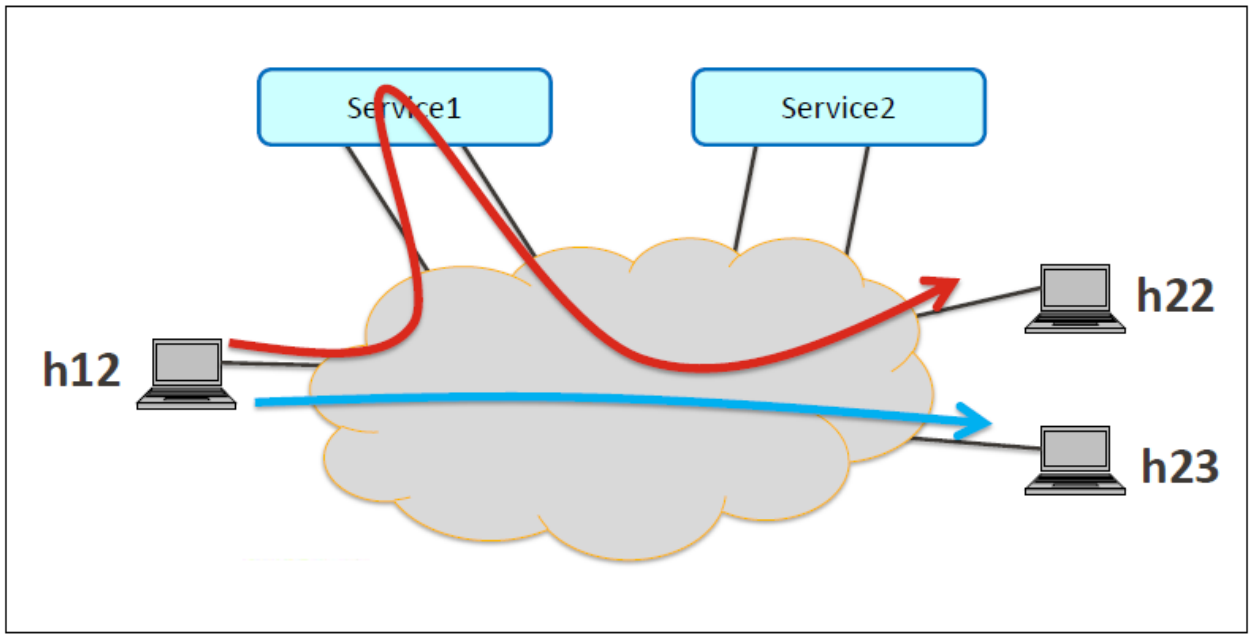


Fig. 1.140: Service Chaining With One Service

```
mininet> h12 ping h22
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=35 ttl=64 time=209 ms
64 bytes from 10.0.0.4: icmp_seq=36 ttl=64 time=201 ms
64 bytes from 10.0.0.4: icmp_seq=37 ttl=64 time=200 ms
64 bytes from 10.0.0.4: icmp_seq=38 ttl=64 time=200 ms
```

With two services

- Please execute the below commands to configure the network topology which sends some specific traffic via two services(External device) in the controller as in the below image.

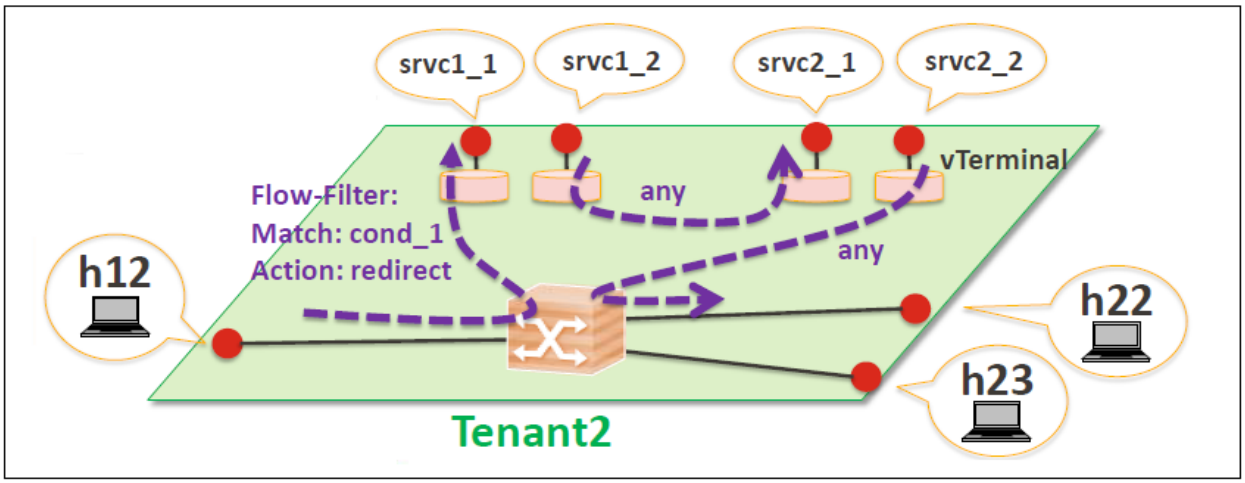


Fig. 1.141: Service Chaining With Two Services LLD

- Create a virtual terminal named vt_srvc2_1 in the tenant vtn1 by executing the update-vterminal RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vterminal:update-vterminal -d '{"input":{
↪"update-mode":"CREATE","operation":"SET","tenant-name":"vtn1","terminal-name":"vt_
↪srvc2_1","description":"Creating vterminal"}}'
```

- Create interface IF into the virtual terminal vt_srvc2_1 by executing the update-vinterface RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vinterface:update-vinterface -d '{"input":{
↪"update-mode":"CREATE","operation":"SET","description":"Creating vterminal IF",
↪"enabled":"true","tenant-name":"vtn1","terminal-name":"vt_srvc2_1","interface-name":
↪"IF"}}'
```

- Configure port mapping on the interfaces by executing the set-port-map RPC.
  - The interface IF of the virtual terminal will be mapped to the port "s3-eth4" of the switch "openflow:3" of the Mininet.
    - The host h12 is connected to the port "s3-eth4".

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-port-map:set-port-map -d '{"input":{"tenant-
↪name":"vtn1","terminal-name":"vt_srvc2_1","interface-name":"IF","node":"openflow:3",
↪"port-name":"s3-eth4"}}'
```

- Create a virtual terminal named vt_srvc2_2 in the tenant vtn1 by executing the update-vterminal RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vterminal:update-vterminal -d '{"input":{
↪"update-mode":"CREATE","operation":"SET","tenant-name":"vtn1","terminal-name":"vt_
↪srvc2_2","description":"Creating vterminal"}}'
```

- Create interfaces IF into the virtual terminal vt_srvc2_2 by executing the update-vinterface RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vinterface:update-vinterface -d '{"input":{
↪"update-mode":"CREATE","operation":"SET","description":"Creating vterminal IF",
↪"enabled":"true","tenant-name":"vtn1","terminal-name":"vt_srvc2_2","interface-name":
↪"IF"}}'
```

- Configure port mapping on the interfaces by executing the set-port-map RPC.
  - The interface IF of the virtual terminal will be mapped to the port "s4-eth4" of the switch "openflow:4" of the mininet.
    - The host h22 is connected to the port "s4-eth4".

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-port-map:set-port-map -d '{"input":{"tenant-
↪name":"vtn1","terminal-name":"vt_srvc2_2","interface-name":"IF","node":"openflow:4",
↪"port-name":"s4-eth4"}}'
```

- Flow filter demonstration with redirect action-type. Create Flowfilter in virtual terminal vt_srvc2_2 interface IF by executing the set-flow-filter RPC.
  - Flow filter redirects vt_srvc2_2 to Bridge1-IF2.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-filter:set-flow-filter -d '{"input":{
↪"output":"false","tenant-name":"vtn1","terminal-name":"vt_srvc2_2","interface-name":
↪"IF","vtn-flow-filter":[{"condition":"cond_any","index":10,"vtn-redirect-filter":{
↪"redirect-destination":{"bridge-name":"vbr1","interface-name":"if2"},"output":"true
↪"}}]}}'
```

- Flow filter demonstration with redirect action-type. Create Flowfilter in virtual terminal vt_srvc2_2 interface IF by executing the set-flow-filter RPC.
  - Flow filter redirects vt_srvc1_2 to vt_srvc2_1.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-filter:set-flow-filter -d '{"input":{
↪"output":"false","tenant-name":"vtn1","terminal-name":"vt_srvc1_2","interface-name":
↪"IF","vtn-flow-filter":[{"condition":"cond_any","index":10,"vtn-redirect-filter":{
↪"redirect-destination":{"terminal-name":"vt_srvc2_1","interface-name":"IF"},"output
↪":"true"}}]}}'
```

Verification

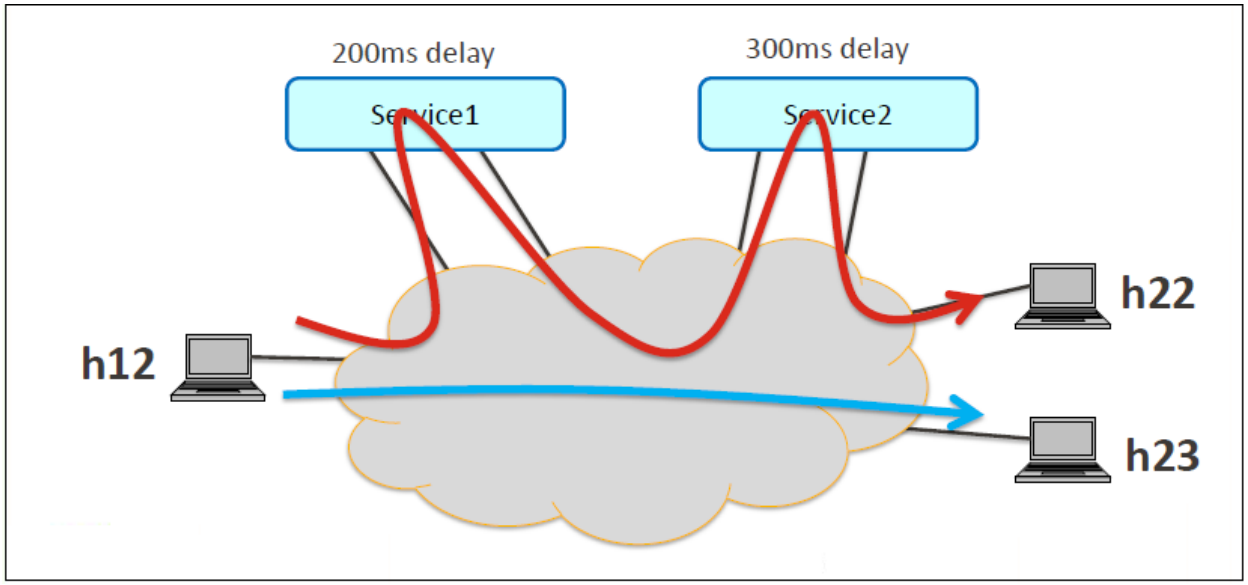


Fig. 1.142: Service Chaining With Two Service

- Ping host12 to host22 to view the host rechability, a delay of 500ms will be taken to reach host22 as below.

```
mininet> h12 ping h22
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=512 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=501 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=500 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=500 ms
```

- You can verify the configuration by executing the following REST API. It shows all configuration in VTN Manager.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X GET http://
↪localhost:8181/restconf/operational/vtn:vtns
```


```
{
  "vtn": [
  {
    "name": "vtn1",
      "vtenant-config": {
        "hard-timeout": 0,
        "idle-timeout": 300,
        "description": "creating vtn"
      },
      "vbridge": [
      {
        "name": "vbr1",
        "vbridge-config": {
          "age-interval": 600,
          "description": "creating vbr"
        },
        "bridge-status": {
          "state": "UP",
          "path-faults": 0
        },
        "vinterface": [
        {
          "name": "if1",
          "vinterface-status": {
            "mapped-port": "openflow:1:2",
            "state": "UP",
            "entity-state": "UP"
          },
          "port-map-config": {
            "vlan-id": 0,
            "node": "openflow:1",
            "port-name": "s1-eth2"
          },
          "vinterface-config": {
            "description": "Creating vbrif1 interface",
            "enabled": true
          },
          "vinterface-input-filter": {
            "vtn-flow-filter": [
            {
              "index": 10,
              "condition": "cond_1",
              "vtn-redirect-filter": {
                "output": true,
                "redirect-destination": {
                  "terminal-name": "vt_srvc1_1",
                  "interface-name": "IF"
                }
              }
            }
            ]
          }
        },
        {
          "name": "if2",
```

```
          "vinterface-status": {
            "mapped-port": "openflow:2:2",
            "state": "UP",
            "entity-state": "UP"
          },
          "port-map-config": {
            "vlan-id": 0,
            "node": "openflow:2",
            "port-name": "s2-eth2"
          },
          "vinterface-config": {
            "description": "Creating vbrif2 interface",
            "enabled": true
          }
        },
        {
          "name": "if3",
          "vinterface-status": {
            "mapped-port": "openflow:2:3",
            "state": "UP",
            "entity-state": "UP"
          },
          "port-map-config": {
            "vlan-id": 0,
            "node": "openflow:2",
            "port-name": "s2-eth3"
          },
          "vinterface-config": {
            "description": "Creating vbrif3 interface",
            "enabled": true
          }
        }
        ]
      }
    ],
      "vterminal": [
      {
        "name": "vt_srvc2_2",
        "bridge-status": {
          "state": "UP",
          "path-faults": 0
        },
        "vinterface": [
        {
          "name": "IF",
          "vinterface-status": {
            "mapped-port": "openflow:4:4",
            "state": "UP",
            "entity-state": "UP"
          },
          "port-map-config": {
            "vlan-id": 0,
            "node": "openflow:4",
            "port-name": "s4-eth4"
          },
          "vinterface-config": {
            "description": "Creating vterminal IF",
            "enabled": true
```

```
        },
        "vinterface-input-filter": {
          "vtn-flow-filter": [
          {
            "index": 10,
            "condition": "cond_any",
            "vtn-redirect-filter": {
              "output": true,
              "redirect-destination": {
                "bridge-name": "vbr1",
                "interface-name": "if2"
              }
            }
          }
          ]
        }
      }
      ],
        "vterminal-config": {
          "description": "Creating vterminal"
        }
    },
    {
      "name": "vt_srvc1_1",
      "bridge-status": {
        "state": "UP",
        "path-faults": 0
      },
      "vinterface": [
      {
        "name": "IF",
        "vinterface-status": {
          "mapped-port": "openflow:3:3",
          "state": "UP",
          "entity-state": "UP"
        },
        "port-map-config": {
          "vlan-id": 0,
          "node": "openflow:3",
          "port-name": "s3-eth3"
        },
        "vinterface-config": {
          "description": "Creating vterminal IF",
          "enabled": true
        }
      }
      ],
        "vterminal-config": {
          "description": "Creating vterminal"
        }
    },
    {
      "name": "vt_srvc1_2",
      "bridge-status": {
        "state": "UP",
        "path-faults": 0
      },
      "vinterface": [
```

```
        {
          "name": "IF",
          "vinterface-status": {
            "mapped-port": "openflow:4:3",
            "state": "UP",
            "entity-state": "UP"
          },
          "port-map-config": {
            "vlan-id": 0,
            "node": "openflow:4",
            "port-name": "s4-eth3"
          },
          "vinterface-config": {
            "description": "Creating vterminal IF",
            "enabled": true
          },
          "vinterface-input-filter": {
            "vtn-flow-filter": [
            {
              "index": 10,
              "condition": "cond_any",
              "vtn-redirect-filter": {
                "output": true,
                "redirect-destination": {
                  "terminal-name": "vt_srvc2_1",
                  "interface-name": "IF"
                }
              }
            }
            ]
          }
        }
        ],
          "vterminal-config": {
            "description": "Creating vterminal"
          }
      },
      {
        "name": "vt_srvc2_1",
        "bridge-status": {
          "state": "UP",
          "path-faults": 0
        },
        "vinterface": [
        {
          "name": "IF",
          "vinterface-status": {
            "mapped-port": "openflow:3:4",
            "state": "UP",
            "entity-state": "UP"
          },
          "port-map-config": {
            "vlan-id": 0,
            "node": "openflow:3",
            "port-name": "s3-eth4"
          },
          "vinterface-config": {
            "description": "Creating vterminal IF",
```

```
            "enabled": true
          }
        }
        ],
          "vterminal-config": {
            "description": "Creating vterminal"
          }
      }
    ]
  }
  ]
}
```

Cleaning Up

- To clean up both VTN and flowconditions.
- You can delete the virtual tenant vtn1 by executing the remove-vtn RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn:remove-vtn -d '{"input":{"tenant-name":"vtn1
↪"}}'
```

- You can delete the flowcondition cond_1 and cond_any by executing the remove-flow-condition RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-condition:remove-flow-condition -d '{
↪"input":{"name":"cond_1"}}'
```

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-condition:remove-flow-condition -d '{
↪"input":{"name":"cond_any"}}'
```

How To View Dataflows

Overview

This page explains how to view Dataflows using VTN Manager. This page targets Boron release, so the procedure described here does not work in other releases.

Dataflow feature enables retrieval and display of data flows in the OpenFlow network. The data flows can be retrieved based on an OpenFlow switch or a switch port or a L2 source host.

The flow information provided by this feature are

- Location of virtual node which maps the incoming packet and outgoing packets.
- Location of physical switch port where incoming and outgoing packets is sent and received.
- A sequence of physical route info which represents the packet route in the physical network.

Configuration

- To view Dataflow information, configure with VLAN Mapping https://wiki.opendaylight.org/view/VTN:Mananger:How_to_test_Vlan-map_using_mininet.

Verification

After creating vlan mapping configuration from the above page, execute as below in mininet to get switch details.

```
mininet> net
h1 h1-eth0.200:s1-eth1
h2 h2-eth0.300:s2-eth2
h3 h3-eth0.200:s2-eth3
h4 h4-eth0.300:s2-eth4
h5 h5-eth0.200:s3-eth2
h6 h6-eth0.300:s3-eth3
s1 lo:  s1-eth1:h1-eth0.200 s1-eth2:s2-eth1 s1-eth3:s3-eth1
s2 lo:  s2-eth1:s1-eth2 s2-eth2:h2-eth0.300 s2-eth3:h3-eth0.200 s2-eth4:h4-eth0.300
s3 lo:  s3-eth1:s1-eth3 s3-eth2:h5-eth0.200 s3-eth3:h6-eth0.300
c0
mininet>
```

Please execute ping from h1 to h3 to check hosts reachability.

```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=11.4 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.654 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.093 ms
```

Parallely execute below Restconf command to get data flow information of node "openflow:1" and its port "s1-eth1".

- Get the Dataflows information by executing the get-data-flow RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow:get-data-flow -d '{"input":{"tenant-name
↪":"vtn1","mode":"DETAIL","node":"openflow:1","data-flow-port":{"port-id":"1","port-
↪name":"s1-eth1"}}}'
```

```
{
  "output": {
    "data-flow-info": [
    {
      "averaged-data-flow-stats": {
        "packet-count": 1.1998800119988002,
          "start-time": 1455241209151,
          "end-time": 1455241219152,
          "byte-count": 117.58824117588242
      },
        "physical-route": [
        {
          "physical-ingress-port": {
            "port-name": "s2-eth3",
            "port-id": "3"
          },
          "physical-egress-port": {
```

```
            "port-name": "s2-eth1",
            "port-id": "1"
          },
          "node": "openflow:2",
          "order": 0
        },
        {
          "physical-ingress-port": {
            "port-name": "s1-eth2",
            "port-id": "2"
          },
          "physical-egress-port": {
            "port-name": "s1-eth1",
            "port-id": "1"
          },
          "node": "openflow:1",
          "order": 1
        }
      ],
        "data-egress-node": {
          "bridge-name": "vbr1",
          "tenant-name": "vtn1"
        },
        "hard-timeout": 0,
        "idle-timeout": 300,
        "data-flow-stats": {
          "duration": {
            "nanosecond": 640000000,
            "second": 362
          },
          "packet-count": 134,
          "byte-count": 12932
        },
        "data-egress-port": {
          "node": "openflow:1",
          "port-name": "s1-eth1",
          "port-id": "1"
        },
        "data-ingress-node": {
          "bridge-name": "vbr1",
          "tenant-name": "vtn1"
        },
        "data-ingress-port": {
          "node": "openflow:2",
          "port-name": "s2-eth3",
          "port-id": "3"
        },
        "creation-time": 1455240855753,
        "data-flow-match": {
          "vtn-ether-match": {
            "vlan-id": 200,
            "source-address": "6a:ff:e2:81:86:bb",
            "destination-address": "26:9f:82:70:ec:66"
          }
        },
        "virtual-route": [
        {
          "reason": "VLANMAPPED",
```

```
          "virtual-node-path": {
            "bridge-name": "vbr1",
            "tenant-name": "vtn1"
          },
          "order": 0
        },
        {
          "reason": "FORWARDED",
          "virtual-node-path": {
            "bridge-name": "vbr1",
            "tenant-name": "vtn1"
          },
          "order": 1
        }
      ],
        "flow-id": 16
    },
    {
      "averaged-data-flow-stats": {
        "packet-count": 1.1998800119988002,
        "start-time": 1455241209151,
        "end-time": 1455241219152,
        "byte-count": 117.58824117588242
      },
      "physical-route": [
      {
        "physical-ingress-port": {
          "port-name": "s1-eth1",
          "port-id": "1"
        },
        "physical-egress-port": {
          "port-name": "s1-eth2",
          "port-id": "2"
        },
        "node": "openflow:1",
        "order": 0
      },
      {
        "physical-ingress-port": {
          "port-name": "s2-eth1",
          "port-id": "1"
        },
        "physical-egress-port": {
          "port-name": "s2-eth3",
          "port-id": "3"
        },
        "node": "openflow:2",
        "order": 1
      }
      ],
        "data-egress-node": {
          "bridge-name": "vbr1",
          "tenant-name": "vtn1"
        },
        "hard-timeout": 0,
        "idle-timeout": 300,
        "data-flow-stats": {
          "duration": {
```

```
            "nanosecond": 587000000,
            "second": 362
          },
          "packet-count": 134,
          "byte-count": 12932
        },
        "data-egress-port": {
          "node": "openflow:2",
          "port-name": "s2-eth3",
          "port-id": "3"
        },
        "data-ingress-node": {
          "bridge-name": "vbr1",
          "tenant-name": "vtn1"
        },
        "data-ingress-port": {
          "node": "openflow:1",
          "port-name": "s1-eth1",
          "port-id": "1"
        },
        "creation-time": 1455240855747,
        "data-flow-match": {
          "vtn-ether-match": {
            "vlan-id": 200,
            "source-address": "26:9f:82:70:ec:66",
            "destination-address": "6a:ff:e2:81:86:bb"
          }
        },
        "virtual-route": [
        {
          "reason": "VLANMAPPED",
          "virtual-node-path": {
            "bridge-name": "vbr1",
            "tenant-name": "vtn1"
          },
          "order": 0
        },
        {
          "reason": "FORWARDED",
          "virtual-node-path": {
            "bridge-name": "vbr1",
            "tenant-name": "vtn1"
          },
          "order": 1
        }
      ],
        "flow-id": 15
    }
    ]
  }
}
```

How To Create Mac Map In VTN

Overview

- This page demonstrates Mac Mapping. This demonstration aims at enabling communication between two hosts and denying communication of particular host by associating a Vbridge to the hosts and configuring Mac Mapping (mac address) to the Vbridge.
- This page targets Boron release, so the procedure described here does not work in other releases.

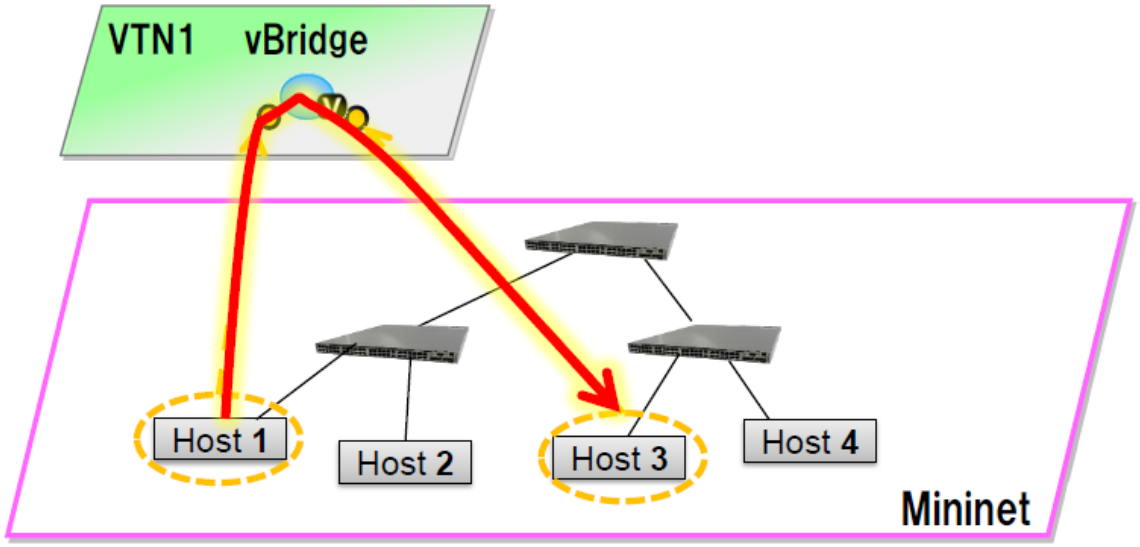


Fig. 1.143: Single Controller Mapping

Requirement

Configure mininet and create a topology

- Script for emulating network with multiple hosts.
- Before executing the mininet script, please confirm Controller is up and running.
- Run the mininet script.
- Replace <path> and <Controller IP> based on your environment.

```
sudo mn --controller=remote,ip=<Controller IP> --custom <path>\topo_handson.py --topo␣
↪mytopo2
```

```
mininet> net
h11 h11-eth0:s1-eth1
h12 h12-eth0:s1-eth2
h21 h21-eth0:s2-eth1
h22 h22-eth0:s2-eth2
h23 h23-eth0:s2-eth3
srvc1 srvc1-eth0:s3-eth3 srvc1-eth1:s4-eth3
srvc2 srvc2-eth0:s3-eth4 srvc2-eth1:s4-eth4
```

```
s1 lo:  s1-eth1:h11-eth0 s1-eth2:h12-eth0 s1-eth3:s2-eth4 s1-eth4:s3-eth2
s2 lo:  s2-eth1:h21-eth0 s2-eth2:h22-eth0 s2-eth3:h23-eth0 s2-eth4:s1-eth3 s2-eth5:s4-
↪eth1
s3 lo:  s3-eth1:s4-eth2 s3-eth2:s1-eth4 s3-eth3:srvc1-eth0 s3-eth4:srvc2-eth0
s4 lo:  s4-eth1:s2-eth5 s4-eth2:s3-eth1 s4-eth3:srvc1-eth1 s4-eth4:srvc2-eth1
```

Configuration

To create Mac Map in VTN, execute REST API provided by VTN Manager as follows. It uses curl command to call REST API.

- Create a virtual tenant named Tenant1 by executing the update-vtn RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn:update-vtn -d '{"input":{"tenant-name":
↪"Tenant1"}}'
```

- Create a virtual bridge named vBridge1 in the tenant Tenant1 by executing the update-vbridge RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vbridge:update-vbridge -d '{"input":{"tenant-
↪name":"Tenant1","bridge-name":"vBridge1"}}'
```

- Configuring Mac Mappings on the vBridge1 by giving the mac address of host h12 and host h22 as follows to allow the communication by executing the set-mac-map RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-mac-map:set-mac-map -d '{"input":{"operation
↪":"SET","allowed-hosts":["de:05:40:c4:96:76@0","62:c5:33:bc:d7:4e@0"],"tenant-name":
↪"Tenant1","bridge-name":"vBridge1"}}'
```

Note: Mac Address of host h12 and host h22 can be obtained with the following command in mininet.

```
mininet> h12 ifconfig
h12-eth0  Link encap:Ethernet  HWaddr 62:c5:33:bc:d7:4e
inet addr:10.0.0.2  Bcast:10.255.255.255  Mask:255.0.0.0
inet6 addr: fe80::60c5:33ff:febc:d74e/64 Scope:Link
```

```
mininet> h22 ifconfig
h22-eth0  Link encap:Ethernet  HWaddr de:05:40:c4:96:76
inet addr:10.0.0.4  Bcast:10.255.255.255  Mask:255.0.0.0
inet6 addr: fe80::dc05:40ff:fec4:9676/64 Scope:Link
```

- MAC Mapping will not be activated just by configuring it, a two end communication needs to be established to activate Mac Mapping.
- Ping host h22 from host h12 in mininet, the ping will not happen between the hosts as only one way activation is enabled.

```
mininet> h12 ping h22
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable
From 10.0.0.2 icmp_seq=2 Destination Host Unreachable
```

- Ping host h12 from host h22 in mininet, now the ping communication will take place as the two end communication is enabled.

```
mininet> h22 ping h12
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=91.8 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.510 ms
```

- After two end communication enabled, now host h12 can ping host h22

```
mininet> h12 ping h22
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_req=1 ttl=64 time=0.780 ms
64 bytes from 10.0.0.4: icmp_req=2 ttl=64 time=0.079 ms
```

Verification

- To view the configured Mac Map of allowed host execute the following command.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X GET http://
↪localhost:8181/restconf/operational/vtn:vtns/vtn/Tenant1/vbridge/vBridge1/mac-map
```

```
{
  "mac-map": {
    "mac-map-status": {
      "mapped-host": [
      {
        "mac-address": "c6:44:22:ba:3e:72",
          "vlan-id": 0,
          "port-id": "openflow:1:2"
      },
      {
        "mac-address": "f6:e0:43:b6:3a:b7",
        "vlan-id": 0,
        "port-id": "openflow:2:2"
      }
      ]
    },
      "mac-map-config": {
        "allowed-hosts": {
          "vlan-host-desc-list": [
          {
            "host": "c6:44:22:ba:3e:72@0"
          },
          {
            "host": "f6:e0:43:b6:3a:b7@0"
          }
          ]
        }
      }
  }
}
```

Note: When Deny is configured a broadcast message is sent to all the hosts connected to the vBridge, so a two end communication need not be establihed like allow, the hosts can communicate directly without any two way communi-

cation enabled.

1. To Deny host h23 communication from hosts connected on vBridge1, the following configuration can be applied.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-mac-map:set-mac-map -d '{"input":{"operation
↪": "SET", "denied-hosts": ["0a:d3:ea:3d:8f:a5@0"],"tenant-name": "Tenant1","bridge-
↪name": "vBridge1"}}'
```

Cleaning Up

- You can delete the virtual tenant Tenant1 by executing the remove-vtn RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn:remove-vtn -d '{"input":{"tenant-name":
↪"Tenant1"}}'
```

How To Configure Flowfilters

Overview

- This page explains how to provision flowfilter using VTN Manager. This page targets Boron release, so the procedure described here does not work in other releases.
- The flow-filter function discards, permits, or redirects packets of the traffic within a VTN, according to specified flow conditions. The table below lists the actions to be applied when a packet matches the condition:

Action	Function
Pass	Permits the packet to pass along the determined path. As options, packet transfer priority (set priority) and DSCP change (set ip-dscp) is specified.
Drop	Discards the packet.
Redirect	Redirects the packet to a desired virtual interface. As an option, it is possible to change the MAC address when the packet is transferred.

- Following steps explain flow-filter function:
  - when a packet is transferred to an interface within a virtual network, the flow-filter function evaluates whether the transferred packet matches the condition specifed in the flow-list.
  - If the packet matches the condition, the flow-filter applies the flow-list matching action specified in the flow-filter.

Requirements

To apply the packet filter, configure the following:

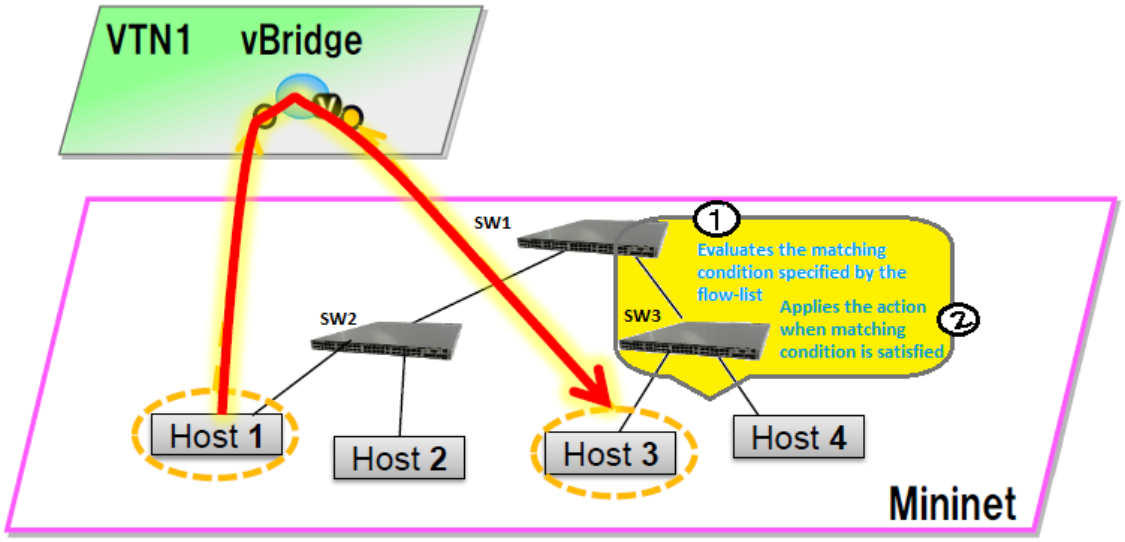


Fig. 1.144: Flow Filter Example

- Create a flow condition.
- Specify where to apply the flow-filter, for example VTN, vBridge, or interface of vBridge.

To provision OpenFlow switches, this page uses Mininet. Mininet details and set-up can be referred at the below page: https://wiki.opendaylight.org/view/OpenDaylight_Controller:Installation#Using_Mininet

Start Mininet, and create three switches (s1, s2, and s3) and four hosts (h1, h2, h3 and h4) in it.

```
sudo mn --controller=remote,ip=192.168.0.100 --topo tree,2
```

Note: Replace "192.168.0.100" with the IP address of OpenDaylight controller based on your environment.

You can check the topology that you have created by executing "net" command in the Mininet console.

```
mininet> net
h1 h1-eth0:s2-eth1
h2 h2-eth0:s2-eth2
h3 h3-eth0:s3-eth1
h4 h4-eth0:s3-eth2
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s3-eth3
s2 lo:  s2-eth1:h1-eth0 s2-eth2:h2-eth0 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h3-eth0 s3-eth2:h4-eth0 s3-eth3:s1-eth2
```

In this guide, you will provision flowfilters to establish communication between h1 and h3.

Configuration

To provision the virtual L2 network for the two hosts (h1 and h3), execute REST API provided by VTN Manager as follows. It uses curl command to call the REST API.

- Create a virtual tenant named vtn1 by executing the update-vtn RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn:update-vtn -d '{"input":{"tenant-name":"vtn1
↪"}}'
```

- Create a virtual bridge named vbr1 in the tenant vtn1 by executing the update-vbridge RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vbridge:update-vbridge -d '{"input":{"tenant-
↪name":"vtn1","bridge-name":"vbr1"}}'
```

- Create two interfaces into the virtual bridge by executing the update-vinterface RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vinterface:update-vinterface -d '{"input":{
↪"tenant-name":"vtn1","bridge-name":"vbr1","interface-name":"if1"}}'
```

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vinterface:update-vinterface -d '{"input":{
↪"tenant-name":"vtn1","bridge-name":"vbr1","interface-name":"if2"}}'
```

- Configure two mappings on the interfaces by executing the set-port-map RPC.
  - The interface if1 of the virtual bridge will be mapped to the port "s2-eth1" of the switch "openflow:2" of the Mininet.
    - The h1 is connected to the port "s2-eth1".
  - The interface if2 of the virtual bridge will be mapped to the port "s3-eth1" of the switch "openflow:3" of the Mininet.
    - The h3 is connected to the port "s3-eth1".

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-port-map:set-port-map -d '{"input":{"tenant-
↪name":"vtn1", "bridge-name":"vbr1", "interface-name":"if1", "node":"openflow:2",
↪"port-name":"s2-eth1"}}'
```

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-port-map:set-port-map -d '{"input":{"tenant-
↪name":"vtn1", "bridge-name":"vbr1", "interface-name":"if2", "node":"openflow:3",
↪"port-name":"s3-eth1"}}'
```

- Create flowcondition named cond_1 by executing the set-flow-condition RPC.
  - For option source and destination-network, get inet address of host h1 and h3 from mininet.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-condition:set-flow-condition -d '{"input
↪":{"name":"cond_1", "vtn-flow-match":[{"vtn-ether-match":{},"vtn-inet-match":{
↪"source-network":"10.0.0.1/32","protocol":1,"destination-network":"10.0.0.3/32"},
↪"index":"1"}]}}'
```

- Flowfilter can be applied either in VTN, VBR or VBR Interfaces. Here in this page we provision flowfilter with VBR Interface and demonstrate with action type drop and then pass.
- Flow filter demonstration with DROP action-type. Create Flowfilter in VBR Interface if1 by executing the set-flow-filter RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-filter:set-flow-filter -d '{"input": {
↪"tenant-name": "vtn1", "bridge-name": "vbr1","interface-name":"if1","vtn-flow-filter
↪":[{"condition":"cond_1","vtn-drop-filter":{},"vtn-flow-action":[{"order": "1","vtn-
↪set-inet-src-action":{"ipv4-address":"10.0.0.1/32"}},{"order": "2","vtn-set-inet-
↪dst-action":{"ipv4-address":"10.0.0.3/32"}}],"index": "1"}]}}'
```

Verification of the drop filter

- Please execute ping from h1 to h3. As we have applied the action type "drop" , ping should fail with no packet flows between hosts h1 and h3 as below,

```
mininet> h1 ping h3
```

Configuration for pass filter

- Update the flow filter to pass the packets by executing the set-flow-filter RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-filter:set-flow-filter -d '{"input": {
↪"tenant-name": "vtn1", "bridge-name": "vbr1","interface-name":"if1","vtn-flow-filter
↪":[{"condition":"cond_1","vtn-pass-filter":{},"vtn-flow-action":[{"order": "1","vtn-
↪set-inet-src-action":{"ipv4-address":"10.0.0.1/32"}},{"order": "2","vtn-set-inet-
↪dst-action":{"ipv4-address":"10.0.0.3/32"}}],"index": "1"}]}}'
```

Verification For Packets Success

- As we have applied action type PASS now ping should happen between hosts h1 and h3.

```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=0.984 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=0.110 ms
64 bytes from 10.0.0.3: icmp_req=3 ttl=64 time=0.098 ms
```

- You can also verify the configurations by executing the following REST API. It shows all configuration in VTN Manager.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X GET http://
↪localhost:8181/restconf/operational/vtn:vtns/vtn/vtn1
```

```
{
  "vtn": [
  {
    "name": "vtn1",
      "vtenant-config": {
        "hard-timeout": 0,
        "idle-timeout": 300,
        "description": "creating vtn"
      },
      "vbridge": [
      {
```

```
        "name": "vbr1",
        "vbridge-config": {
          "age-interval": 600,
          "description": "creating vBridge1"
        },
        "bridge-status": {
          "state": "UP",
          "path-faults": 0
        },
        "vinterface": [
        {
          "name": "if1",
          "vinterface-status": {
            "mapped-port": "openflow:2:1",
            "state": "UP",
            "entity-state": "UP"
          },
          "port-map-config": {
            "vlan-id": 0,
            "node": "openflow:2",
            "port-name": "s2-eth1"
          },
          "vinterface-config": {
            "description": "Creating if1 interface",
            "enabled": true
          },
          "vinterface-input-filter": {
            "vtn-flow-filter": [
            {
              "index": 1,
              "condition": "cond_1",
              "vtn-flow-action": [
              {
                "order": 1,
                "vtn-set-inet-src-action": {
                  "ipv4-address": "10.0.0.1/32"
                }
              },
              {
                "order": 2,
                "vtn-set-inet-dst-action": {
                  "ipv4-address": "10.0.0.3/32"
                }
              }
              ],
                "vtn-pass-filter": {}
            },
            {
              "index": 10,
              "condition": "cond_1",
              "vtn-drop-filter": {}
            }
            ]
          }
        },
        {
          "name": "if2",
          "vinterface-status": {
```

```
            "mapped-port": "openflow:3:1",
            "state": "UP",
            "entity-state": "UP"
          },
          "port-map-config": {
            "vlan-id": 0,
            "node": "openflow:3",
            "port-name": "s3-eth1"
          },
          "vinterface-config": {
            "description": "Creating if2 interface",
            "enabled": true
          }
        }
        ]
      }
    ]
  }
  ]
}
```

Cleaning Up

- To clean up both VTN and flowcondition.
- You can delete the virtual tenant vtn1 by executing the remove-vtn RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn:remove-vtn -d '{"input":{"tenant-name":"vtn1
↪"}}'
```

- You can delete the flowcondition cond_1 by executing the remove-flow-condition RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-condition:remove-flow-condition -d '{
↪"input":{"name":"cond_1"}}'
```

How to use VTN to change the path of the packet flow

Overview

- This page explains how to create specific VTN Pathmap using VTN Manager. This page targets Boron release, so the procedure described here does not work in other releases.

Requirement

- Save the mininet script given below as pathmap_test.py and run the mininet script in the mininet environment where Mininet is installed.
- Create topology using the below mininet script:

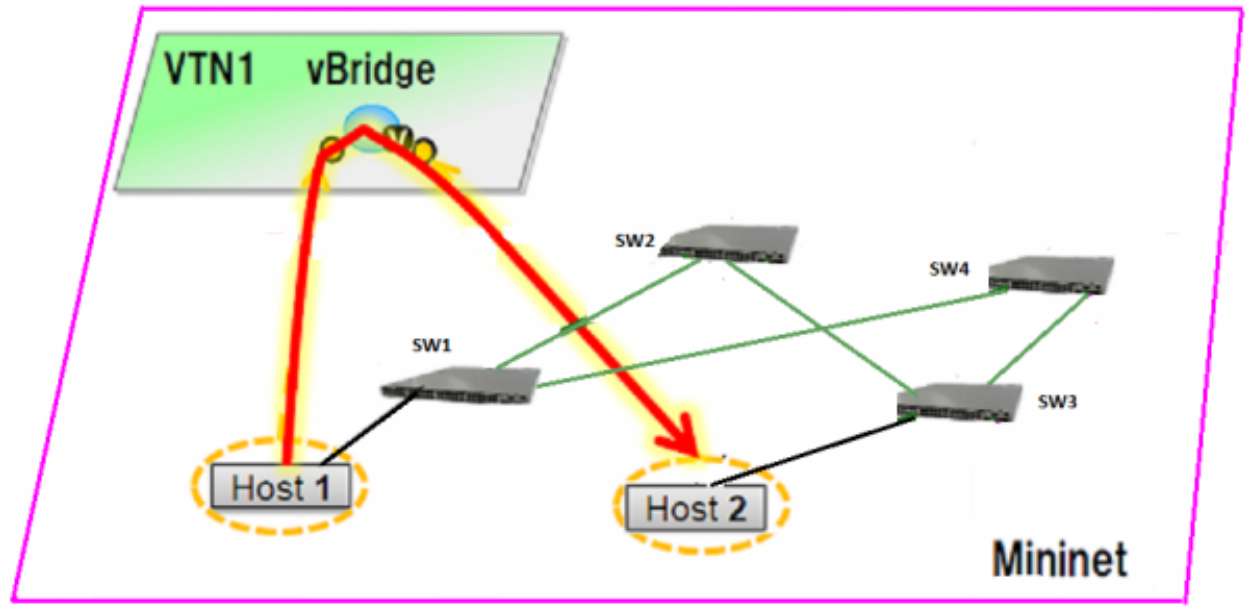


Fig. 1.145: Pathmap

```
from mininet.topo import Topo
class MyTopo( Topo ):
   "Simple topology example."
   def __init__( self ):
       "Create custom topo."
       # Initialize topology
       Topo.__init__( self )
       # Add hosts and switches
       leftHost = self.addHost( 'h1' )
       rightHost = self.addHost( 'h2' )
       leftSwitch = self.addSwitch( 's1' )
       middleSwitch = self.addSwitch( 's2' )
       middleSwitch2 = self.addSwitch( 's4' )
       rightSwitch = self.addSwitch( 's3' )
       # Add links
       self.addLink( leftHost, leftSwitch )
       self.addLink( leftSwitch, middleSwitch )
       self.addLink( leftSwitch, middleSwitch2 )
       self.addLink( middleSwitch, rightSwitch )
       self.addLink( middleSwitch2, rightSwitch )
       self.addLink( rightSwitch, rightHost )
topos = { 'mytopo': ( lambda: MyTopo() ) }
```

- After creating new file with the above script start the mininet as below,

```
sudo mn --controller=remote,ip=10.106.138.124 --custom pathmap_test.py --topo mytopo
```

Note: Replace "10.106.138.124" with the IP address of OpenDaylight controller based on your environment.

```
mininet> net
h1 h1-eth0:s1-eth1
```

```
h2 h2-eth0:s3-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth1 s1-eth3:s4-eth1
s2 lo:  s2-eth1:s1-eth2 s2-eth2:s3-eth1
s3 lo:  s3-eth1:s2-eth2 s3-eth2:s4-eth2 s3-eth3:h2-eth0
s4 lo:  s4-eth1:s1-eth3 s4-eth2:s3-eth2
c0
```

- Generate traffic by pinging between host h1 and host h2 before creating the portmaps respectively.

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
```

Configuration

- To change the path of the packet flow, execute REST API provided by VTN Manager as follows. It uses curl command to call the REST API.
- Create a virtual tenant named vtn1 by executing the update-vtn RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn:update-vtn -d '{"input":{"tenant-name":"vtn1
↪"}}'
```

- Create a virtual bridge named vbr1 in the tenant vtn1 by executing the update-vbridge RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vbridge:update-vbridge -d '{"input":{"tenant-
↪name":"vtn1","bridge-name":"vbr1"}}'
```

- Create two interfaces into the virtual bridge by executing the update-vinterface RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vinterface:update-vinterface -d '{"input":{
↪"tenant-name":"vtn1","bridge-name":"vbr1","interface-name":"if1"}}'
```

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-vinterface:update-vinterface -d '{"input":{
↪"tenant-name":"vtn1","bridge-name":"vbr1","interface-name":"if2"}}'
```

- Configure two mappings on the interfaces by executing the set-port-map RPC.
  - The interface if1 of the virtual bridge will be mapped to the port “s2-eth1” of the switch “openflow:1” of the Mininet.
    - The h1 is connected to the port “s1-eth1”.
  - The interface if2 of the virtual bridge will be mapped to the port “s3-eth1” of the switch “openflow:3” of the Mininet.
    - The h3 is connected to the port “s3-eth3”.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-port-map:set-port-map -d '{"input":{"tenant-
↪name":"vtn1", "bridge-name":"vbr1", "interface-name":"if1", "node":"openflow:1",
↪"port-name":"s1-eth1"}}'
```

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-port-map:set-port-map -d '{"input":{"tenant-
↪name":"vtn1", "bridge-name":"vbr1", "interface-name":"if2", "node":"openflow:3",
↪"port-name":"s3-eth3"}}'
```

- Genarate traffic by pinging between host h1 and host h2 after creating the portmaps respectively.

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.861 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.101 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.101 ms
```

- Get the Dataflows information by executing the get-data-flow RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow:get-data-flow -d '{"input":{"tenant-name
↪":"vtn1","mode":"DETAIL","node":"openflow:1","data-flow-port":{"port-id":1,"port-
↪name":"s1-eth1"}}}'
```

- Create flowcondition named cond_1 by executing the set-flow-condition RPC.
  - For option source and destination-network, get inet address of host h1 or host h2 from mininet

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-condition:set-flow-condition -d '{"input
↪":{"operation":"SET","present":"false","name":"cond_1", "vtn-flow-match":[{"vtn-
↪ether-match":{},"vtn-inet-match":{"source-network":"10.0.0.1/32","protocol":1,
↪"destination-network":"10.0.0.2/32"},"index":"1"}]}}'
```

- Create pathmap with flowcondition cond_1 by executing the set-path-map RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-path-map:set-path-map -d '{"input":{"tenant-
↪name":"vtn1","path-map-list":[{"condition":"cond_1","policy":"1","index": "1","idle-
↪timeout":"300","hard-timeout":"0"}]}}'
```

- Create pathpolicy by executing the set-path-policy RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-path-policy:set-path-policy -d '{"input":{
↪"operation":"SET","id": "1","default-cost": "10000","vtn-path-cost": [{"port-desc":
↪"openflow:1,3,s1-eth3","cost":"1000"},{"port-desc":"openflow:4,2,s4-eth2","cost":
↪"1000"},{"port-desc":"openflow:3,3,s3-eth3","cost":"100000"}]}}'
```

Verification

- Before applying Path policy get node information by executing get dataflow command.

```
"data-flow-info": [
{
  "physical-route": [
  {
    "physical-ingress-port": {
      "port-name": "s3-eth3",
```

```
        "port-id": "3"
    },
      "physical-egress-port": {
        "port-name": "s3-eth1",
        "port-id": "1"
      },
      "node": "openflow:3",
      "order": 0
  },
  {
    "physical-ingress-port": {
      "port-name": "s2-eth2",
      "port-id": "2"
    },
    "physical-egress-port": {
      "port-name": "s2-eth1",
      "port-id": "1"
    },
    "node": "openflow:2",
    "order": 1
  },
  {
    "physical-ingress-port": {
      "port-name": "s1-eth2",
      "port-id": "2"
    },
    "physical-egress-port": {
      "port-name": "s1-eth1",
      "port-id": "1"
    },
    "node": "openflow:1",
    "order": 2
  }
  ],
    "data-egress-node": {
      "interface-name": "if1",
      "bridge-name": "vbr1",
      "tenant-name": "vtn1"
    },
    "data-egress-port": {
      "node": "openflow:1",
      "port-name": "s1-eth1",
      "port-id": "1"
    },
    "data-ingress-node": {
      "interface-name": "if2",
      "bridge-name": "vbr1",
      "tenant-name": "vtn1"
    },
    "data-ingress-port": {
      "node": "openflow:3",
      "port-name": "s3-eth3",
      "port-id": "3"
    },
    "flow-id": 32
  },
}
```

- After applying Path policy get node information by executing get dataflow command.


```
"data-flow-info": [
{
  "physical-route": [
  {
    "physical-ingress-port": {
      "port-name": "s1-eth1",
        "port-id": "1"
    },
      "physical-egress-port": {
        "port-name": "s1-eth3",
        "port-id": "3"
      },
      "node": "openflow:1",
      "order": 0
  },
  {
    "physical-ingress-port": {
      "port-name": "s4-eth1",
      "port-id": "1"
    },
    "physical-egress-port": {
      "port-name": "s4-eth2",
      "port-id": "2"
    },
    "node": "openflow:4",
    "order": 1
  },
  {
    "physical-ingress-port": {
      "port-name": "s3-eth2",
      "port-id": "2"
    },
    "physical-egress-port": {
      "port-name": "s3-eth3",
      "port-id": "3"
    },
    "node": "openflow:3",
    "order": 2
  }
  ],
    "data-egress-node": {
      "interface-name": "if2",
      "bridge-name": "vbr1",
      "tenant-name": "vtn1"
    },
    "data-egress-port": {
      "node": "openflow:3",
      "port-name": "s3-eth3",
      "port-id": "3"
    },
    "data-ingress-node": {
      "interface-name": "if1",
      "bridge-name": "vbr1",
      "tenant-name": "vtn1"
    },
    "data-ingress-port": {
      "node": "openflow:1",
```

```
      "port-name": "s1-eth1",
      "port-id": "1"
    },
}
```

Cleaning Up

- To clean up both VTN and flowcondition.
- You can delete the virtual tenant vtn1 by executing the remove-vtn RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn:remove-vtn -d '{"input":{"tenant-name":"vtn1
↪"}}'
```

- You can delete the flowcondition cond_1 by executing the remove-flow-condition RPC.

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-condition:remove-flow-condition -d '{
↪"input":{"name":"cond_1"}}'
```

VTN Coordinator Usage Examples

How to configure L2 Network with Single Controller

Overview

This example provides the procedure to demonstrate configuration of VTN Coordinator with L2 network using VTN Virtualization(single controller). Here is the Example for vBridge Interface Mapping with Single Controller using mininet. mininet details and set-up can be referred at below URL: https://wiki.opendaylight.org/view/OpenDaylight_Controller:Installation#Using_Mininet

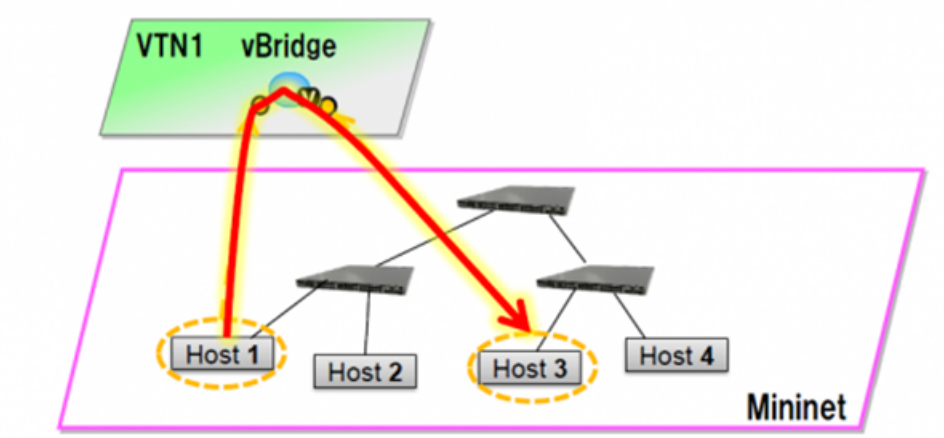


Fig. 1.146: EXAMPLE DEMONSTRATING SINGLE CONTROLLER

Requirements

- Configure mininet and create a topology:

```
mininet@mininet-vm:~$ sudo mn --controller=remote,ip=<controller-ip> --topo tree,2
```

- mininet> net

```
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth1
s2 lo:  s2-eth1:s1-eth2 s2-eth2:h2-eth0
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth2
```

Configuration

- Create a Controller named controllerone and mention its ip-address in the below create-controller command.

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{
↪"controller": {"controller_id": "controllerone", "ipaddr":"10.0.0.2", "type": "odc",
↪ "version": "1.0", "auditstatus":"enable"}}' http://127.0.0.1:8083/vtn-webapi/
↪controllers.json
```

- Create a VTN named vtn1 by executing the create-vtn command

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vtn" : {
↪"vtn_name":"vtn1","description":"test VTN" }}' http://127.0.0.1:8083/vtn-webapi/
↪vtns.json
```

- Create a vBridge named vBridge1 in the vtn1 by executing the create-vbr command.

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vbridge
↪" : {"vbr_name":"vBridge1","controller_id":"controllerone","domain_id":"(DEFAULT)" }
↪}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges.json
```

- Create two Interfaces named if1 and if2 into the vBridge1

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{
↪"interface": {"if_name": "if1","description": "if_desc1"}}' http://127.0.0.1:8083/
↪vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{
↪"interface": {"if_name": "if2","description": "if_desc2"}}' http://127.0.0.1:8083/
↪vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces.json
```

- Get the list of logical ports configured

```
Curl --user admin:adminpass -H 'content-type: application/json' -X GET http://127.0.0.
↪1:8083/vtn-webapi/controllers/controllerone/domains/\(DEFAULT\)/logical_ports.json
```

- Configure two mappings on each of the interfaces by executing the below command.

The interface if1 of the virtual bridge will be mapped to the port “s2-eth1” of the switch “openflow:2” of the Mininet. The h1 is connected to the port “s2-eth1”.

The interface if2 of the virtual bridge will be mapped to the port “s3-eth1” of the switch “openflow:3” of the Mininet. The h3 is connected to the port “s3-eth1”.

```
curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d '{"portmap":
↪{"logical_port_id": "PP-OF:00:00:00:00:00:00:00:03-s3-eth1"}}' http://127.0.0.
↪1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if1/portmap.json
curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d '{"portmap":
↪{"logical_port_id": "PP-OF:00:00:00:00:00:00:00:02-s2-eth1"}}' http://127.0.0.
↪1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if2/portmap.json
```

Verification

Please verify whether the Host1 and Host3 are pinging.

- Send packets from Host1 to Host3

```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=0.780 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=0.079 ms
```

How to configure L2 Network with Multiple Controllers

- This example provides the procedure to demonstrate configuration of VTN Coordinator with L2 network using VTN Virtualization Here is the Example for vBridge Interface Mapping with Multi-controller using mininet.

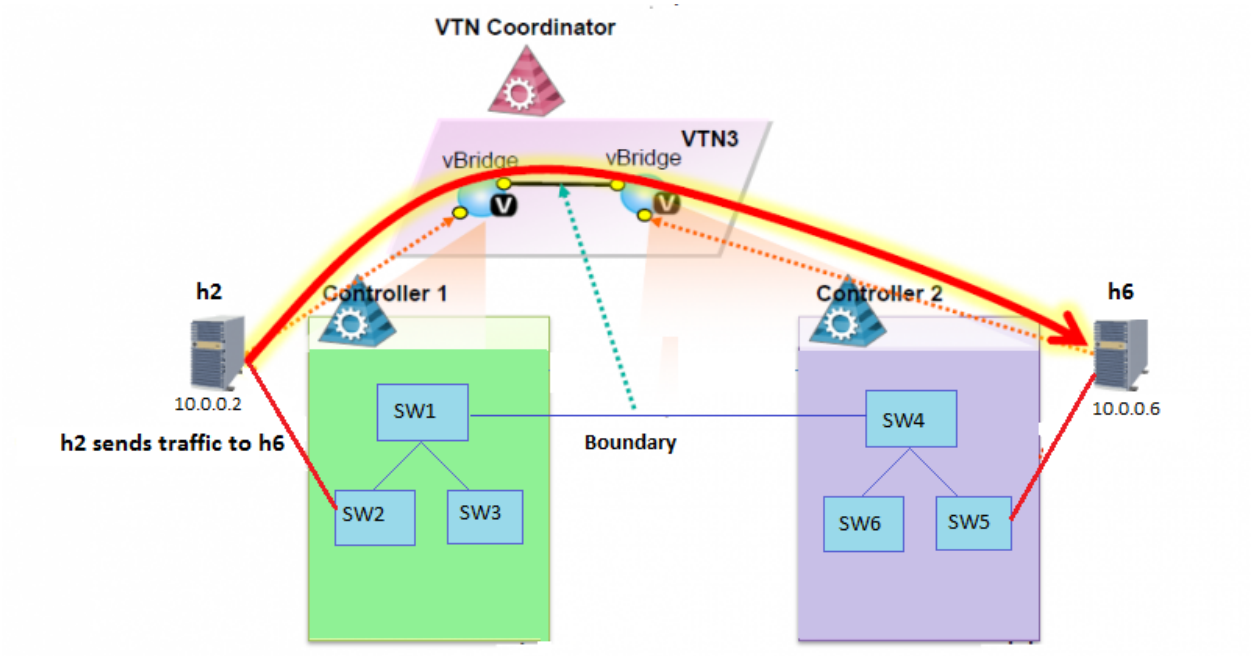


Fig. 1.147: EXAMPLE DEMONSTRATING MULTIPLE CONTROLLERS

Requirements

- Configure multiple controllers using the mininet script given below: https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_%28VTN%29:Scripts:Mininet#Network_with_multiple_switches_

and_OpenFlow_controllers

Configuration

- Create a VTN named vtn3 by executing the create-vtn command

```
curl --user admin:adminpass -H 'content-type: application/json'  -X POST -d '{"vtn" :
↪{"vtn_name":"vtn3"}}' http://127.0.0.1:8083/vtn-webapi/vtns.json
```

- Create two Controllers named odc1 and odc2 with its ip-address in the below create-controller command.

```
curl --user admin:adminpass -H 'content-type: application/json'  -X POST -d '{
↪"controller": {"controller_id": "odc1", "ipaddr":"10.100.9.52", "type": "odc",
↪"version": "1.0", "auditstatus":"enable"}}' http://127.0.0.1:8083/vtn-webapi/
↪controllers.json
```

```
curl --user admin:adminpass -H 'content-type: application/json'  -X POST -d '{
↪"controller": {"controller_id": "odc2", "ipaddr":"10.100.9.61", "type": "odc",
↪"version": "1.0", "auditstatus":"enable"}}' http://127.0.0.1:8083/vtn-webapi/
↪controllers.json
```

- Create two vBridges in the VTN like, vBridge1 in Controller1 and vBridge2 in Controller2

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vbridge
↪" : {"vbr_name":"vbr1","controller_id":"odc1","domain_id":"(DEFAULT)" }}' http://
↪127.0.0.1:8083/vtn-webapi/vtns/vtn3/vbridges.json
```

```
curl --user admin:adminpass -H 'content-type: application/json'  -X POST -d '{"vbridge
↪" : {"vbr_name":"vbr2","controller_id":"odc2","domain_id":"(DEFAULT)" }}' http://
↪127.0.0.1:8083/vtn-webapi/vtns/vtn3/vbridges.json
```

- Create two Interfaces if1, if2 for the two vBridges vbr1 and vbr2.

```
curl --user admin:adminpass -H 'content-type: application/json'  -X POST -d '{
↪"interface": {"if_name": "if1"}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/
↪vbridges/vbr1/interfaces.json
```

```
curl --user admin:adminpass -H 'content-type: application/json'  -X POST -d '{
↪"interface": {"if_name": "if2"}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/
↪vbridges/vbr1/interfaces.json
```

```
curl --user admin:adminpass -H 'content-type: application/json'  -X POST -d '{
↪"interface": {"if_name": "if1"}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/
↪vbridges/vbr2/interfaces.json
```

```
curl --user admin:adminpass -H 'content-type: application/json'  -X POST -d '{
↪"interface": {"if_name": "if2"}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/
↪vbridges/vbr2/interfaces.json
```

- Get the list of logical ports configured

```
curl --user admin:adminpass -H 'content-type: application/json' -X GET http://127.0.0.
↪1:8083/vtn-webapi/controllers/odc1/domains/\(DEFAULT\)/logical_ports/detail.json
```

- Create boundary and vLink for the two controllers

```
curl --user admin:adminpass -H 'content-type: application/json'    -X POST -d '{
↪"boundary": {"boundary_id": "b1", "link": {"controller1_id": "odc1", "domain1_id":
↪"(DEFAULT)", "logical_port1_id": "PP-OF:00:00:00:00:00:00:00:01-s1-eth3",
↪"controller2_id": "odc2", "domain2_id": "(DEFAULT)", "logical_port2_id": "PP-
↪OF:00:00:00:00:00:00:00:04-s4-eth3"}}}' http://127.0.0.1:8083/vtn-webapi/boundaries.
↪json
```

```
curl --user admin:adminpass -H 'content-type: application/json'  -X POST -d '{"vlink
↪": {"vlk_name": "vlink1" , "vnode1_name": "vbr1", "if1_name":"if2", "vnode2_name":
↪"vbr2", "if2_name": "if2", "boundary_map": {"boundary_id":"b1","vlan_id": "50"}}}'␣
↪http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/vlinks.json
```

- Configure two mappings on each of the interfaces by executing the below command.

The interface if1 of the vbr1 will be mapped to the port "s2-eth2" of the switch "openflow:2" of the Mininet. The h2 is connected to the port "s2-eth2".

The interface if2 of the vbr2 will be mapped to the port "s5-eth2" of the switch "openflow:5" of the Mininet. The h6 is connected to the port "s5-eth2".

```
curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d '{"portmap":
↪{"logical_port_id": "PP-OF:00:00:00:00:00:00:00:02-s2-eth2"}}' http://127.0.0.
↪1:8083/vtn-webapi/vtns/vtn3/vbridges/vbr1/interfaces/if1/portmap.json
```

```
curl --user admin:adminpass -H 'content-type: application/json'  -X PUT -d '{"portmap
↪":{"logical_port_id": "PP-OF:00:00:00:00:00:00:00:05-s5-eth2"}}' http://127.0.0.
↪1:8083/vtn-webapi/vtns/vtn3/vbridges/vbr2/interfaces/if1/portmap.json
```

Verification

Please verify whether Host h2 and Host h6 are pinging.

- Send packets from h2 to h6

```
mininet> h2 ping h6
```

```
PING 10.0.0.6 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.6: icmp_req=1 ttl=64 time=0.780 ms
64 bytes from 10.0.0.6: icmp_req=2 ttl=64 time=0.079 ms
```

How To Test Vlan-Map In Mininet Environment

Overview

This example explains how to test vlan-map in a multi host scenario.

Requirements

- Save the mininet script given below as vlan_vtn_test.py and run the mininet script in the mininet environment where Mininet is installed.

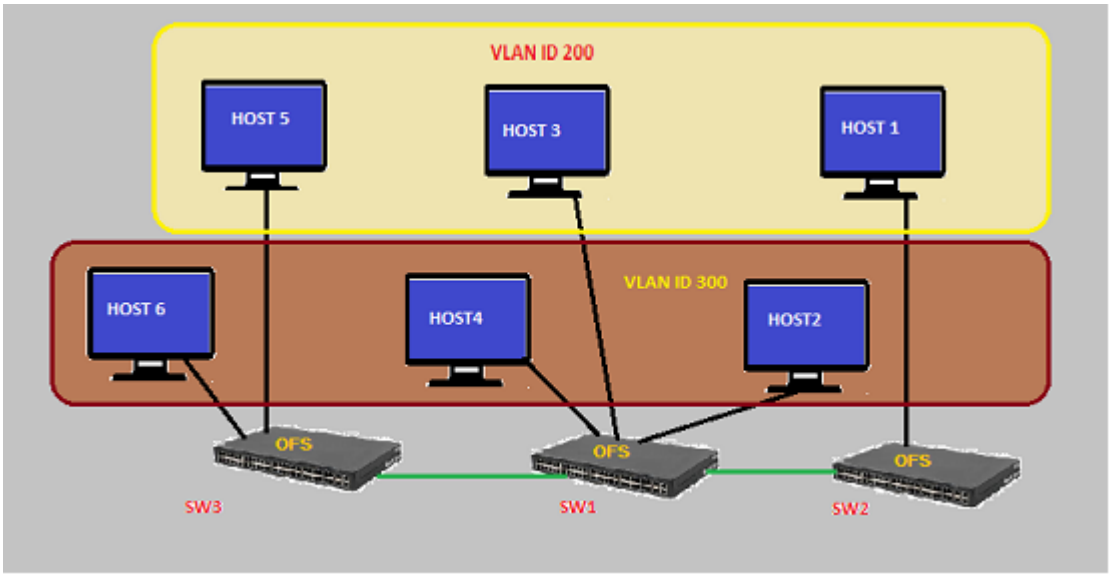


Fig. 1.148: Example that demonstrates vlanmap testing in Mininet Environment

Mininet Script

https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_(VTN):Scripts:Mininet#Network_with_hosts_in_different_vlan

- Run the mininet script

```
sudo mn --controller=remote,ip=192.168.64.13 --custom vlan_vtn_test.py --topo mytopo
```

Configuration

Please follow the below steps to test a vlan map using mininet:

- Create a Controller named controllerone and mention its ip-address in the below create-controller command.

```
curl --user admin:adminpass -H 'content-type: application/json'  -X POST -d '{
↪"controller": {"controller_id": "controllerone", "ipaddr":"10.0.0.2", "type": "odc",
↪ "version": "1.0", "auditstatus":"enable"}}' http://127.0.0.1:8083/vtn-webapi/
↪controllers
```

- Create a VTN named vtn1 by executing the create-vtn command

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H 'password:␣
↪adminpass' -d '{"vtn" : {"vtn_name":"vtn1","description":"test VTN" }}' http://127.
↪0.0.1:8083/vtn-webapi/vtns.json
```

- Create a vBridge named vBridge1 in the vtn1 by executing the create-vbr command.

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H 'password:␣
↪adminpass' -d '{"vbridge" : {"vbr_name":"vBridge1","controller_id":"controllerone",
↪"domain_id":"(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges.json
```

- Create a vlan map with vlanid 200 for vBridge vBridge1

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H 'password:␣
↪adminpass' -d '{"vlanmap" : {"vlan_id": 200 }}' http://127.0.0.1:8083/vtn-webapi/
↪vtns/vtn1/vbridges/vBridge1/vlanmaps.json
```

- Create a vBridge named vBridge2 in the vtn1 by executing the create-vbr command.

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H 'password:␣
↪adminpass' -d '{"vbridge" : {"vbr_name":"vBridge2","controller_id":"controllerone",
↪"domain_id":"(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges.json
```

- Create a vlan map with vlanid 300 for vBridge vBridge2

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H 'password:␣
↪adminpass' -d '{"vlanmap" : {"vlan_id": 300 }}' http://127.0.0.1:8083/vtn-webapi/
↪vtns/vtn1/vbridges/vBridge2/vlanmaps.json
```

Verification

Ping all in mininet environment to view the host reachability.

```
mininet> pingall
Ping: testing ping reachability
h1 -> X h3 X h5 X
h2 -> X X h4 X h6
h3 -> h1 X X h5 X
h4 -> X h2 X X h6
h5 -> h1 X h3 X X
h6 -> X h2 X h4 X
```

How To View Specific VTN Station Information.

This example demonstrates on how to view a specific VTN Station information.

Requirement

- Configure mininet and create a topology:

```
 $ sudo mn --custom /home/mininet/mininet/custom/topo-2sw-2host.py --
↪controller=remote,ip=10.100.9.61 --topo mytopo
mininet> net

 s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth1
 s2 lo:  s2-eth1:s1-eth2 s2-eth2:h2-eth0
 h1 h1-eth0:s1-eth1
 h2 h2-eth0:s2-eth2
```

- Generate traffic by pinging between hosts h1 and h2 after configuring the portmaps respectively

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=16.7 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=13.2 ms
```

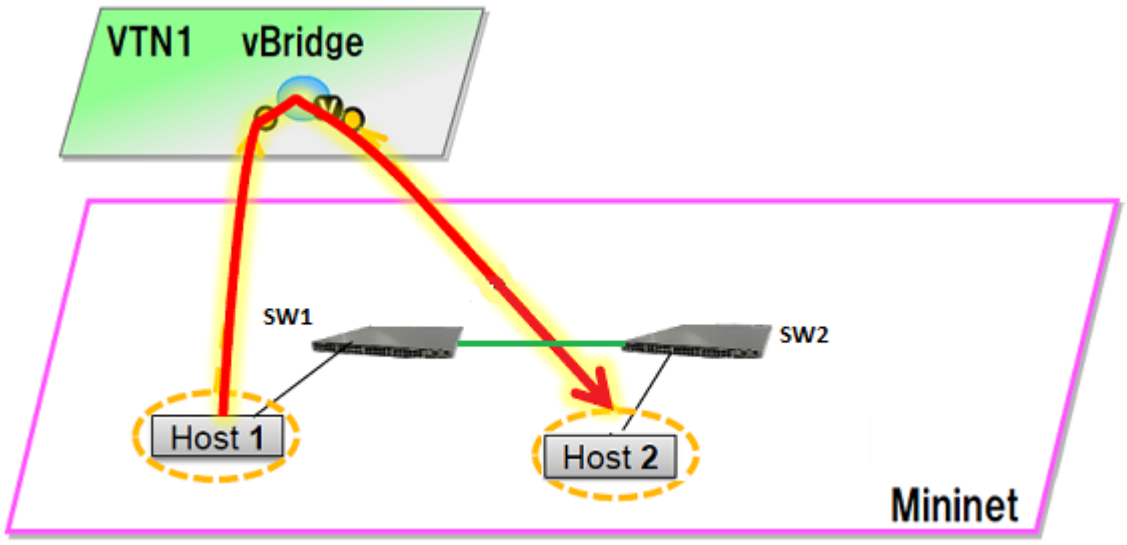


Fig. 1.149: EXAMPLE DEMONSTRATING VTN STATIONS

Configuration

- Create a Controller named controllerone and mention its ip-address in the below create-controller command

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{
↪"controller": {"controller_id": "controllerone", "ipaddr":"10.100.9.61", "type":
↪"odc", "version": "1.0", "auditstatus":"enable"}}' http://127.0.0.1:8083/vtn-webapi/
↪controllers.json
```

- Create a VTN named vtn1 by executing the create-vtn command

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vtn" : {
↪"vtn_name":"vtn1","description":"test VTN" }}' http://127.0.0.1:8083/vtn-webapi/
↪vtns.json
```

- Create a vBridge named vBridge1 in the vtn1 by executing the create-vbr command.

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vbridge
↪" : {"vbr_name":"vBridge1","controller_id":"controllerone","domain_id":"(DEFAULT)" }
↪}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges.json
```

- Create two Interfaces named if1 and if2 into the vBridge1

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{
↪"interface": {"if_name": "if1","description": "if_desc1"}}' http://127.0.0.1:8083/
↪vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces.json
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d '{
↪"interface": {"if_name": "if2","description": "if_desc2"}}' http://127.0.0.1:8083/
↪vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces.json
```

- Configure two mappings on each of the interfaces by executing the below command.

The interface if1 of the virtual bridge will be mapped to the port “s1-eth1” of the switch “openflow:1” of the Mininet. The h1 is connected to the port “s1-eth1”.

The interface if2 of the virtual bridge will be mapped to the port "s1-eth2" of the switch "openflow:1" of the Mininet. The h2 is connected to the port "s1-eth2".

```
curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d '{"portmap":
↪{"logical_port_id": "PP-OF:00:00:00:00:00:00:01-s1-eth1"}}' http://127.0.0.
↪1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if1/portmap.json
curl -v --user admin:adminpass -H 'content-type: application/json' -X PUT -d '{
↪"portmap":{"logical_port_id": "PP-OF:00:00:00:00:00:00:02-s2-eth2"}}' http://17.
↪0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if2/portmap.json
```

- Get the VTN stations information

```
curl -X GET -H 'content-type: application/json' -H 'username: admin' -H 'password:␣
↪adminpass' "http://127.0.0.1:8083/vtn-webapi/vtnstations?controller_
↪id=controllerone&vtn_name=vtn1"
```

Verification

```
curl -X GET -H 'content-type: application/json' -H 'username: admin' -H 'password:␣
↪adminpass' "http://127.0.0.1:8083/vtn-webapi/vtnstations?controller_
↪id=controllerone&vtn_name=vtn1"
{
   "vtnstations": [
       {
           "domain_id": "(DEFAULT)",
           "interface": {},
           "ipaddrs": [
               "10.0.0.2"
           ],
           "macaddr": "b2c3.06b8.2dac",
           "no_vlan_id": "true",
           "port_name": "s2-eth2",
           "station_id": "178195618445172",
           "switch_id": "00:00:00:00:00:00:00:02",
           "vnode_name": "vBridge1",
           "vnode_type": "vbridge",
           "vtn_name": "vtn1"
       },
       {
           "domain_id": "(DEFAULT)",
           "interface": {},
           "ipaddrs": [
               "10.0.0.1"
           ],
           "macaddr": "ce82.1b08.90cf",
           "no_vlan_id": "true",
           "port_name": "s1-eth1",
           "station_id": "206130278144207",
           "switch_id": "00:00:00:00:00:00:00:01",
           "vnode_name": "vBridge1",
           "vnode_type": "vbridge",
           "vtn_name": "vtn1"
       }
   ]
}
```

How To View Dataflows in VTN

This example demonstrates on how to view a specific VTN Dataflow information.

Configuration

The same Configuration as Vlan Mapping Example(https://wiki.opendaylight.org/view/VTN:Coordinator:Beryllium:HowTos:How_To_test_vlanmap_using_mininet)

Verification

Get the VTN Dataflows information

```
curl -X GET -H 'content-type: application/json' --user 'admin:adminpass' "http://127.
↪0.0.1:8083/vtn-webapi/dataflows?controller_id=controllerone&srcmacaddr=924c.e4a3.
↪a743&vlan_id=300&switch_id=openflow:2&port_name=s2-eth1"
```


```
{
   "dataflows": [
       {
           "controller_dataflows": [
               {
                   "controller_id": "controllerone",
                   "controller_type": "odc",
                   "egress_domain_id": "(DEFAULT)",
                   "egress_port_name": "s3-eth3",
                   "egress_station_id": "3",
                   "egress_switch_id": "00:00:00:00:00:00:00:03",
                   "flow_id": "29",
                   "ingress_domain_id": "(DEFAULT)",
                   "ingress_port_name": "s2-eth2",
                   "ingress_station_id": "2",
                   "ingress_switch_id": "00:00:00:00:00:00:00:02",
                   "match": {
                       "macdstaddr": [
                           "4298.0959.0e0b"
                       ],
                       "macsrcaddr": [
                           "924c.e4a3.a743"
                       ],
                       "vlan_id": [
                           "300"
                       ]
                   },
                   "pathinfos": [
                       {
                           "in_port_name": "s2-eth2",
                           "out_port_name": "s2-eth1",
                           "switch_id": "00:00:00:00:00:00:00:02"
                       },
                       {
                           "in_port_name": "s1-eth2",
                           "out_port_name": "s1-eth3",
                           "switch_id": "00:00:00:00:00:00:00:01"
                       },
```

```
                        {
                            "in_port_name": "s3-eth1",
                            "out_port_name": "s3-eth3",
                            "switch_id": "00:00:00:00:00:00:00:03"
                        }
                    ]
                }
            ],
            "reason": "success"
        }
    ]
}
```


How To Configure Flow Filters Using VTN

Overview

The flow-filter function discards, permits, or redirects packets of the traffic within a VTN, according to specified flow conditions The table below lists the actions to be applied when a packet matches the condition:

Action	Function
Pass	Permits the packet to pass. As options, packet transfer priority (set priority) and DSCP change (se t ip-dscp) is specified.
Drop	Discards the packet.
Redirect	Redirects the packet to a desired virtual interface. As an option, it is possible to change the MAC address when the packet is transferred.

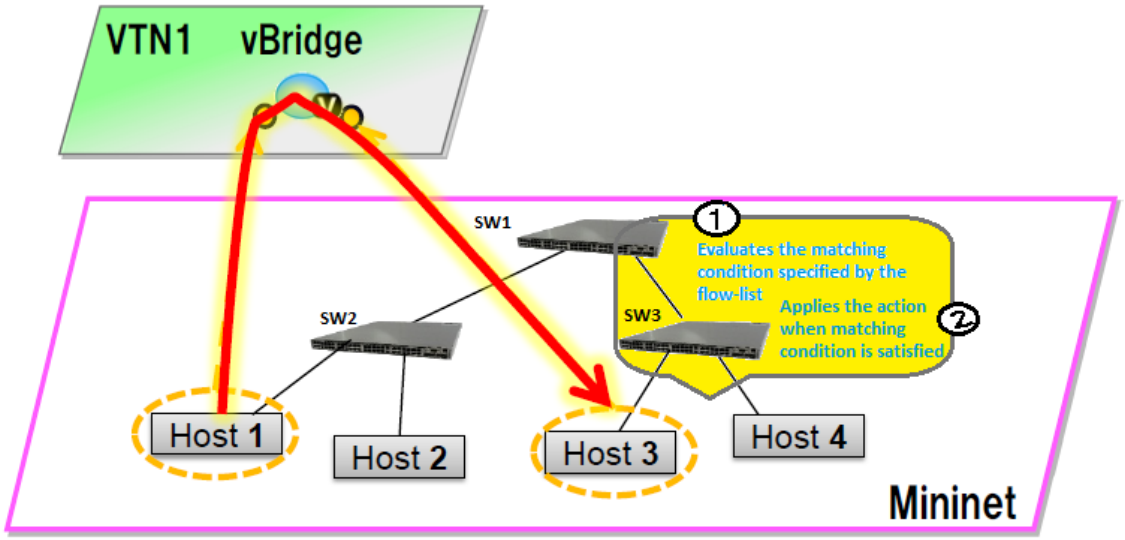


Fig. 1.150: Flow Filter

Following steps explain flow-filter function:

- When a packet is transferred to an interface within a virtual network, the flow-filter function evaluates whether the transferred packet matches the condition specified in the flow-list.

- If the packet matches the condition, the flow-filter applies the flow-list matching action specified in the flow-filter.

Requirements

To apply the packet filter, configure the following:

- Create a flow-list and flow-listentry.
- Specify where to apply the flow-filter, for example VTN, vBridge, or interface of vBridge.

Configure mininet and create a topology:

```
$  mininet@mininet-vm:~$ sudo mn --controller=remote,ip=<controller-ip> --topo tree
```

Please generate the following topology

```
$  mininet@mininet-vm:~$ sudo mn --controller=remote,ip=<controller-ip> --topo tree,2
mininet> net
c0
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s3-eth3
s2 lo:  s2-eth1:h1-eth0 s2-eth2:h2-eth0 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h3-eth0 s3-eth2:h4-eth0 s3-eth3:s1-eth2
h1 h1-eth0:s2-eth1
h2 h2-eth0:s2-eth2
h3 h3-eth0:s3-eth1
h4 h4-eth0:s3-eth2
```

Configuration

- Create a Controller named controller1 and mention its ip-address in the below create-controller command.

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{
↪"controller": {"controller_id": "controller1", "ipaddr":"10.100.9.61", "type": "odc
↪", "version": "1.0", "auditstatus":"enable"}}' http://127.0.0.1:8083/vtn-webapi/
↪controllers
```

- Create a VTN named vtn_one by executing the create-vtn command

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vtn" : {
↪"vtn_name":"vtn_one","description":"test VTN" }}' http://127.0.0.1:8083/vtn-webapi/
↪vtns.json
```

- Create a vBridge named vbr_two in the vtn1 by executing the create-vbr command.

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vbridge
↪" : {"vbr_name":"vbr_one^C"controller_id":"controller1","domain_id":"(DEFAULT)" }}'␣
↪http://127.0.0.1:8083/vtn-webapi/vtns/vtn_one/vbridges.json
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d '{
↪"vbridge" :
{"vbr_name":"vbr_two","controller_id":"controller1","domain_id":"(DEFAULT)" }}' http:/
↪/127.0.0.1:8083/vtn-webapi/vtns/vtn_one/vbridges.json
```

- Create two Interfaces named if1 and if2 into the vbr_two

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{
↪"interface": {"if_name": "if1","description": "if_desc1"}}' http://127.0.0.1:8083/
↪vtn-webapi/vtns/vtn_one/vbridges/vbr_two/interfaces.json
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d '{
↪"interface": {"if_name": "if1","description": "if_desc1"}}' http://127.0.0.1:8083/
↪vtn-webapi/vtns/vtn_one/vbridges/vbr_two/interfaces.json
```

- Get the list of logical ports configured

```
curl --user admin:adminpass -H 'content-type: application/json' -X GET  http://127.0.
↪0.1:8083/vtn-webapi/controllers/controllerone/domains/\(DEFAULT\)/logical_ports.json
```

- Configure two mappings on each of the interfaces by executing the below command.

The interface if1 of the virtual bridge will be mapped to the port "s2-eth1" of the switch "openflow:2" of the Mininet. The h1 is connected to the port "s2-eth1".

The interface if2 of the virtual bridge will be mapped to the port "s3-eth1" of the switch "openflow:3" of the Mininet. The h3 is connected to the port "s3-eth1".

```
curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d '{"portmap":
↪{"logical_port_id": "PP-OF:00:00:00:00:00:00:03-s3-eth1"}}' http://127.0.0.
↪1:8083/vtn-webapi/vtns/vtn_one/vbridges/vbr_two/interfaces/if1/portmap.json
curl -v --user admin:adminpass -H 'content-type: application/json' -X PUT -d '{
↪"portmap":{"logical_port_id": "PP-OF:00:00:00:00:00:00:02-s2-eth1"}}' http://127.
↪0.0.1:8083/vtn-webapi/vtns/vtn_one/vbridges/vbr_two/interfaces/if2/portmap.json
```

- Create Flowlist

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"flowlist
↪": {"fl_name": "flowlist1", "ip_version":"IP"}}' http://127.0.0.1:8083/vtn-webapi/
↪flowlists.json
```

- Create Flowlistentry

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{
↪"flowlistentry": {"seqnum": "233","macethertype": "0x8000","ipdstaddr": "10.0.0.3",
↪"ipdstaddrprefix": "2","ipsrcaddr": "10.0.0.2","ipsrcaddrprefix": "2","ipproto": "17
↪","ipdscp": "55","icmptypenum":"232","icmpcodenum": "232"}}' http://127.0.0.1:8083/
↪vtn-webapi/flowlists/flowlist1/flowlistentries.json
```

- Create vBridge Interface Flowfilter

```
curl --user admin:adminpass -X POST -H 'content-type: application/json' -d '{
↪"flowfilter" : {"ff_type": "in"}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn_one/
↪vbridges/vbr_two/interfaces/if1/flowfilters.json
```

Flow filter demonstration with DROP action-type

```
curl --user admin:adminpass -X POST -H 'content-type: application/json' -d '{
↪"flowfilterentry": {"seqnum": "233", "fl_name": "flowlist1", "action_type":"drop",
↪"priority":"3", "dscp":"55" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn_one/
↪vbridges/vbr_two/interfaces/if1/flowfilters/in/flowfilterentries.json
```

Verification

As we have applied the action type “drop” , ping should fail.

```
mininet> h1 ping h3
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
```

Flow filter demonstration with PASS action-type

```
curl --user admin:adminpass -X PUT -H 'content-type: application/json' -d '{
↪"flowfilterentry": {"seqnum": "233", "fl_name": "flowlist1", "action_type":"pass",
↪"priority":"3", "dscp":"55" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn_one/
↪vbridges/vbr_two/interfaces/if1/flowfilters/in/flowfilterentries/233.json
```

Verification

```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=0.984 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=0.110 ms
64 bytes from 10.0.0.3: icmp_req=3 ttl=64 time=0.098 ms
```

How To Use VTN To Make Packets Take Different Paths

This example demonstrates on how to create a specific VTN Path Map information.

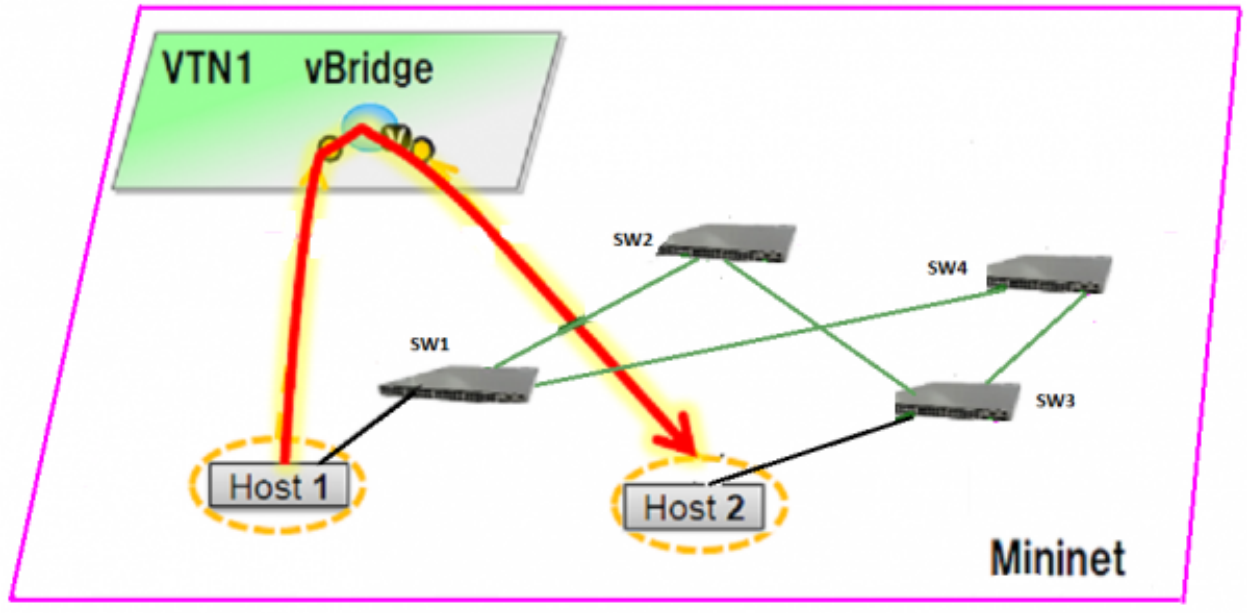


Fig. 1.151: PathMap

Requirement

- Save the mininet script given below as pathmap_test.py and run the mininet script in the mininet environment where Mininet is installed.
- Create topology using the below mininet script:

```
from mininet.topo import Topo
class MyTopo( Topo ):
   "Simple topology example."
   def __init__( self ):
       "Create custom topo."
       # Initialize topology
       Topo.__init__( self )
       # Add hosts and switches
       leftHost = self.addHost( 'h1' )
       rightHost = self.addHost( 'h2' )
       leftSwitch = self.addSwitch( 's1' )
       middleSwitch = self.addSwitch( 's2' )
       middleSwitch2 = self.addSwitch( 's4' )
       rightSwitch = self.addSwitch( 's3' )
       # Add links
       self.addLink( leftHost, leftSwitch )
       self.addLink( leftSwitch, middleSwitch )
       self.addLink( leftSwitch, middleSwitch2 )
       self.addLink( middleSwitch, rightSwitch )
       self.addLink( middleSwitch2, rightSwitch )
       self.addLink( rightSwitch, rightHost )
topos = { 'mytopo': ( lambda: MyTopo() ) }
```

```
mininet> net
c0
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth1 s1-eth3:s4-eth1
s2 lo:  s2-eth1:s1-eth2 s2-eth2:s3-eth1
s3 lo:  s3-eth1:s2-eth2 s3-eth2:s4-eth2 s3-eth3:h2-eth0
s4 lo:  s4-eth1:s1-eth3 s4-eth2:s3-eth2
h1 h1-eth0:s1-eth1
h2 h2-eth0:s3-eth3
```

- Generate traffic by pinging between hosts h1 and h2 before creating the portmaps respectively

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
```

Configuration

- Create a Controller named controller1 and mention its ip-address in the below create-controller command.

```
curl --user admin:adminpass -H 'content-type: application/json'  -X POST -d '{
↪"controller": {"controller_id": "odc", "ipaddr":"10.100.9.42", "type": "odc",
↪"version": "1.0", "auditstatus":"enable"}}' http://127.0.0.1:8083/vtn-webapi/
↪controllers.json
```

- Create a VTN named vtn1 by executing the create-vtn command

```
curl --user admin:adminpass -H 'content-type: application/json'  -X POST -d '{"vtn" :
↪{"vtn_name":"vtn1","description":"test VTN" }}' http://127.0.0.1:8083/vtn-webapi/
↪vtns.json
```

- Create a vBridge named vBridge1 in the vtn1 by executing the create-vbr command.

```
curl --user admin:adminpass -H 'content-type: application/json'  -X POST -d '{"vbridge
↪" : {"vbr_name":"vBridge1","controller_id":"odc","domain_id":"(DEFAULT)" }}' http://
↪127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges.json
```

- Create two Interfaces named if1 and if2 into the vBridge1

```
curl --user admin:adminpass -H 'content-type: application/json'  -X POST -d '{
↪"interface": {"if_name": "if1","description": "if_desc1"}}' http://127.0.0.1:8083/
↪vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces.json
curl --user admin:adminpass -H 'content-type: application/json'  -X POST -d '{
↪"interface": {"if_name": "if2","description": "if_desc2"}}' http://127.0.0.1:8083/
↪vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces.json
```

- Configure two mappings on each of the interfaces by executing the below command.

The interface if1 of the virtual bridge will be mapped to the port "s1-eth1" of the switch "openflow:1" of the Mininet. The h1 is connected to the port "s1-eth1".

The interface if2 of the virtual bridge will be mapped to the port "s3-eth3" of the switch "openflow:3" of the Mininet. The h2 is connected to the port "s3-eth3".

```
curl --user admin:adminpass -H 'content-type: application/json'  -X PUT -d '{"portmap
↪":{"logical_port_id": "PP-OF:00:00:00:00:00:00:00:01-s1-eth1"}}' http://127.0.0.
↪1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if1/portmap.json
curl --user admin:adminpass -H 'content-type: application/json'  -X PUT -d '{"portmap
↪":{"logical_port_id": "PP-OF:00:00:00:00:00:00:03-s3-eth3"}}' http://127.0.0.
↪1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if2/portmap.json
```

- Generate traffic by pinging between hosts h1 and h2 after creating the portmaps respectively

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=36.4 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.880 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_req=4 ttl=64 time=0.081 ms
```

- Get the VTN Dataflows information

```
curl -X GET -H 'content-type: application/json' --user 'admin:adminpass' "http://127.
↪0.0.1:8083/vtn-webapi/dataflows?&switch_id=00:00:00:00:00:00:00:01&port_name=s1-
↪eth1&controller_id=odc&srcmacaddr=de3d.7dec.e4d2&no_vlan_id=true"
```

- Create a Flowcondition in the VTN

(The flowconditions, pathmap and pathpolicy commands have to be executed in the controller).

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-flow-condition:set-flow-condition -d '{"input
↪":{"operation":"SET","present":"false","name":"cond_1", "vtn-flow-match":[{"vtn-
↪ether-match":{},"vtn-inet-match":{"source-network":"10.0.0.1/32","protocol":1,
↪"destination-network":"10.0.0.2/32"},"index":"1"}]}}'
```

- Create a Pathmap in the VTN

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-path-map:set-path-map -d '{"input":{"tenant-
↪name":"vtn1","path-map-list":[{"condition":"cond_1","policy":"1","index": "1","idle-
↪timeout":"300","hard-timeout":"0"}]}}'
```

- Get the Path policy information

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST http://
↪localhost:8181/restconf/operations/vtn-path-policy:set-path-policy -d '{"input":{
↪"operation":"SET","id": "1","default-cost": "10000","vtn-path-cost": [{"port-desc":
↪"openflow:1,3,s1-eth3","cost":"1000"},{"port-desc":"openflow:4,2,s4-eth2","cost":
↪"100000"},{"port-desc":"openflow:3,3,s3-eth3","cost":"10000"}]}}'
```

Verification

- Before applying Path policy information in the VTN

```
{
        "pathinfos": [
            {
              "in_port_name": "s1-eth1",
              "out_port_name": "s1-eth3",
              "switch_id": "openflow:1"
            },
            {
              "in_port_name": "s4-eth1",
              "out_port_name": "s4-eth2",
              "switch_id": "openflow:4"
            },
            {
               "in_port_name": "s3-eth2",
               "out_port_name": "s3-eth3",
               "switch_id": "openflow:3"
            }
                    ]
}
```

- After applying Path policy information in the VTN

```
{
    "pathinfos": [
            {
              "in_port_name": "s1-eth1",
              "out_port_name": "s1-eth2",
              "switch_id": "openflow:1"
            },
            {
              "in_port_name": "s2-eth1",
              "out_port_name": "s2-eth2",
              "switch_id": "openflow:2"
            },
            {
               "in_port_name": "s3-eth1",
```

```
                "out_port_name": "s3-eth3",
                "switch_id": "openflow:3"
            }
                    ]
}
```

VTN Coordinator(Troubleshooting HowTo)

Overview

This page demonstrates Installation troubleshooting steps of VTN Coordinator. OpenDaylight VTN provides multi-tenant virtual network functions on OpenDaylight controllers. OpenDaylight VTN consists of two parts:

- VTN Coordinator.
- VTN Manager.

VTN Coordinator orchestrates multiple VTN Managers running in OpenDaylight Controllers, and provides VTN Applications with VTN API. VTN Manager is OSGi bundles running in OpenDaylight Controller. Current VTN Manager supports only OpenFlow switches. It handles PACKET_IN messages, sends PACKET_OUT messages, manages host information, and installs flow entries into OpenFlow switches to provide VTN Coordinator with virtual network functions. The requirements for installing these two are different.Therefore, we recommend that you install VTN Manager and VTN Coordinator in different machines.

List of installation Troubleshooting How to's

- https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_(VTN):Installation:VTN_Coordinator

After executing db_setup, you have encountered the error "Failed to setup database"?

The error could be due to the below reasons

- Access Restriction

The user who owns /usr/local/vtn/ directory and installs VTN Coordinator, can only start db_setup. Example :

```
The directory should appear as below (assuming the user as "vtn"):
# ls -l /usr/local/
  drwxr-xr-x. 12 vtn  vtn  4096 Mar 14 21:53 vtn
If the user doesnot own /usr/local/vtn/ then, please run the below command (assuming␣
↪the username as vtn),
            chown -R vtn:vtn /usr/local/vtn
```

- Postgres not Present

```
1. In case of Fedora/CentOS/RHEL, please check if /usr/pgsql/<version> directory is␣
↪present and also ensure the commands initdb, createdb,pg_ctl,psql are working. If,␣
↪not please re-install postgres packages
2. In case of Ubuntu, check if /usr/lib/postgres/<version> directory is present and␣
↪check for the commands as in the previous step.
```

- Not enough space to create tables

```
Please check df -k and ensure enough free space is available.
```

- If the above steps do not solve the problem, please refer to the log file for the exact problem

```
/usr/local/vtn/var/dbm/unc_setup_db.log for the exact error.
```

- list of VTN Coordinator processes
- Run the below command ensure the Coordinator daemons are running.

```
   Command:      /usr/local/vtn/bin/unc_dmctl status
   Name               Type            IPC Channel       PID
-----------        -----------       --------------     ------
    drvodcd          DRIVER            drvodcd           15972
    lgcnwd          LOGICAL            lgcnwd            16010
    phynwd          PHYSICAL           phynwd            15996
```

- Issue the curl command to fetch version and ensure the process is able to respond.

How to debug a startup failure?.

The following activities take place in order during startup

- Database server is started after setting virtual memory to required value,Any database startup errors will be reflected in any of the below logs.

```
/usr/local/vtn/var/dbm/unc_db_script.log.
/usr/local/vtn/var/db/pg_log/postgresql-*.log (the pattern will have the date)
```

- uncd daemon is kicked off, The daemon in turn kicks off the rest of the daemons.

```
Any  uncd startup failures will be reflected in /usr/local/vtn/var/uncd/uncd_start.
↪err.
```

After setting up the apache tomcat server, what are the aspects that should be checked.

Please check if catalina is running..

```
The command ps -ef | grep catalina | grep -v grep should list a catalina process
```

If you encounter an erroneous situation where the REST API is always failing..

```
Please ensure the firewall settings for port:8181 (Beryllium release) or port:8083␣
↪(Post Beryllium release) and enable the same.
```

How to debug a REST API returning a failure message?.

Please check the /usr/share/java/apache-tomcat-7.0.39/logs/core/core.log for failure details.

REST API for VTN configuration fails, how to debug?.

The default log level for all daemons is "INFO", to debug the situation TRACE or DEBUG logs may be needed. To increase the log level for individual daemons, please use the commands suggested below

```
/usr/local/vtn/bin/lgcnw_control loglevel trace -- upll daemon log
 /usr/local/vtn/bin/phynw_control loglevel trace -- uppl daemon log
 /usr/local/vtn/bin/unc_control loglevel trace -- uncd daemon log
 /usr/local/vtn/bin/drvodc_control loglevel trace -- Driver daemon log
```

After setting the log levels, the operation can be repeated and the log files can be referred for debugging.

Problems while Installing PostgreSQL due to openssl.

Errors may occur when trying to install postgreSQL rpms. Recently PostgreSQL has upgraded all their binaries to use the latest openssl versions with fix for http://en.wikipedia.org/wiki/Heartbleed Please upgrade the openssl package to the latest version and re-install. For RHEL 6.1/6.4 : If you have subscription, Please use the same and update the rpms. The details are available in the following link https://access.redhat.com/site/solutions/781793 ACCESS-REDHAT

```
rpm -Uvh http://mirrors.kernel.org/centos/6/os/x86_64/Packages/openssl-1.0.1e-15.el6.
↪x86_64.rpm
rpm -ivh http://mirrors.kernel.org/centos/6/os/x86_64/Packages/openssl-devel-1.0.1e-
↪15.el6.x86_64.rpm
```

For other linux platforms, Please do yum update, the public respositroes will have the latest openssl, please install the same.

Support for Microsoft SCVMM 2012 R2 with ODL VTN

Introduction

System Center Virtual Machine Manager (SCVMM) is Microsoft's virtual machine support center for window's based emulations. SCVMM is a management solution for the virtualized data center. You can use it to configure and manage your virtualization host, networking, and storage resources in order to create and deploy virtual machines and services to private clouds that you have created.

The VSEM Provider is a plug-in to bridge between SCVMM and OpenDaylight.

Microsoft Hyper-V is a server virtualization developed by Microsoft, which provides virtualization services through hypervisor-based emulations.

The topology used in this set-up is:

- A SCVMM with VSEM Provider installed and a running VTN Coordinator and OpenDaylight with VTN Feature installed.
- PF1000 virtual switch extension has been installed in the two Hyper-V servers as it implements the OpenFlow capability in Hyper-V.
- Three OpenFlow switches simulated using mininet and connected to Hyper-V.
- Four VM's hosted using SCVMM.

It is implemented as two major components:

- SCVMM
- OpenDaylight (VTN Feature)
- VTN Coordinator

VTN Coordinator

OpenDaylight VTN as Network Service provider for SCVMM where VSEM provider is added in the Network Service which will handle all requests from SCVMM and communicate with the VTN Coordinator. It is used to manage the network virtualization provided by OpenDaylight.

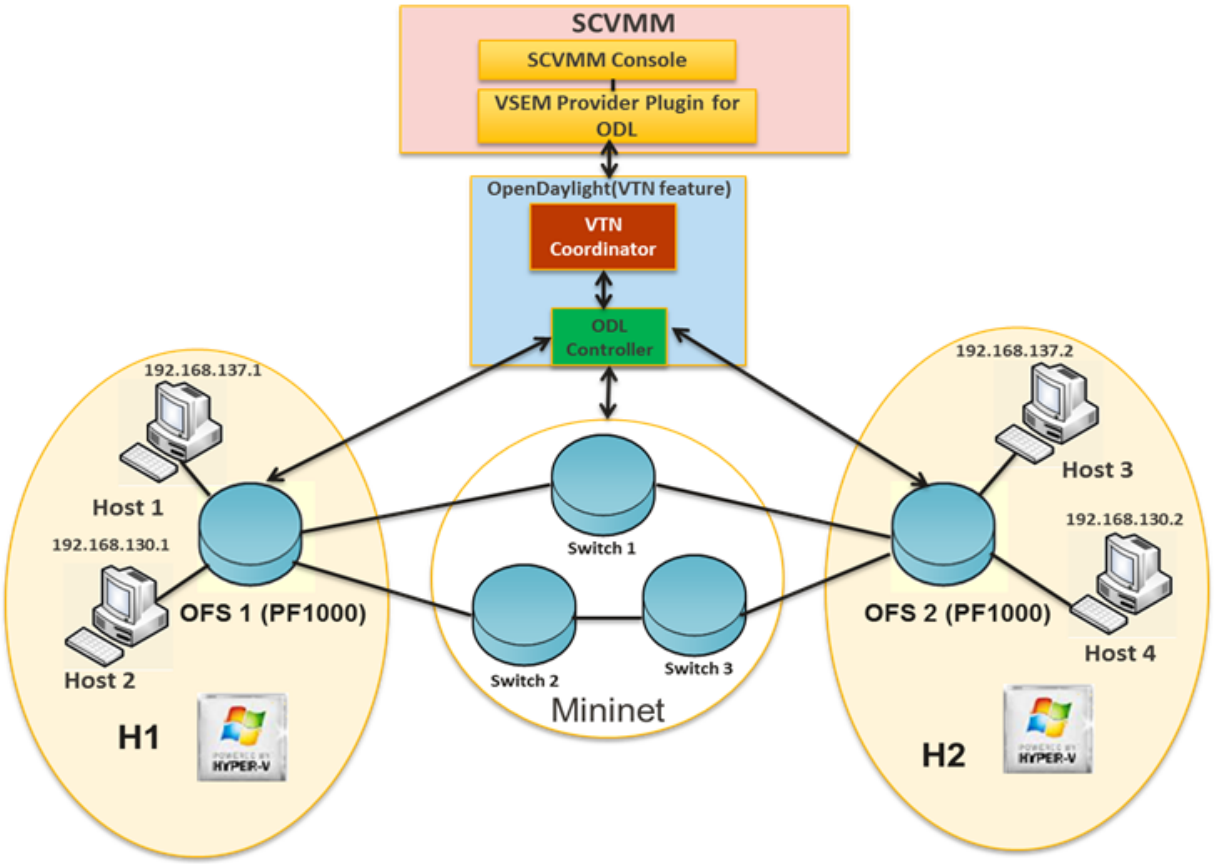


Fig. 1.152: Set-Up Diagram

Installing HTTPS in VTN Coordinator

- System Center Virtual Machine Manager (SCVMM) supports only https protocol.

Apache Portable Runtime (APR) Installation Steps

- Enter the command "yum install **apr**" in VTN Coordinator installed machine.
- In /usr/bin, create a soft link as "ln –s /usr/bin/apr-1-config /usr/bin/apr-config".
- Extract tomcat under "/usr/share/java" by using the below command "tar -xvf apache-tomcat-8.0.27.tar.gz –C /usr/share/java".

Note: Please go through the bleow link to download apache-tomcat-8.0.27.tar.gz file, https://archive.apache.org/dist/tomcat/tomcat-8/v8.0.27/bin/

- Please go to the directory "cd /usr/share/java/apache-tomcat-8.0.27/bin and unzip tomcat-native.gz using this command "tar -xvf tomcat-native.gz".
- Go to the directory "cd /usr/share/java/apache-tomcat-8.0.27/bin/tomcat-native-1.1.33-src/jni/native".
- Enter the command "./configure –with-os-type=bin –with-apr=/usr/bin/apr-config".
- Enter the command "make" and "make install".
- Apr libraries are successfully installed in "/usr/local/apr/lib".

Enable HTTP/HTTPS in VTN Coordinator

Enter the command "firewall-cmd –zone=public –add-port=8083/tcp –permanent" and "firewall-cmd –reload" to enable firewall settings in server.

Create a CA's private key and a self-signed certificate in server

- Execute the following command "openssl req -x509 -days 365 -extensions v3_ca -newkey rsa:2048 –out /etc/pki/CA/cacert.pem –keyout /etc/pki/CA/private/cakey.pem" in a single line.

Argument	Description
Country Name	Specify the country code. For example, JP
State or Province Name	Specify the state or province. For example, Tokyo
Locality Name	Locality Name For example, Chuo-Ku
Organization Name	Specify the company.
Organizational Unit Name	Specify the department, division, or the like.
Common Name	Specify the host name.
Email Address	Specify the e-mail address.

- Execute the following commands: "touch /etc/pki/CA/index.txt" and "echo 00 > /etc/pki/CA/serial" in server after setting your CA's private key.

Create a private key and a CSR for web server

- Execute the following command "openssl req -new -newkey rsa:2048 -out csr.pem –keyout /usr/local/vtn/tomcat/conf/key.pem" in a single line.
- Enter the PEM pass phrase: Same password you have given in CA's private key PEM pass phrase.

Argument	Description
Country Name	Specify the country code. For example, JP
State or Province Name	Specify the state or province. For example, Tokyo
Locality Name	Locality Name For example, Chuo-Ku
Organization Name	Specify the company.
Organizational Unit Name	Specify the department, division, or the like.
Common Name	Specify the host name.
Email Address	Specify the e-mail address.
A challenge password	Specify the challenge password.
An optional company name	Specify an optional company name.

Create a certificate for web server

- Execute the following command "openssl ca –in csr.pem –out /usr/local/vtn/tomcat/conf/cert.pem –days 365 –batch" in a single line.
- Enter pass phrase for /etc/pki/CA/private/cakey.pem: Same password you have given in CA's private key PEM pass phrase.
- Open the tomcat file using "vim /usr/local/vtn/tomcat/bin/tomcat".
- Include the line " TOMCAT_PROPS="$TOMCAT_PROPS -Djava.library.path=\"/usr/local/apr/lib\"" " in 131th line and save the file.

Edit server.xml file and restart the server

- Open the server.xml file using "vim /usr/local/vtn/tomcat/conf/server.xml" and add the below lines.

```
<Connector port="${vtn.port}" protocol="HTTP/1.1" SSLEnabled="true"
maxThreads="150" scheme="https" secure="true"
SSLCertificateFile="/usr/local/vtn/tomcat/conf/cert.pem"
SSLCertificateKeyFile="/usr/local/vtn/tomcat/conf/key.pem"
SSLPassword=same password you have given in CA's private key PEM pass phrase
connectionTimeout="20000" />
```

- Save the file and restart the server.
- To stop vtn use the following command.

```
/usr/local/vtn/bin/vtn_stop
```

- To start vtn use the following command.

```
/usr/local/vtn/bin/vtn_start
```

- Copy the created CA certificate from cacert.pem to cacert.crt by using the following command,

```
openssl x509 -in /etc/pki/CA/cacert.pem -out cacert.crt
```

Checking the HTTP and HTTPS connection from client

- You can check the HTTP connection by using the following command:

```
curl -X GET -H 'contenttype:application/json' -H 'username:admin' -H
↪'password:adminpass' http://<server IP address>:8083/vtn-webapi/api_version.json
```

- You can check the HTTPS connection by using the following command:

```
curl -X GET -H 'contenttype:application/json' -H 'username:admin' -H
↪'password:adminpass' https://<server IP address>:8083/vtn-webapi/api_version.
↪json --cacert /etc/pki/CA/cacert.pem
```

- The response should be like this for both HTTP and HTTPS:

```
{"api_version":{"version":"V1.4"}}
```

Prerequisites to create Network Service in SCVMM machine, Please follow the below steps

1. Please go through the below link to download VSEM Provider zip file, https://nexus.opendaylight.org/content/groups/public/org/opendaylight/vtn/application/vtnmanager-vsemprovider/1.2.0-Boron/vtnmanager-vsemprovider-1.2.0-Boron-bin.zip
2. Unzip the vtnmanager-vsemprovider-1.2.0-Boron-bin.zip file anywhere in your SCVMM machine.
3. Stop SCVMM service from **"service manager→tools→servers→select system center virtual machine manager"** and click stop.
4. Go to **"C:/Program Files"** in your SCVMM machine. Inside **"C:/Program Files"**, create a folder named as **"ODLProvider"**.
5. Inside **"C:/Program Files/ODLProvider"**, create a folder named as "Module" in your SCVMM machine.
6. Inside "C:/Program Files/ODLProvider/Module", Create two folders named as **"Odl.VSEMProvider"** and **"VSEMOdlUI"** in your SCVMM machine.
7. Copy the **"VSEMOdl.dll"** file from **"ODL_SCVMM_PROVIDER/ODL_VSEM_PROVIDER"** to **"C:/Program Files/ODLProvider/Module/Odl.VSEMProvider"** in your SCVMM machine.
8. Copy the **"VSEMOdlProvider.psd1"** file from **"application/vsemprovider/VSEMOdlProvider/VSEMOdlProvider.psd1"** to **"C:/Program Files/ODLProvider/Module/Odl.VSEMProvider"** in your SCVMM machine.
9. Copy the **"VSEMOdlUI.dll"** file from **"ODL_SCVMM_PROVIDER/ODL_VSEM_PROVIDER_UI"** to **"C:/Program Files/ODLProvider/Module/VSEMOdlUI"** in your SCVMM machine.
10. Copy the **"VSEMOdlUI.psd1"** file from **"application/vsemprovider/VSEMOdlUI"** to **"C:/Program Files/ODLProvider/Module/VSEMOdlUI"** in your SCVMM machine.
11. Copy the **"reg_entry.reg"** file from **"ODL_SCVMM_PROVIDER/Register_settings"** to your SCVMM desktop and double click the **"reg_entry.reg"** file to install registry entry in your SCVMM machine.
12. Download **"PF1000.msi"** from this link, https://www.pf-info.com/License/en/index.php?url=index/index_non_buyer and place into **"C:/Program Files/Switch Extension Drivers"** in your SCVMM machine.

13. Start SCVMM service from **"service manager→tools→servers→select system center virtual machine manager"** and click start.

System Center Virtual Machine Manager (SCVMM)

It supports two major features:

- Failover Clustering
- Live Migration

Failover Clustering

A single Hyper-V can host a number of virtual machines. If the host were to fail then all of the virtual machines that are running on it will also fail, thereby resulting in a major outage. Failover clustering treats individual virtual machines as clustered resources. If a host were to fail then clustered virtual machines are able to fail over to a different Hyper-V server where they can continue to run.

Live Migration

Live Migration is used to migrate the running virtual machines from one Hyper-V server to another Hyper-V server without any interruptions. Please go through the below video for more details,

- https://youtu.be/34YMOTzbNJM

SCVMM User Guide

- Please go through the below link for SCVMM user guide: https://wiki.opendaylight.org/images/c/ca/ODL_SCVMM_USER_GUIDE_final.pdf
- Please go through the below links for more details
  - OpenDaylight SCVMM VTN Integration: https://youtu.be/iRt4dxtiz94
  - OpenDaylight Congestion Control with SCVMM VTN: https://youtu.be/34YMOTzbNJM

1.4 OpenDaylight with Openstack Guide

1.4.1 Overview

OpenStack is a popular open source Infrastructure as a service project, covering compute, storage and network management. OpenStack can use OpenDaylight as its network management provider through the Modular Layer 2 (ML2) north-bound plug-in. OpenDaylight manages the network flows for the OpenStack compute nodes via the OVSDB south-bound plug-in. This page describes how to set that up, and how to tell when everything is working.

1.4.2 Installing OpenStack

Installing OpenStack is out of scope for this document, but to get started, it is useful to have a minimal multi-node OpenStack deployment.

The reference deployment we will use for this document is a 3 node cluster:

- One control node containing all of the management services for OpenStack (Nova, Neutron, Glance, Swift, Cinder, Keystone)
- Two compute nodes running nova-compute
- Neutron using the OVS back-end and vxlan for tunnels

Once you have installed OpenStack, verify that it is working by connecting to Horizon and performing a few operations. To check the Neutron configuration, create two instances on a private subnet bridging to your public network, and verify that you can connect to them, and that they can see each other.

1.4.3 Installing OpenDaylight

OpenStack with NetVirt

OpenStack with NetVirt

Table of Contents

Prerequisites: OpenDaylight requires Java 1.8.0 and Open vSwitch >= 2.5.0

Installing OpenDaylight on an existing OpenStack

- On the control host, Download the latest OpenDaylight release
- Uncompress it as root, and start OpenDaylight (you can start OpenDaylight by running karaf directly, but exiting from the shell will shut it down):

```
tar xvfz distribution-karaf-0.5.1-Boron-SR1.tar.gz
cd distribution-karaf-0.5.1-Boron-SR1
./bin/start # Start OpenDaylight as a server process
```

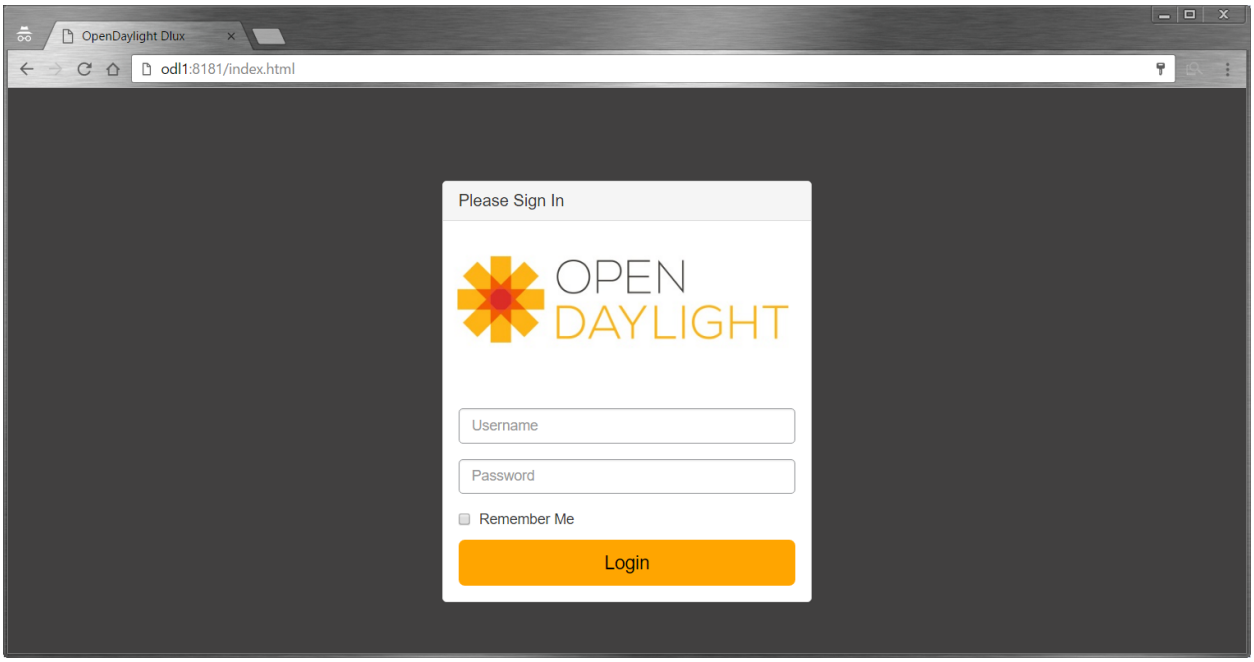
- Connect to the Karaf shell, and install the odl-netvirt-openstack bundle, dlux and their dependencies:

```
./bin/client # Connect to OpenDaylight with the client
opendaylight-user@root> feature:install odl-netvirt-openstack odl-dlux-core odl-
↪mdsal-apidocs
```

- If everything is installed correctly, you should now be able to log in to the dlux interface on http://CONTROL_HOST:8181/index.html - the default username and password is "admin/admin" (see screenshot below)

Optional - Advanced OpenDaylight Installation - Configurations and Clustering

- ACL Implementation - Security Groups - Stateful:



  - Default implementation used is stateful, requiring OVS compiled with conntrack modules.
  - This requires using a linux kernel that is >= 4.3
  - To check if OVS is running with conntrack support:

```
root@devstack:~/# lsmod | grep conntrack | grep openvswitch
  nf_conntrack          106496  9 xt_CT,openvswitch,nf_nat,nf_nat_ipv4,xt_
↪conntrack,nf_conntrack_netlink,xt_connmark,nf_conntrack_ipv4,nf_conntrack_
↪ipv6
```

  - If the conntrack modules are not installed for OVS, either recompile/install an OVS version with conntrack support, or alternatively configure OpenDaylight to use a non-stateful implementation.
  - OpenvSwitch 2.5 with conntrack support can be acquired from this repository for yum based linux distributions:

```
yum install -y http://rdoproject.org/repos/openstack-newton/rdo-release-
↪newton.rpm
yum install -y --nogpgcheck openvswitch
```

- ACL Implementations - Alternative options:
  - "learn" - semi-stateful implementation that does not require conntrack support. This is the most complete non-conntrack implementation.
  - "stateless" - naive security group implementation for TCP connections only. UDP and ICMP packets are allowed by default.
  - "transparent" - no security group support. all traffic is allowed, this is the recommended mode if you don't need to use security groups at all.
  - To configure one of these alternative implementations, the following needs to be done prior to running OpenDaylight:

```
mkdir -p <ODL_FOLDER>/etc/opendaylight/datastore/initial/config/
export CONFFILE=\`find <ODL_FOLDER> -name "\*aclservice\*config.xml"\`
cp \CONFFILE <ODL_FOLDER>/etc/opendaylight/datastore/initial/config/netvirt-
↪aclservice-config.xml
sed -i s/stateful/<learn/transparent>/ <ODL_FOLDER>/etc/opendaylight/
↪datastore/initial/config/netvirt-aclservice-config.xml
cat <ODL_FOLDER>/etc/opendaylight/datastore/initial/config/netvirt-aclservice-
↪config.xml
```

- Running multiple OpenDaylight controllers in a cluster:
  - For redundancy, it is possible to run OpenDaylight in a 3-node cluster.
  - More info on Clustering available here.
  - To configure OpenDaylight in clustered mode, run <ODL_FOLDER>/bin/configure_cluster.sh on each node prior to running OpenDaylight. This script is used to configure cluster parameters on this controller. The user should restart controller to apply changes.

```
Usage: ./configure_cluster.sh <index> <seed_nodes_list>
- index: Integer within 1..N, where N is the number of seed nodes.
- seed_nodes_list: List of seed nodes, separated by comma or space.
```

  - The address at the provided index should belong this controller. When running this script on multiple seed nodes, keep the seed_node_list same, and vary the index from 1 through N.
  - Optionally, shards can be configured in a more granular way by modifying the file "custom_shard_configs.txt" in the same folder as this tool. Please see that file for more details.

Note: OpenDaylight should be restarted after applying any of the above changes via configuration files.

Ensuring OpenStack network state is clean

When using OpenDaylight as the Neutron back-end, OpenDaylight expects to be the only source of truth for Neutron configurations. Because of this, it is necessary to remove existing OpenStack configurations to give OpenDaylight a clean slate.

- Delete instances:

```
nova list
nova delete <instance names>
```

- Remove links from subnets to routers:

```
neutron subnet-list
neutron router-list
neutron router-port-list <router name>
neutron router-interface-delete <router name> <subnet ID or name>
```

- Delete subnets, networks, routers:

```
neutron subnet-delete <subnet name>
neutron net-list
neutron net-delete <net name>
neutron router-delete <router name>
```

- Check that all ports have been cleared - at this point, this should be an empty list:

```
neutron port-list
```

Ensure Neutron is stopped

While Neutron is managing the OVS instances on compute and control nodes, OpenDaylight and Neutron can be in conflict. To prevent issues, we turn off Neutron server on the network controller, and Neutron's Open vSwitch agents on all hosts.

- Turn off neutron-server on control node:

```
systemctl stop neutron-server
systemctl stop neutron-l3-agent
```

- On each node in the cluster, shut down and disable Neutron's agent services to ensure that they do not restart after a reboot:

```
systemctl stop neutron-openvswitch-agent
systemctl disable
neutron-openvswitch-agent
systemctl stop neutron-l3-agent
systemctl disable neutron-l3-agent
```

Configuring Open vSwitch to be managed by OpenDaylight

On each host (both compute and control nodes) we will clear the pre-existing Open vSwitch config and set OpenDaylight to manage the switch:

- Stop the Open vSwitch service, and clear existing OVSDB (OpenDaylight expects to manage vSwitches completely):

```
systemctl stop openvswitch
rm -rf /var/log/openvswitch/*
rm -rf /etc/openvswitch/conf.db
systemctl start openvswitch
```

- At this stage, your Open vSwitch configuration should be empty:

```
[root@odl-compute2 ~]# ovs-vsctl show
9f3b38cb-eefc-4bc7-828b-084b1f66fbfd
    ovs_version: "2.5.1"
```

- Set OpenDaylight as the manager on all nodes:

```
ovs-vsctl set-manager tcp:{CONTROL_HOST}:6640
```

- Set the IP to be used for VXLAN connectivity on all nodes. This IP must correspond to an actual linux interface on each machine.

```
sudo ovs-vsctl set Open_vSwitch . other_config:local_ip=<ip>
```

- You should now see a new section in your Open vSwitch configuration showing that you are connected to the OpenDaylight server via OVSDB, and OpenDaylight will automatically create a br-int bridge that is connected via OpenFlow to the controller:

```
[root@odl-compute2 ~]# ovs-vsctl show
9f3b38cb-eefc-4bc7-828b-084b1f66fbfd
     Manager "tcp:172.16.21.56:6640"
         is_connected: true
     Bridge br-int
         Controller "tcp:172.16.21.56:6633"
             is_connected: true
         fail_mode: secure
         Port br-int
             Interface br-int
     ovs_version: "2.5.1"

 [root@odl-compute2 ~]# ovs-vsctl get Open_vSwitch . other_config
 {local_ip="10.0.42.161"}
```

- If you do not see the result above (specifically, if you do not see "is_connected: true" in the Manager section or in the Controller section), you may not have a security policies in place to allow Open vSwitch remote administration.

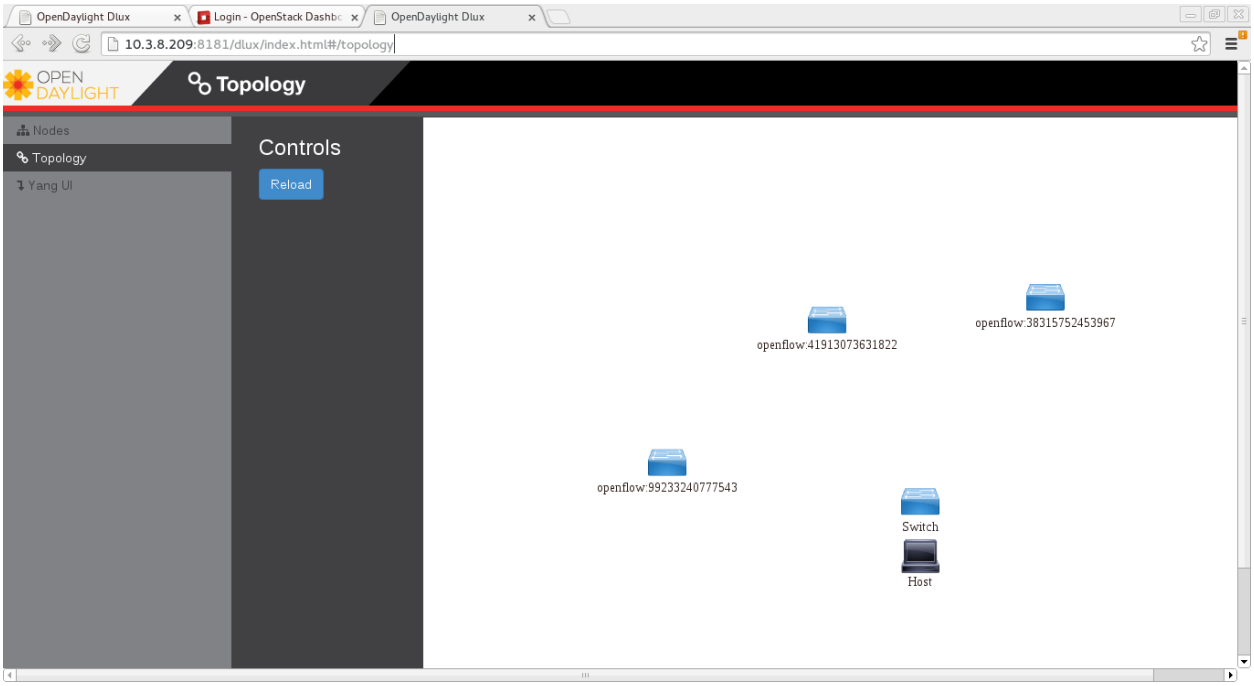
Note:

There might be iptables restrictions - if so the relevant ports should be opened (6640, 6653).

If SELinux is running on your linux, set to permissive mode on all nodes and ensure it stays that way after boot.

```
setenforce 0
sed -i -e 's/SELINUX=enforcing/SELINUX=permissive/g' /etc/selinux/config
```

- Make sure all nodes, including the control node, are connected to OpenDaylight.
- If you reload DLUX, you should now see that all of your Open vSwitch nodes are now connected to OpenDaylight.



- If something has gone wrong, check `data/log/karaf.log` under the OpenDaylight distribution directory. If you do not see any interesting log entries, set logging for netvirt to TRACE level inside Karaf and try again:

```
log:set TRACE netvirt
```

Configuring Neutron to use OpenDaylight

Once you have configured the vSwitches to connect to OpenDaylight, you can now ensure that OpenStack Neutron is using OpenDaylight.

This requires the neutron networking-odl module to be installed. | `pip install networking-odl`

First, ensure that port 8080 (which will be used by OpenDaylight to listen for REST calls) is available. By default, swift-proxy-service listens on the same port, and you may need to move it (to another port or another host), or disable that service. It can be moved to a different port (e.g. 8081) by editing `/etc/swift/proxy-server.conf` and `/etc/cinder/cinder.conf`, modifying iptables appropriately, and restarting swift-proxy-service. Alternatively, OpenDaylight can be configured to listen on a different port, by modifying the `jetty.port` property value in `etc/jetty.conf`.

```
<Set name="port">
    <Property name="jetty.port" default="8080" />
</Set>
```

- Configure Neutron to use OpenDaylight's ML2 driver:

```
crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 mechanism_drivers␣
↪opendaylight
crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 tenant_network_types vxlan

cat <<EOT>> /etc/neutron/plugins/ml2/ml2_conf.ini
[ml2_odl]
url = http://{CONTROL_HOST}:8080/controller/nb/v2/neutron
password = admin
username = admin
EOT
```

- Configure Neutron to use OpenDaylight's odl-router service plugin for L3 connectivity:

```
crudini --set /etc/neutron/plugins/neutron.conf DEFAULT service_plugins odl-router
```

- Configure Neutron DHCP agent to provide metadata services:

```
crudini --set /etc/neutron/plugins/dhcp_agent.ini DEFAULT force_metadata True
```

Note:

If the OpenStack version being used is Newton, this workaround should be applied,
configuring the Neutron DHCP agent to use vsctl as the OVSDB interface:

```
crudini --set /etc/neutron/plugins/dhcp_agent.ini OVS ovsdb_interface vsctl
```

- Reset Neutron's database

```
mysql -e "DROP DATABASE IF EXISTS neutron;"
mysql -e "CREATE DATABASE neutron CHARACTER SET utf8;"
/usr/local/bin/neutron-db-manage --config-file /etc/neutron/neutron.conf --config-
↪file /etc/neutron/plugins/ml2/ml2_conf.ini upgrade head
```

- Restart neutron-server:

```
systemctl start neutron-server
```

Verifying it works

- Verify that OpenDaylight's ML2 interface is working:

```
curl -u admin:admin http://{CONTROL_HOST}:8080/controller/nb/v2/neutron/networks

{
   "networks" : [ ]
}
```

If this does not work or gives an error, check Neutron's log file in `/var/log/neutron/server.log`. Error messages here should give some clue as to what the problem is in the connection with OpenDaylight.

- Create a network, subnet, router, connect ports, and start an instance using the Neutron CLI:

```
neutron router-create router1
neutron net-create private
neutron subnet-create private --name=private_subnet 10.10.5.0/24
neutron router-interface-add router1 private_subnet
nova boot --flavor <flavor> --image <image id> --nic net-id=<network id> test1
nova boot --flavor <flavor> --image <image id> --nic net-id=<network id> test2
```

At this point, you have confirmed that OpenDaylight is creating network end-points for instances on your network and managing traffic to them.

VMs can be reached using Horizon console, or alternatively by issuing `nova get-vnc-console <vm> novnc` Through the console, connectivity between VMs can be verified.

Adding an external network for floating IP connectivity

- In order to connect to the VM using a floating IP, we need to configure external network connectivity, by creating an external network and subnet. This external network must be linked to a physical port on the machine, which will provide connectivity to an external gateway.

```
sudo ovs-vsctl set Open_vSwitch . other_config:provider_mappings=physnet1:eth1
neutron net-create public-net -- --router:external --is-default --
↪provider:network_type=flat --provider:physical_network=physnet1
neutron subnet-create --allocation-pool start=10.10.10.2,end=10.10.10.254 --
↪gateway 10.10.10.1 --name public-subnet public-net 10.10.0.0/16 -- --enable_
↪dhcp=False
```

```
neutron router-gateway-set router1 public-net

neutron floatingip-create public-net
nova floating-ip-associate test1 <floating_ip>
```

Installing OpenStack and OpenDaylight using DevStack

The easiest way to load and OpenStack setup using OpenDaylight is by using devstack, which does all the steps mentioned in previous sections. | `git clone https://git.openstack.org/openstack-dev/devstack`

- The following lines need to be added to your local.conf:

```
enable_plugin networking-odl http://git.openstack.org/openstack/networking-odl
 ↪<branch>
ODL_MODE=allinone
Q_ML2_PLUGIN_MECHANISM_DRIVERS=opendaylight,logger
ODL_GATE_SERVICE_PROVIDER=vpnservice
disable_service q-l3
ML2_L3_PLUGIN=odl-router
ODL_PROVIDER_MAPPINGS={PUBLIC_PHYSICAL_NETWORK}:<external linux interface>
```

- More details on using devstack can be found in the following links:
  - Devstack All-In-One Single Machine Tutorial
  - Devstack networking-odl README

Troubleshooting

VM DHCP Issues

- Trigger DHCP requests - access VM console:
  - View log: `nova console-log <vm>`
  - Access using VNC console: nova get-vnc-console <vm> novnc
  - Trigger DHCP requests: `sudo ifdown eth0 ; sudo ifup eth0`

```
udhcpc (v1.20.1) started
Sending discover...
Sending select for 10.0.123.3...
Lease of 10.0.123.3 obtained, lease time 86400 # This only happens when DHCP␣
 ↪is properly obtained.
```

- Check if the DHCP requests are reaching the qdhcp agent using the following commands on the OpenStack controller:

```
sudo ip netns
sudo ip netns exec qdhcp-xxxxx ifconfig # xxxx is the neutron network id
sudo ip netns exec qdhcp-xxxxx tcpdump -nei tapxxxxx # xxxxx is the neutron port␣
 ↪id

# Valid request and response:
```

```
15:08:41.684932 fa:16:3e:02:14:bb > ff:ff:ff:ff:ff:ff, ethertype IPv4 (0x0800),␣
↪length 329: 0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from␣
↪fa:16:3e:02:14:bb, length 287
15:08:41.685152 fa:16:3e:79:07:98 > fa:16:3e:02:14:bb, ethertype IPv4 (0x0800),␣
↪length 354: 10.0.123.2.67 > 10.0.123.3.68: BOOTP/DHCP, Reply, length 312
```

- If the requests aren't reaching qdhcp:
  - Verify VXLAN tunnels exist between compute and control nodes by using `ovs-vsctl show`
  - Run the following commands to debug the OVS processing of the DHCP request packet:
    `ovs-ofctl -OOpenFlow13 dump-ports-desc br-int` # retrieve VMs ofport and MAC
    `ovs-appctl ofproto/trace br-int in_port=<ofport>,dl_src=<mac>, dl_dst=ff:ff:ff:ff:ff:ff,udp,ip_src=0.0.0.0,ip_dst=255.255.255.255 | grep "Rule\|action"`

```
root@devstack:~# ovs-appctl ofproto/trace br-int in_port=1,dl_
↪src=fe:16:3e:33:8b:d8,dl_dst=ff:ff:ff:ff:ff:ff,udp,ip_src=0.0.0.0,ip_
↪dst=255.255.255.255 | grep "Rule\|action"
    Rule: table=0 cookie=0x8000000 priority=1,in_port=1
    OpenFlow actions=write_metadata:0x20000000001/0xffffff0000000001,goto_
↪table:17
        Rule: table=17 cookie=0x8000001 priority=5,metadata=0x20000000000/
↪0xffffff0000000000
        OpenFlow actions=write_metadata:0xc0000200000222e2/0xfffffffffffffffe,
↪goto_table:19
            Rule: table=19 cookie=0x1080000 priority=0
            OpenFlow actions=resubmit(,17)
                Rule: table=17 cookie=0x8040000 priority=6,
↪metadata=0xc000020000000000/0xffffff0000000000
                OpenFlow actions=write_metadata:0xe00002138a000000/
↪0xfffffffffffffffe,goto_table:50
                    Rule: table=50 cookie=0x8050000 priority=0
                    OpenFlow actions=CONTROLLER:65535,goto_table:51
                        Rule: table=51 cookie=0x8030000 priority=0
                        OpenFlow actions=goto_table:52
                            Rule: table=52 cookie=0x870138a priority=5,
↪metadata=0x138a000001/0xffff000001
                            OpenFlow actions=write_actions(group:210003)
    Datapath actions: drop

root@devstack:~# ovs-ofctl -OOpenFlow13 dump-groups br-int | grep 'group_
↪id=210003'
    group_id=210003,type=all
```

- If the requests are reaching qdhcp, but the response isn't arriving to the VM:
  - Locate the compute the VM is residing on (can use `nova show <vm>`).
    - If the VM is on the same node as the qdhcp namespace, `ofproto/trace` can be used to track the packet:
      `ovs-appctl ofproto/trace br-int in_port=<dhcp_ofport>,dl_src=<dhcp_port_mac>,dl_dst=<vm_port_mac>, udp,ip_src=<dhcp_port_ip>,ip_dst=<vm_port_ip> | grep "Rule\|action"`

```
root@devstack:~# ovs-appctl ofproto/trace br-int in_port=2,dl_
↪src=fa:16:3e:79:07:98,dl_dst=fa:16:3e:02:14:bb,udp,ip_src=10.0.123.2,ip_
↪dst=10.0.123.3 | grep "Rule\|action"
```

```
    Rule: table=0 cookie=0x8000000 priority=4,in_port=2
    OpenFlow actions=write_metadata:0x10000000000/0xffffff0000000001,goto_
↪table:17
        Rule: table=17 cookie=0x8000001 priority=5,metadata=0x10000000000/
↪0xffffff0000000000
        OpenFlow actions=write_metadata:0x60000100000222e0/
↪0xfffffffffffffffe,goto_table:19
            Rule: table=19 cookie=0x1080000 priority=0
            OpenFlow actions=resubmit(,17)
                Rule: table=17 cookie=0x8040000 priority=6,
↪metadata=0x6000010000000000/0xffffff0000000000
                OpenFlow actions=write_metadata:0x7000011389000000/
↪0xfffffffffffffffe,goto_table:50
                    Rule: table=50 cookie=0x8051389 priority=20,
↪metadata=0x11389000000/0xffffffffff000000,dl_src=fa:16:3e:79:07:98
                    OpenFlow actions=goto_table:51
                        Rule: table=51 cookie=0x8031389 priority=20,
↪metadata=0x1389000000/0xffff000000,dl_dst=fa:16:3e:02:14:bb
                        OpenFlow actions=load:0x300->NXM_NX_REG6[],
↪resubmit(,220)
                            Rule: table=220 cookie=0x8000007 priority=7,
↪reg6=0x300
                            OpenFlow actions=output:3
```

* If the VM isn't on the same node as the qdhcp namepsace:
  - Check if the packet is arriving via VXLAN by running `tcpdump -nei <vxlan_port> port 4789`
  - If it is arriving via VXLAN, the packet can be tracked on the compute node rules, using `ofproto/trace` in a similiar manner to the previous section. Note that packets arriving from a tunnels have a unique tunnel_id (VNI) that should be used as well in the trace, due to the special processing of packets arriving from a VXLAN tunnel.

Floating IP Issues

- If you have assigned an external network and associated a floating IP to a VM but there is still no connectivity:
  - Verify the external gateway IP is reachable through the provided provider network port.
  - Verify OpenDaylight has successfully resolved the MAC address of the external gateway IP. This can be verified by searching for the line "Installing ext-net group" in the karaf.log.
  - Locate the compute the VM is residing on (can use `nova show <vm>`).
  - Run a ping to the VM floating IP.
  - If the ping fails, execute a flow dump of br-int, and search for the flows that are relevant to the VM's floating IP address: `ovs-ofctl -OOpenFlow13 dump-flows br-int | grep "<floating_ip>"`
    * Are there packets on the incoming flow (matching dst_ip=<floating_ip>)?
      If not this probably means the provider network has not been set up properly, verify provider_mappings configuration and the configured external network physical_network value match. Also verify that the Flat/VLAN network configured is actually reachable via the configured port.
    * Are there packets on the outgoing flow (matching src_ip=<floating_ip>)?

If not, this probably means that OpenDaylight is failing to resolve the MAC of the provided external gateway, required for forwarding packets to the external network.

* Are there packets being sent on the external network port?
  This can be checked using `tcpdump <port>` or by viewing the appropriate OpenFlow rules. The mapping between the OpenFlow port number and the linux interface can be acquired using `ovs-ofctl dump-ports-desc br-int`

```
ovs-ofctl -OOpenFlow13 dump-flows br-int | grep "<floating_ip>"
cookie=0x8000003, duration=436.710s, table=21, n_packets=190, n_
 ↪bytes=22602, priority=42,ip,metadata=0x222e2/0xfffffffe,nw_dst=10.64.98.
 ↪17 actions=goto_table:25
cookie=0x8000004, duration=436.739s, table=25, n_packets=190, n_
 ↪bytes=22602, priority=10,ip,nw_dst=10.64.98.17 actions=set_field:10.0.
 ↪123.3->ip_dst,write_metadata:0x222e0/0xfffffffe,goto_table:27
cookie=0x8000004, duration=436.730s, table=26, n_packets=120, n_
 ↪bytes=15960, priority=10,ip,metadata=0x222e0/0xfffffffe,nw_src=10.0.123.
 ↪3 actions=set_field:10.64.98.17->ip_src,write_metadata:0x222e2/
 ↪0xfffffffe,goto_table:28
cookie=0x8000004, duration=436.728s, table=28, n_packets=120, n_
 ↪bytes=15960, priority=10,ip,metadata=0x222e2/0xfffffffe,nw_src=10.64.98.
 ↪17 actions=set_field:fa:16:3e:ec:a8:84->eth_src,group:200000
```

Useful Links

- NetVirt Tables Pipeline
- NetVirt Wiki Page
- NetVirt Basic Tutorial (OpenDaylight Summit 2016)
- NetVirt Advanced Tutorial (OpenDaylight Summit 2016)
- Other OpenDaylight Documentation

OpenStack with GroupBasedPolicy

This section is for Application Developers and Network Administrators who are looking to integrate Group Based Policy with OpenStack.

To enable the **GBP** Neutron Mapper feature, at the karaf console:

```
feature:install odl-groupbasedpolicy-neutronmapper
```

Neutron Mapper has the following dependencies that are automatically loaded:

```
odl-neutron-service
```

Neutron Northbound implementing REST API used by OpenStack

```
odl-groupbasedpolicy-base
```

Base **GBP** feature set, such as policy resolution, data model etc.

```
odl-groupbasedpolicy-ofoverlay
```

For this release, **GBP** has one renderer, hence this is loaded by default.

REST calls from OpenStack Neutron are by the Neutron NorthBound project.

GBP provides the implementation of the Neutron V2.0 API.

Features

List of supported Neutron entities:

- Port
- Network
  - Standard Internal
  - External provider L2/L3 network
- Subnet
- Security-groups
- Routers
  - Distributed functionality with local routing per compute
  - External gateway access per compute node (dedicated port required)
  - Multiple routers per tenant
- FloatingIP NAT
- IPv4/IPv6 support

The mapping of Neutron entities to **GBP** entities is as follows:

Neutron Port

The Neutron port is mapped to an endpoint.

The current implementation supports one IP address per Neutron port.

An endpoint and L3-endpoint belong to multiple EndpointGroups if the Neutron port is in multiple Neutron Security Groups.

The key for endpoint is L2-bridge-domain obtained as the parent of L2-flood-domain representing Neutron network. The MAC address is from the Neutron port. An L3-endpoint is created based on L3-context (the parent of the L2-bridge-domain) and IP address of Neutron Port.

Neutron Network

A Neutron network has the following characteristics:

- defines a broadcast domain
- defines a L2 transmission domain
- defines a L2 name space.

To represent this, a Neutron Network is mapped to multiple **GBP** entities. The first mapping is to an L2 flood-domain to reflect that the Neutron network is one flooding or broadcast domain. An L2-bridge-domain is then associated as the parent of L2 flood-domain. This reflects both the L2 transmission domain as well as the L2 addressing namespace.

The third mapping is to L3-context, which represents the distinct L3 address space. The L3-context is the parent of L2-bridge-domain.

Neutron Subnet

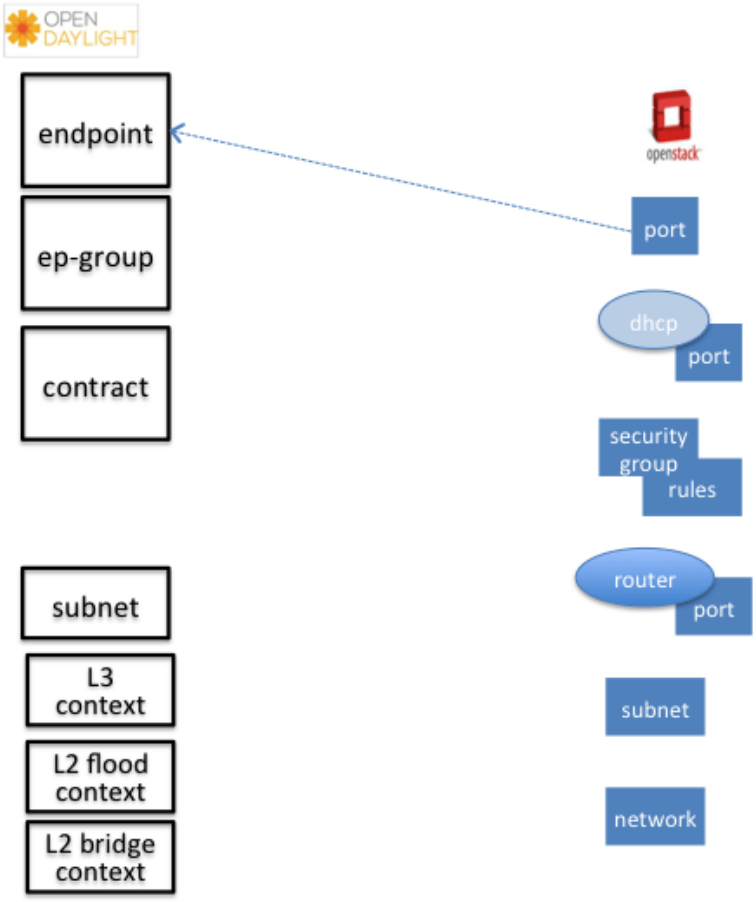


Fig. 1.153: Neutron Port

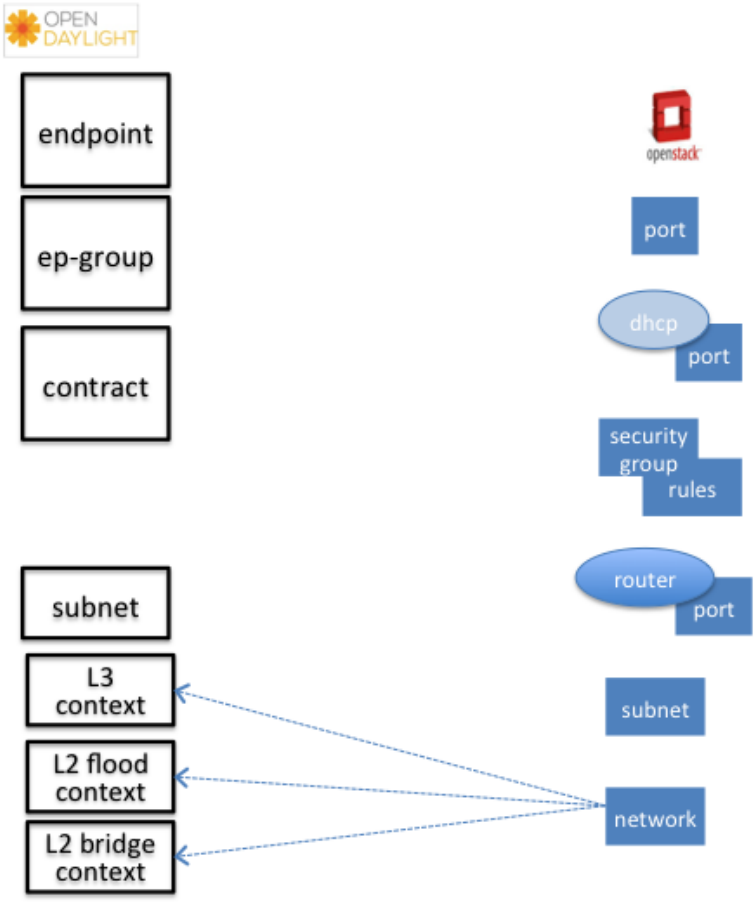


Fig. 1.154: Neutron Network

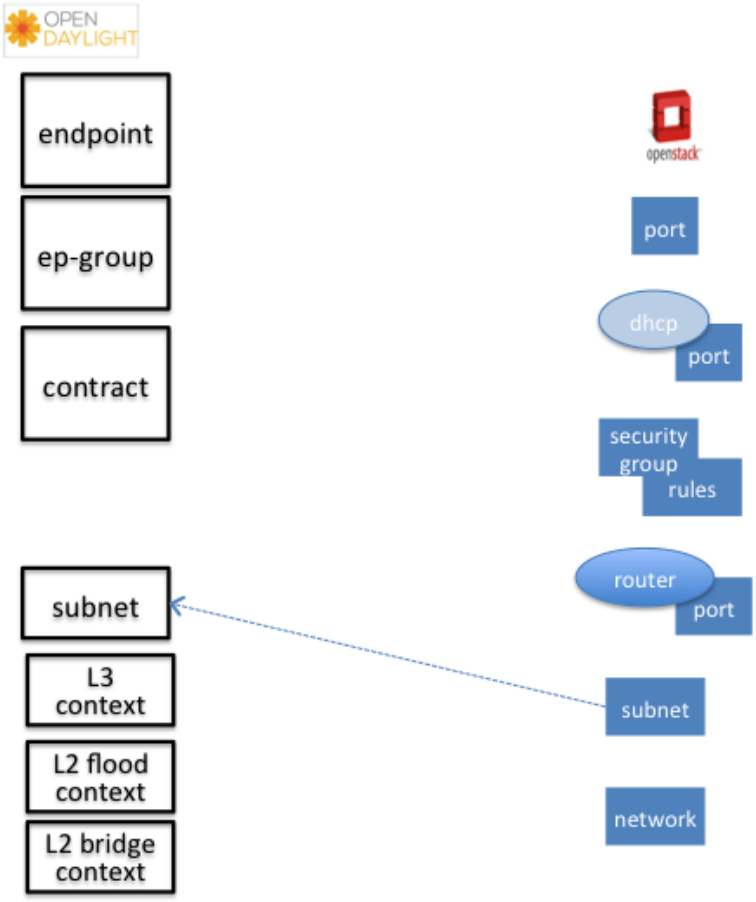


Fig. 1.155: Neutron Subnet

Neutron subnet is associated with a Neutron network. The Neutron subnet is mapped to a *GBP* subnet where the parent of the subnet is L2-flood-domain representing the Neutron network.

Neutron Security Group

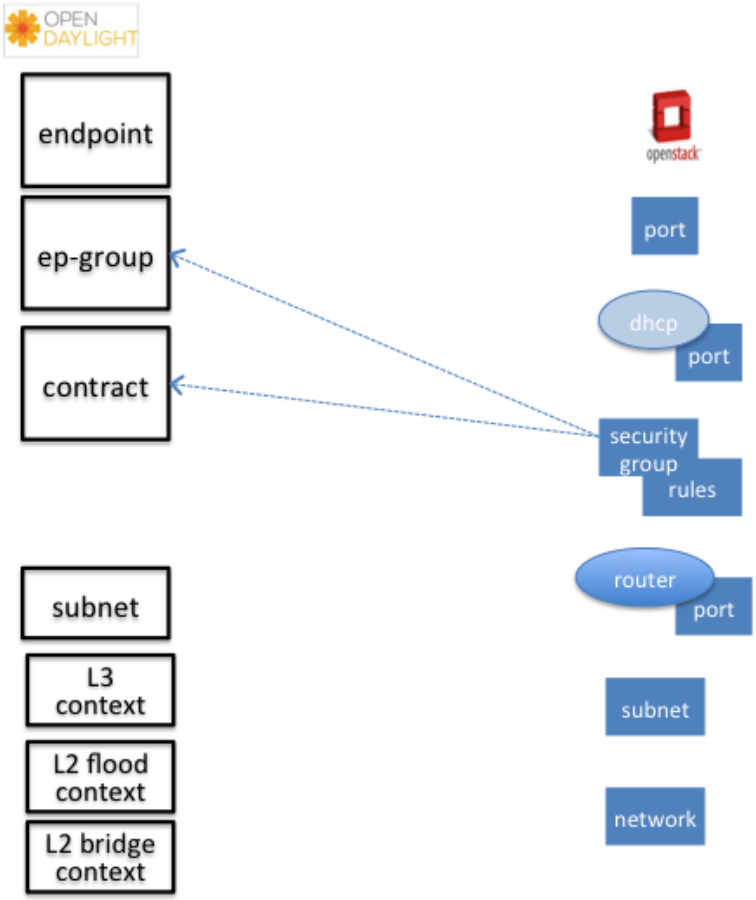


Fig. 1.156: Neutron Security Group and Rules

GBP entity representing Neutron security-group is EndpointGroup.

Infrastructure EndpointGroups

Neutron-mapper automatically creates EndpointGroups to manage key infrastructure items such as:

- DHCP EndpointGroup - contains endpoints representing Neutron DHCP ports
- Router EndpointGroup - contains endpoints representing Neutron router interfaces
- External EndpointGroup - holds L3-endpoints representing Neutron router gateway ports, also associated with FloatingIP ports.

Neutron Security Group Rules

This mapping is most complicated among all others because Neutron security-group-rules are mapped to contracts with clauses, subjects, rules, action-refs, classifier-refs, etc. Contracts are used between endpoint groups representing Neutron Security Groups. For simplification it is important to note that Neutron security-group-rules are similar to a **GBP** rule containing:

- classifier with direction

- action of *allow*.

Neutron Routers

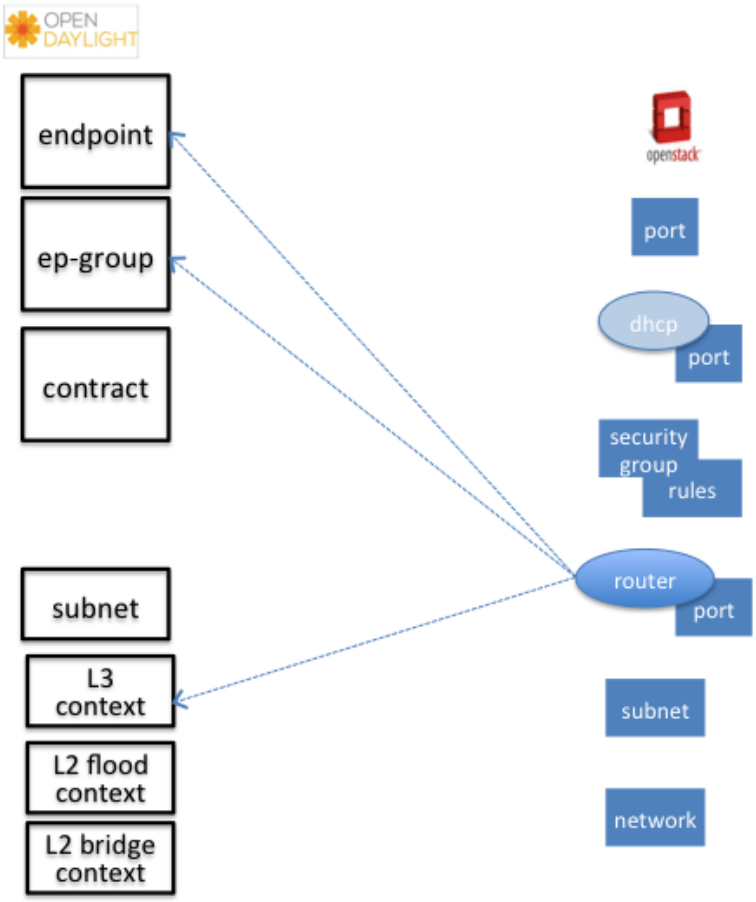


Fig. 1.157: Neutron Router

Neutron router is represented as a L3-context. This treats a router as a Layer3 namespace, and hence every network attached to it a part of that Layer3 namespace.

This allows for multiple routers per tenant with complete isolation.

The mapping of the router to an endpoint represents the router's interface or gateway port.

The mapping to an EndpointGroup represents the internal infrastructure EndpointGroups created by the **GBP** Neutron Mapper

When a Neutron router interface is attached to a network/subnet, that network/subnet and its associated endpoints or Neutron Ports are seamlessly added to the namespace.

Neutron FloatingIP

When associated with a Neutron Port, this leverages the *GBP* OfOverlay renderer's NAT capabilities.

A dedicated *external* interface on each Nova compute host allows for disitributed external access. Each Nova instance associated with a FloatingIP address can access the external network directly without having to route via the Neutron controller, or having to enable any form of Neutron distributed routing functionality.

Assuming the gateway provisioned in the Neutron Subnet command for the external network is reachable, the combination of *GBP* Neutron Mapper and OfOverlay renderer will automatically ARP for this default gateway, requiring no user intervention.

Troubleshooting within GBP

Logging level for the mapping functionality can be set for package org.opendaylight.groupbasedpolicy.neutron.mapper. An example of enabling TRACE logging level on karaf console:

```
log:set TRACE org.opendaylight.groupbasedpolicy.neutron.mapper
```

Neutron mapping example

As an example for mapping can be used creation of Neutron network, subnet and port. When a Neutron network is created 3 **GBP** entities are created: l2-flood-domain, l2-bridge-domain, l3-context.

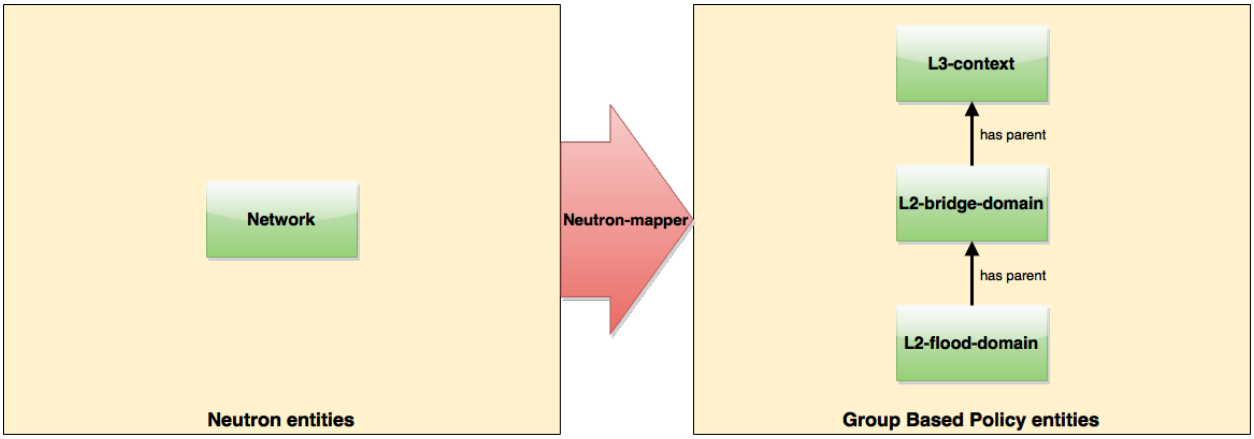


Fig. 1.158: Neutron network mapping

After an subnet is created in the network mapping looks like this.

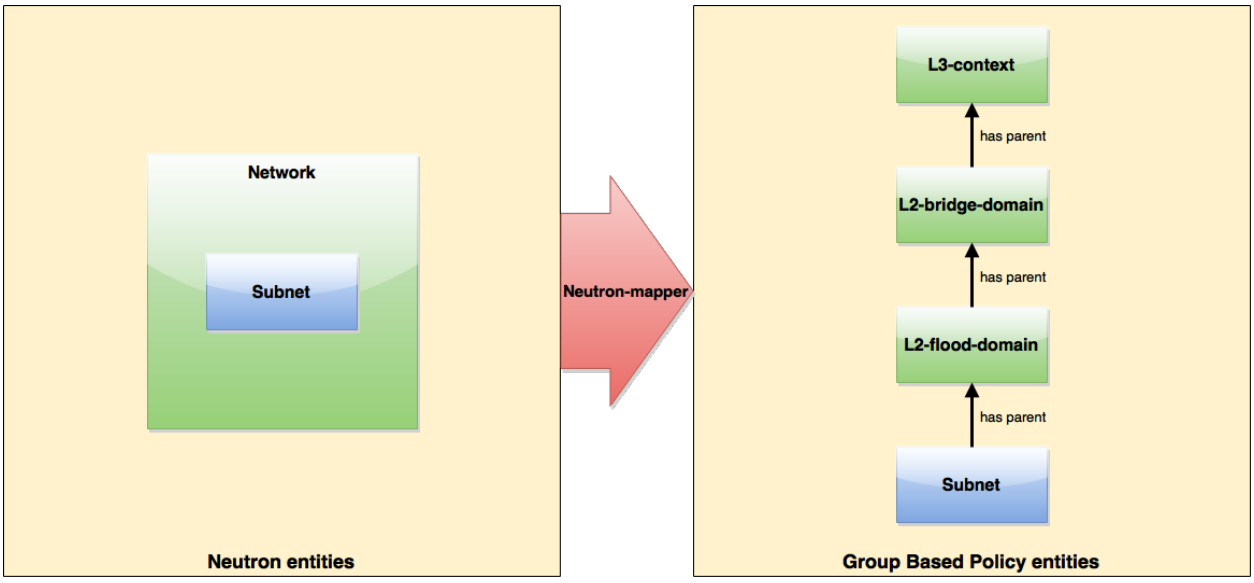


Fig. 1.159: Neutron subnet mapping

If an Neutron port is created in the subnet an endpoint and l3-endpoint are created. The endpoint has key composed from l2-bridge-domain and MAC address from Neutron port. A key of l3-endpoint is compesed from l3-context and IP address. The network containment of endpoint and l3-endpoint points to the subnet.

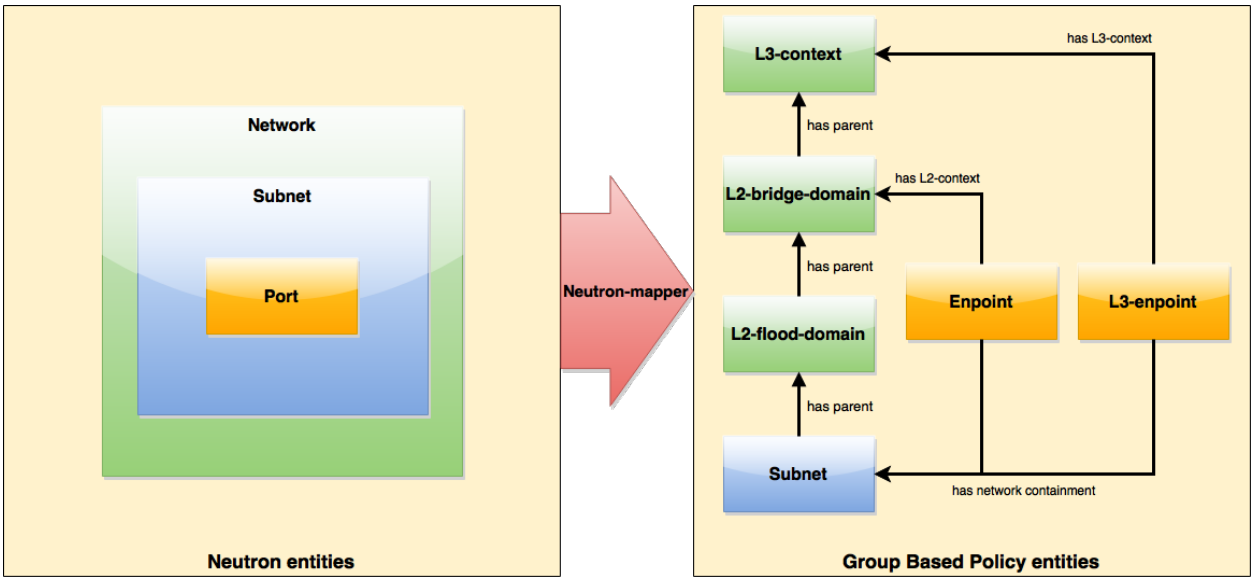


Fig. 1.160: Neutron port mapping

Configuring GBP Neutron

No intervention passed initial OpenStack setup is required by the user.

More information about configuration can be found in our DevStack demo environment on the GBP wiki.

Administering or Managing GBP Neutron

For consistencies sake, all provisioning should be performed via the Neutron API. (CLI or Horizon).

The mapped policies can be augmented via the **GBP** UX,UX, to:

- Enable Service Function Chaining
- Add endpoints from outside of Neutron i.e. VMs/containers not provisioned in OpenStack
- Augment policies/contracts derived from Security Group Rules
- Overlay additional contracts or groupings

Tutorials

A DevStack demo environment can be found on the GBP wiki.

Using Groupbasedpolicy's Neutron VPP Mapper

Overview

Neutron VPP Mapper implements features for support policy-based routing for OpenStack Neutron interface involving VPP devices. It allows using of policy-based schemes defined in GBP controller in a network consisting of OpenStack-provided nodes routed by a VPP node.

Architecture

Neutron VPP Mapper listens to Neutron data store change events, as well as being able to access directly the store. If the data changed match certain criteria (see *Processing Neutron Configuration*), Neutron VPP Mapper converts Neutron data specifically required to render a VPP node configuration with a given End Point, e.g., the virtual host interface name assigned to a `vhostuser` socket. Then the mapped data is stored in the VPP info data store.

Administering Neutron VPP Mapper

To use the Neutron VPP Mapper in Karaf, at least the following Karaf features must be installed:

- odl-groupbasedpolicy-neutron-vpp-mapper
- odl-vbd-ui

Initial pre-requisites

A topology should exist in config datastore, it is necessary to define a node with a particular `node-id`. Later, `node-id` will be used as a physical location reference in VPP renderer's bridge domain:

```
GET http://localhost:8181/restconf/config/network-topology:network-topology/

{
    "network-topology":{
       "topology":[
            {
                "topology-id":"datacentre",
                "node":[
                    {
                       "node-id":"dut2",
                       "vlan-tunnel:super-interface":"GigabitEthernet0/9/0",
                       "termination-point":[
                            {
                                "tp-id":"GigabitEthernet0/9/0",
                                "neutron-provider-topology:physical-interface":{
                                    "interface-name":"GigabitEthernet0/9/0"
                                }
                            }
                        ]
                    }
                ]
            }
        ]
    }
}
```

Processing Neutron Configuration

`NeutronListener` listens to the changes in Neutron datatree in config datastore. It filters the changes, processing only `network` and `port` entities.

For a `network` entity it is checked that it has `physical-network` parameter set (i.e., it is backed-up by a physical network), and that `network-type` is `vlan-network` or `"flat"`, and if this check has passed, a related bridge domain is created in VPP Renderer config datastore (`http://{{controller}}:{{port}}/restconf/config/vpp-renderer:config`), referenced to network by `vlan` field.

In case of `"vlan-network"`, the `vlan` field contains the same value as `neutron-provider-ext:segmentation-id` of network created by Neutron.

In case of `"flat"`, the VLAN specific parameters are not filled out.

Note: In case of VXLAN network (i.e. `network-type` is `"vxlan-network"`), no information is actually written into VPP Renderer datastore, as VXLAN is used for tenant-network (so no packets are going outside). Instead, VPP Renderer looks up GBP flood domains corresponding to existing VPP bridge domains trying to establish a VXLAN tunnel between them.

For a `port` entity it is checked that `vif-type` contains `"vhostuser"` substring, and that `device-owner` contains a specific substring, namely `"compute"`, `"router"` or `"dhcp"`.

In case of `"compute"` substring, a `vhost-user` is written to VPP Renderer config datastore.

In case of `"dhcp"` or `"router"`, a `tap` is written to VPP Renderer config datastore.

Input/output examples

OpenStack is creating network, and these data are being put into the data store:

```
PUT http://{{controller}}:{{port}}/restconf/config/neutron:neutron/networks

{
    "networks": {
        "network": [
            {
                "uuid": "43282482-a677-4102-87d6-90708f30a115",
                "tenant-id": "94836b88-0e56-4150-aaa7-60f1c2b67faa",
                "neutron-provider-ext:segmentation-id": "2016",
                "neutron-provider-ext:network-type": "neutron-networks:network-type-
↪vlan",
                "neutron-provider-ext:physical-network": "datacentre",
                "neutron-L3-ext:external": true,
                "name": "drexternal",
                "shared": false,
                "admin-state-up": true,
                "status": "ACTIVE"
            }
        ]
    }
}
```

Checking bridge domain in VPP Renderer config data store. Note that `physical-location-ref` is referring to `"dut2"`, paired by `neutron-provider-ext:physical-network` -> `topology-id`:

```
GET http://{{controller}}:{{port}}/restconf/config/vpp-renderer:config

{
  "config": {
    "bridge-domain": [
      {
        "id": "43282482-a677-4102-87d6-90708f30a115",
        "type": "vpp-renderer:vlan-network",
        "description": "drexternal",
        "vlan": 2016,
        "physical-location-ref": [
          {
            "node-id": "dut2",
            "interface": [
              "GigabitEthernet0/9/0"
            ]
          }
        ]
      }
    ]
  }
}
```

Port (compute):

```
PUT http://{{controller}}:{{port}}/restconf/config/neutron:neutron/ports

{
    "ports": {
        "port": [
            {
                "uuid": "3d5dff96-25f5-4d4b-aa11-dc03f7f8d8e0",
                "tenant-id": "94836b88-0e56-4150-aaa7-60f1c2b67faa",
                "device-id": "dhcp58155ae3-f2e7-51ca-9978-71c513ab02ee-a91437c0-8492-
↪47e2-b9d0-25c44aef6cda",
                "neutron-binding:vif-details": [
                    {
                        "details-key": "somekey"
                    }
                ],
                "neutron-binding:host-id": "devstack-control",
                "neutron-binding:vif-type": "vhostuser",
                "neutron-binding:vnic-type": "normal",
                "mac-address": "fa:16:3e:4a:9f:c0",
                "name": "",
                "network-id": "a91437c0-8492-47e2-b9d0-25c44aef6cda",
                "neutron-portsecurity:port-security-enabled": false,
                "device-owner": "network:compute",
                "fixed-ips": [
                    {
                        "subnet-id": "0a5834ed-ed31-4425-832d-e273cac26325",
                        "ip-address": "10.1.1.3"
                    }
                ],
                "admin-state-up": true
            }
        ]
    }
```

```
}

GET http://{{controller}}:{{port}}/restconf/config/vpp-renderer:config

{
  "config": {
    "vpp-endpoint": [
      {
        "context-type": "l2-l3-forwarding:l2-bridge-domain",
        "context-id": "a91437c0-8492-47e2-b9d0-25c44aef6cda",
        "address-type": "l2-l3-forwarding:mac-address-type",
        "address": "fa:16:3e:4a:9f:c0",
        "vpp-node-path": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='topology-netconf']/network-
↪topology:node[network-topology:node-id='devstack-control']",
        "vpp-interface-name": "neutron_port_3d5dff96-25f5-4d4b-aa11-dc03f7f8d8e0",
        "socket": "/tmp/socket_3d5dff96-25f5-4d4b-aa11-dc03f7f8d8e0",
        "description": "neutron port"
      }
    ]
  }
}
```

Port (dhcp):

```
PUT http://{{controller}}:{{port}}/restconf/config/neutron:neutron/ports

{
    "ports": {
        "port": [
            {
                "uuid": "3d5dff96-25f5-4d4b-aa11-dc03f7f8d8e0",
                "tenant-id": "94836b88-0e56-4150-aaa7-60f1c2b67faa",
                "device-id": "dhcp58155ae3-f2e7-51ca-9978-71c513ab02ee-a91437c0-8492-
↪47e2-b9d0-25c44aef6cda",
                "neutron-binding:vif-details": [
                    {
                        "details-key": "somekey"
                    }
                ],
                "neutron-binding:host-id": "devstack-control",
                "neutron-binding:vif-type": "vhostuser",
                "neutron-binding:vnic-type": "normal",
                "mac-address": "fa:16:3e:4a:9f:c0",
                "name": "",
                "network-id": "a91437c0-8492-47e2-b9d0-25c44aef6cda",
                "neutron-portsecurity:port-security-enabled": false,
                "device-owner": "network:dhcp",
                "fixed-ips": [
                    {
                        "subnet-id": "0a5834ed-ed31-4425-832d-e273cac26325",
                        "ip-address": "10.1.1.3"
                    }
                ],
                "admin-state-up": true
            }
        ]
    }
```

```
}

GET http://{{controller}}:{{port}}/restconf/config/vpp-renderer:config

{
  "config": {
    "vpp-endpoint": [
      {
        "context-type": "l2-l3-forwarding:l2-bridge-domain",
        "context-id": "a91437c0-8492-47e2-b9d0-25c44aef6cda",
        "address-type": "l2-l3-forwarding:mac-address-type",
        "address": "fa:16:3e:4a:9f:c0",
        "vpp-node-path": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='topology-netconf']/network-
↪topology:node[network-topology:node-id='devstack-control']",
        "vpp-interface-name": "neutron_port_3d5dff96-25f5-4d4b-aa11-dc03f7f8d8e0",
        "physical-address": "fa:16:3e:4a:9f:c0",
        "name": "tap3d5dff96-25",
        "description": "neutron port"
      }
    ]
  }
}
```

OpenStack with Virtual Tenant Network

This section describes using OpenDaylight with the VTN manager feature providing network service for OpenStack. VTN manager utilizes the OVSDB southbound service and Neutron for this implementation. The below diagram depicts the communication of OpenDaylight and two virtual networks connected by an OpenFlow switch using this implementation.

Configure OpenStack to work with OpenDaylight(VTN Feature) using PackStack

Prerequisites to install OpenStack using PackStack

- Fresh CentOS 7.1 minimal install
- Use the below commands to disable and remove Network Manager in CentOS 7.1,

```
systemctl stop NetworkManager
systemctl disable NetworkManager
```

- To make SELINUX as permissive, please open the file "/etc/sysconfig/selinux" and change it as "SELINUX=permissive".
- After making selinux as permissive, please restart the CentOS 7.1 machine.

Steps to install OpenStack PackStack in CentOS 7.1

- To install OpenStack juno, use the following command,

```
yum update -y
yum -y install https://repos.fedorapeople.org/repos/openstack/openstack-juno/rdo-
↪release-juno-1.noarch.rpm
```

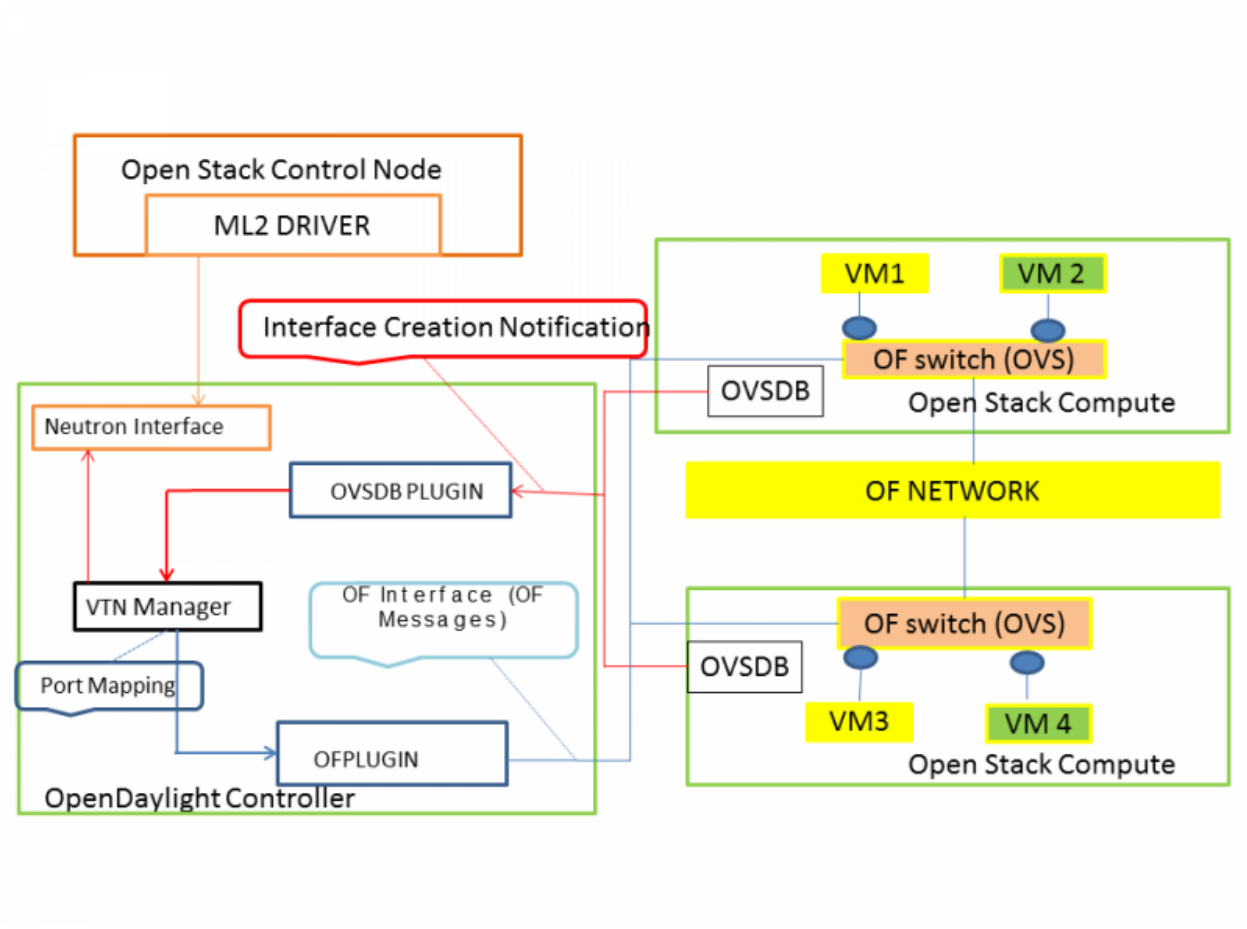


Fig. 1.161: OpenStack Architecture

```
```

- To install the packstack installer, please use the below command,

```
yum -y install openstack-packstack
```

- To create all-in-one setup, please use the below command,

```
packstack --allinone --provision-demo=n --provision-all-in-one-ovs-bridge=n
```

- This will end up with Horizon started successfully message.

Steps to install and deploy OpenDaylight in CentOS 7.1

- Download the latest Boron distribution code in the below link,

```
wget https://nexus.opendaylight.org/content/groups/public/org/opendaylight/
↪integration/distribution-karaf/0.5.0-Boron/distribution-karaf-0.5.0-Boron.zip
```

- Unzip the Boron distribution code by using the below command,

```
unzip distribution-karaf-0.5.0-Boron.zip
```

- Please do the below steps in the OpenDaylight to change jetty port,
  - Change the jetty port from 8080 to something else as swift proxy of OpenStack is using it.
  - Open the file "etc/jetty.xml" and change the jetty port from 8080 to 8910 (we have used 8910 as jetty port you can use any other number).
  - Start VTN Manager and install odl-vtn-manager-neutron in it.
  - Ensure all the required ports(6633/6653,6640 and 8910) are in the listen mode by using the command "netstat -tunpl" in OpenDaylight.

Steps to reconfigure OpenStack in CentOS 7.1

- Steps to stop Open vSwitch Agent and clean up ovs

```
sudo systemctl stop neutron-openvswitch-agent
sudo systemctl disable neutron-openvswitch-agent
sudo systemctl stop openvswitch
sudo rm -rf /var/log/openvswitch/*
sudo rm -rf /etc/openvswitch/conf.db
sudo systemctl start openvswitch
sudo ovs-vsctl show
```

- Stop Neutron Server

```
systemctl stop neutron-server
```

- Verify that OpenDaylight's ML2 interface is working:

```
curl -v admin:admin http://{CONTROL_HOST}:{PORT}/controller/nb/v2/neutron/networks

{
```

```
   "networks" : [ ]
}
```

If this does not work or gives an error, check Neutron's log file in */var/log/neutron/server.log*. Error messages here should give some clue as to what the problem is in the connection with OpenDaylight

- Configure Neutron to use OpenDaylight's ML2 driver:

```
sudo crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 mechanism_drivers␣
↪opendaylight
sudo crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 tenant_network_types␣
↪local
sudo crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 type_drivers local
sudo crudini --set /etc/neutron/dhcp_agent.ini DEFAULT ovs_use_veth True

cat <<EOT | sudo tee -a /etc/neutron/plugins/ml2/ml2_conf.ini > /dev/null
  [ml2_odl]
  password = admin
  username = admin
  url = http://{CONTROL_HOST}:{PORT}/controller/nb/v2/neutron
  EOT
```

- Reset Neutron's ML2 database

```
sudo mysql -e "drop database if exists neutron_ml2;"
sudo mysql -e "create database neutron_ml2 character set utf8;"
sudo mysql -e "grant all on neutron_ml2.* to 'neutron'@'%';"
sudo neutron-db-manage --config-file /usr/share/neutron/neutron-dist.conf --config-
↪file /etc/neutron/neutron.conf --config-file /etc/neutron/plugin.ini upgrade head
```

- Start Neutron Server

```
sudo systemctl start neutron-server
```

- Restart the Neutron DHCP service

```
system restart neutron-dhcp-agent.service
```

- At this stage, your Open vSwitch configuration should be empty:

```
[root@dneary-odl-compute2 ~]# ovs-vsctl show
686989e8-7113-4991-a066-1431e7277e1f
    ovs_version: "2.3.1"
```

- Set OpenDaylight as the manager on all nodes

```
ovs-vsctl set-manager  tcp:127.0.0.1:6640
```

- You should now see a section in your Open vSwitch configuration showing that you are connected to the OpenDaylight server, and OpenDaylight will automatically create a br-int bridge:

```
[root@dneary-odl-compute2 ~]# ovs-vsctl show
686989e8-7113-4991-a066-1431e7277e1f
    Manager "tcp:127.0.0.1:6640"
        is_connected: true
    Bridge br-int
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
```

```
        fail_mode: secure
        Port "ens33"
            Interface "ens33"
    ovs_version: "2.3.1"
```

- Add the default flow to OVS to forward packets to controller when there is a table-miss,

```
ovs-ofctl --protocols=OpenFlow13 add-flow br-int priority=0,actions=output:CONTROLLER
```

- Please see the VTN OpenStack PackStack support guide on the wiki to create VM's from OpenStack Horizon GUI.

Implementation details

VTN Manager

Install **odl-vtn-manager-neutron** feature which provides the integration with Neutron interface.

```
feature:install odl-vtn-manager-neutron
```

It subscribes to the events from Open vSwitch and also implements the Neutron requests received by OpenDaylight.

Functional Behavior

StartUp

- The ML2 implementation for OpenDaylight will ensure that when Open vSwitch is started, the ODL_IP_ADDRESS configured will be set as manager.
- When OpenDaylight receives the update of the Open vSwitch on port 6640 (manager port), VTN Manager handles the event and adds a bridge with required port mappings to the Open vSwitch at the OpenStack node.
- When Neutron starts up, a new network create is POSTed to OpenDaylight, for which VTN Manager creates a Virtual Tenant Network.
- *Network and Sub-Network Create:* Whenever a new sub network is created, VTN Manager will handle the same and create a vbridge under the VTN.
- *VM Creation in OpenStack:* The interface mentioned as integration bridge in the configuration file will be added with more interfaces on creation of a new VM in OpenStack and the network is provisioned for it by the VTN Neutron feature. The addition of a new port is captured by the VTN Manager and it creates a vbridge interface with port mapping for the particular port. When the VM starts to communicate with other VMs, the VTN Manger will install flows in the Open vSwitch and other OpenFlow switches to facilitate communication between them.

Note: To use this feature, VTN feature should be installed

Reference

https://wiki.opendaylight.org/images/5/5c/Integration_of_vtn_and_ovsdb_for_helium.pdf

CHAPTER 2

Content for OpenDaylight Developers

The Following content is intended for developers building applications or code on top of OpenDaylight, but who do not plan to modify OpenDaylight code itself.

2.1 Developer Guide

2.1.1 Overview

Gerrit Guide

Overview of Git and Gerrit

Git is an opensource distributed version control system (dvcs) written in the C language and originally developed by Linus Torvalds and others to manage the Linux kernel. In Git, there is no central copy of the repository. After you have cloned the repository, you have a functioning copy of the source code with all the branches and tagged releases, in your local repository.

Gerrit is an opensource web-based collaborative code review tool that integrates with Git. It was developed at Google by Shawn Pearce. Gerrit provides a framework for reviewing code commits before they are accepted into the code base. Changes can be uploaded to Gerrit by any user. However, the changes are not made a part of the project until a code review is completed. Gerrit is also a good collaboration tool for storing the conversations that occur around the code commits.

The OpenDaylight source code is hosted in a repository in Git. Developers must use Gerrit to commit code to the OpenDaylight repository.

Note: For more information on Git, see http://git-scm.com/. For more information on Gerrit, see https://code.google.com/p/gerrit/.

Prerequisites

Before you get started, you should have:

- an OpenDaylight account (sign up here) See *Creating an OpenDaylight Account* below for detailed instructions.
- git installed (see: http://www.git-scm.com/downloads)
- git configured with your name, e-mail address and editor

```
git config --global user.name "Firstname Lastname"
git config --global user.email "email@address.com"
git config --global core.editor "text-editor-name"
```

Note: Your name and e-mail address (including capitalization) must match what you entered when creating your OpenDaylight account.

- an ssh public/private key pair (see the good Github docs on generating ssh keys)
  - that is registered the OpenDaylight Gerrit server. See *Registering your SSH key with Gerrit* below for detailed instructions.
- git-review installed (see: https://www.mediawiki.org/wiki/Gerrit/git-review#Installation)

Setting up Each Git Repository

To clone a new repository:

```
git clone https://git.opendaylight.org/gerrit/${project_short_name}
```

For example to clone the Documentation repository:

```
git clone https://git.opendaylight.org/gerrit/docs
```

Common Gerrit Tasks

The following sections describe the most common Gerrit tasks you will need to complete while working with code in OpenDaylight.

Submitting a New Patch

1. On your machine, open a shell and switch to the directory with the git repository. Assuming you are using the docs repository:

   ```
   cd docs
   ```

2. To remove any dependencies on other files you are working on, check out the appropriate branch:

   ```
   git checkout ${remote_branch_name}  # will switch to the branch
   ```

Note: normally, `${remote_branch_name}` should be master, but during a release, the master will switch to `stable/${release_name}`, e.g., `stable/carbon` at some point. Also, if you are updating an existing release check out that branch.

Note: If you see an error like‘‘error: pathspec ‘stable/carbon’ did not match any file(s) known to git.‘‘, try this command instead:

```
git checkout -b ${remote_branch_name} origin/${remote_branch_name}
```

Note: This should only be necessary once.

3. Get a copy of the latest files from the server:

```
git pull                                          # will get all the changes from the␣
↪server
git reset --hard origin/${remote_branch_name} # (optional) will undo any local␣
↪changes you've
                                                  # (accidentally) made to ${remote_
↪branch_name}
```

4. Create a new branch for your work:

```
git checkout -b ${local_branch_name}
```

Note: Spaces are not allowed in `${local_branch_name}`.

5. Create new files or edit existing files, as needed.
6. Commit the files you have worked on:
   - If you've created any new files, run:

     ```
     git add ${filename}
     ```

   - To commit existing files you edited, run:
     - `git commit -as`
     - Your default terminal text editor will open.

     Note: The -as options instruct git to commit all of the files you have edited (`-a`) and sign your commit request with your email address and name (`-s`). The sign-off is to indicate that you agree with this statement:

     ```
     Developer's Certificate of Origin 1.1

        By making a contribution to this project, I certify that:

        (a) The contribution was created in whole or in part by me and I
            have the right to submit it under the open source license
            indicated in the file; or
     ```

```
(b) The contribution is based upon previous work that, to the best
    of my knowledge, is covered under an appropriate open source
    license and I have the right under that license to submit that
    work with modifications, whether created in whole or in part
    by me, under the same open source license (unless I am
    permitted to submit under a different license), as indicated
    in the file; or

(c) The contribution was provided directly to me by some other
    person who certified (a), (b) or (c) and I have not modified
    it.

(d) I understand and agree that this project and the contribution
    are public and that a record of the contribution (including all
    personal information I submit with it, including my sign-off) is
    maintained indefinitely and may be redistributed consistent with
    this project or the open source license(s) involved.
```

- Add a brief description of the changes you have made to the beginning of the commit request and then save the request.

Note: Writing good git commit messages is important and relatively easy, but it's good to familiarize yourself with general guidelines like this How to Write a Git Commit Message Guide.

Important: We have linters that check for some of the style guides including that the first line of a git commit message is 50 characters or less. So, make sure to follow that one.

7. Submit your files for review:
   - `git review`
   - You will receive 2 emails from Gerrit Code Review: The first indicating that a build to incorporate your changes has started; and the second indicating whether the build was created successfully.
8. Determine your patch's change number:
   - Open either of the emails you received after submitting your files for review.
   - Locate the following line in the terminal: `To view, visit <patch URL>`

Note: The number at the end of this URL is your patch's change number. You will need this in order to make updates to the patch later.

Updating an Existing Patch

1. On your machine, open a shell and switch to the directory containing the repository: `cd ${repository_name}`, e.g., `cd docs`
2. Download the patch you want to update: `git review -d ${change_number}`
3. (Optional) View information on the latest changes made to that patch:

To view the files that were edited, run `git show`
To view a listing of the files that were edited and the number of lines in those files that were edited, run `git show --stat`

4. Make the necessary changes to the patch's files.
5. Commit your changes:
   (a) run `git commit -a --amend`
   (b) Update the current patch description and then save the commit request.

> **Note:** If you feel as though you did enough additional work on the patch that you should be mentioned, add your name in the footer with a line like `Co-Authored-By: First Last <email>`.

6. Submit your files for review:
   `git review`

You will receive 2 emails from Gerrit Code Review: the first indicating that a build to incorporate your changes has started; and the second indicating whether the build was created successfully.

Code Review

All contributions to OpenDaylight Git repositories use Gerrit for code review.

The code review process is meant to provide constructive feedback about a proposed change. Committers and interested contributors will review the change, give their feedback, propose revisions and work with the change author through iterations of the patch until it's ready to be merged.

Feedback is provided and the change is managed via the Gerrit web UI.

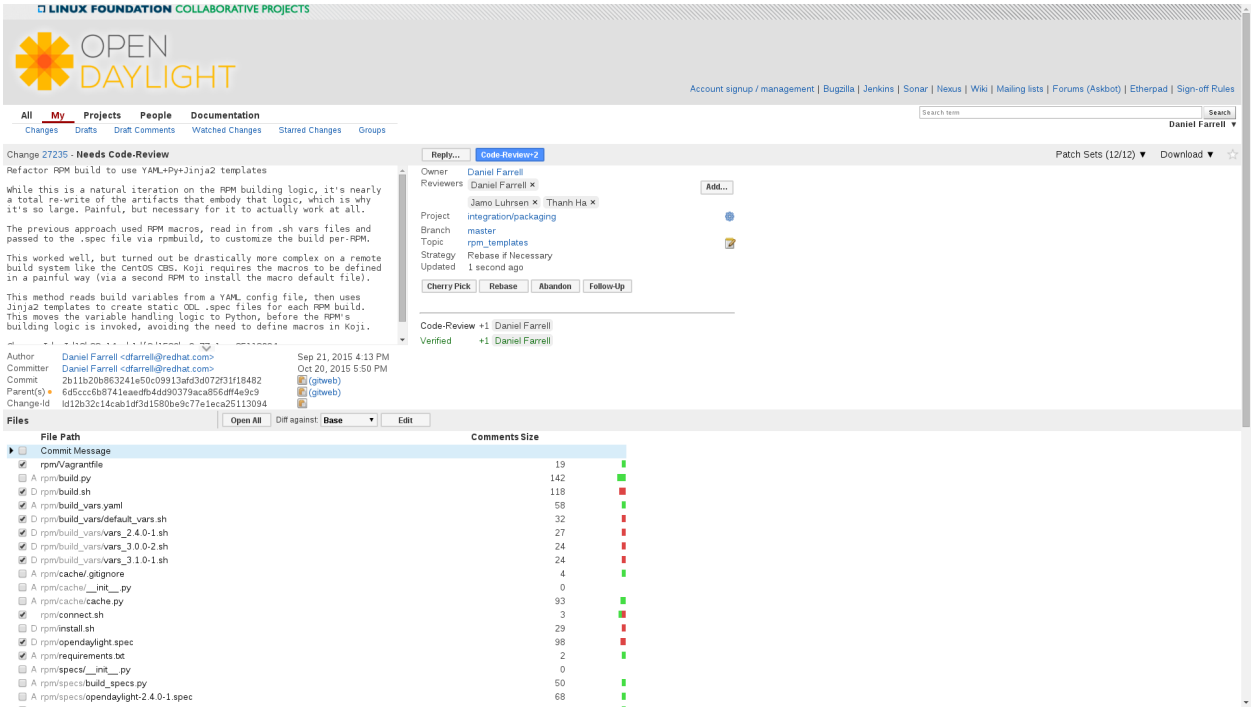


Fig. 2.1: Wide view of a change via the Gerrit web UI

Pre-review

Many times, change authors will want to push changes to Gerrit before they are actually ready for review. This is a good practice and is encouraged. It has been the experience of Integration/* so far that pushing early and often tends to reduce the amount of work overall.

Note: This is not required and in some projects, not encouraged, but the general idea of making sure patches are ready for review when submitted is a good one.

Note: While in draft state, Gerrit triggers, e.g., verify Jenkins jobs, won't run by default. You can trigger them despite it being a draft by adding `jenkins-releng` as a reviewer. You may need to do a recheck by replying with a comment containing recheck to trigger the jobs after adding the reviewer.

To mark an uploaded change as not ready for attention by committers and interested contributors (in order of preference), either mark the Gerrit a draft (by adding a `-D` to your `git review` command), vote -1 on it yourself or modify the commit message with "WIP" ("Work in Progress").

Don't add committers to the Reviewers list for a change while it's in the pre-review state, as it adds noise to their review queue.

Review

Once an author wants a change to be reviewed, they need to take some actions to put it on the radar of the committers.

If the change is marked as a draft, you'll need to publish it. This can be done from the Gerrit web UI.

Fig. 2.2: Gerrit Web UI button to publish a draft change.

Remove your -1 vote if you've marked it with one. If you think the patch is ready to be merged, vote +1. If there isn't an automated job to test your change and vote +1/-1 for Verified, you'll need to do as much testing yourself as possible and then manually vote +1 to Verified. You can also vote +1 for Verified if you've done testing in addition to any automated tests. Describing the testing you did or didn't do is typically helpful.

Once the change is published and you've voted for it to be merged, add the people who need to review/merge the change to the Gerrit Reviewers list. For Integration/Packaging, add all of our committers (Daniel Farrell, Jamo Luhrsen, Thanh Ha) in addition to any other relevant contributors. The auto-complete for this Gerrit UI field is somewhat flaky, but typing the full name from the start typically works.

Reviewers will give feedback via Gerrit comments or inline against the diff.

Updated versions of the proposed change should be pushed as new patchesets to the same Gerrit, either by the original submitter or other contributors. Amending proposed changes owned by others while reviewing may be more efficient than documenting the problem, -1ing, waiting for the original submitter to make the changes, re-reviewing and merging.

Changes can be downloaded for local manipulation and then re-uploaded with updates via git-review. See *Updating an Existing Patch* above. Once you have re-uploaded the patch the Gerrit web UI for the proposed change will reflect the new patchset.

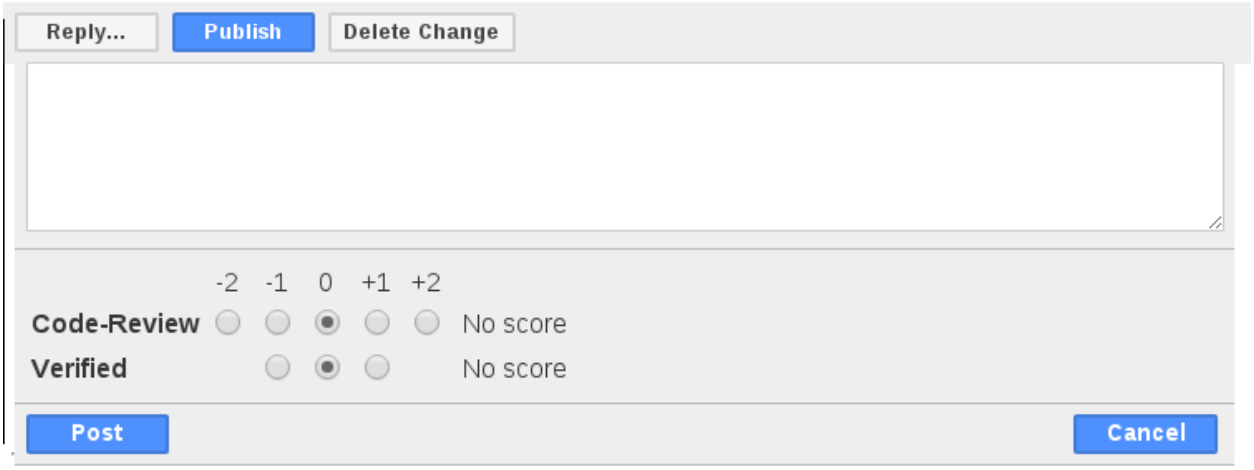


Fig. 2.3: Gerrit voting interface, exposed by the Reply button.

Fig. 2.4: Gerrit Reviewers list with Int/Pack committers added

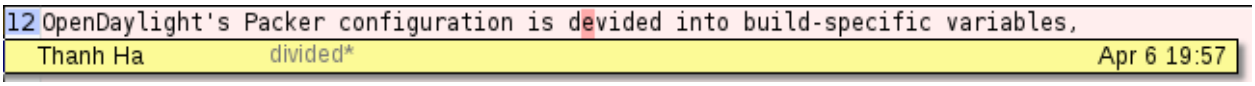


Fig. 2.5: Gerrit inline feedback about a typo

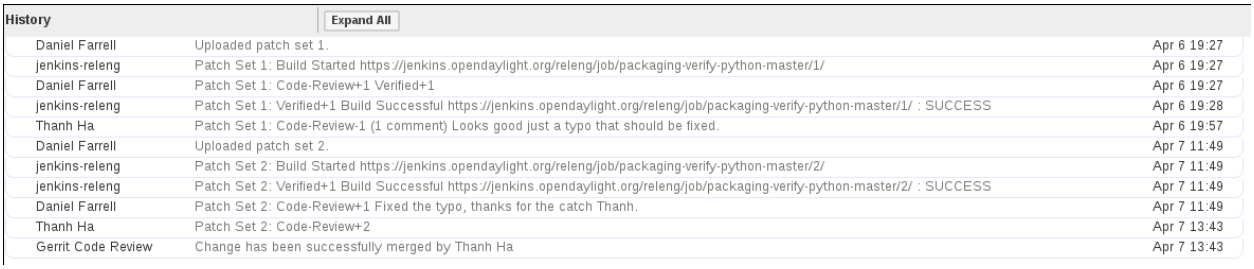


Fig. 2.6: Gerrit history showing a patch update

Reviewers will use the diff between the last time time they gave review and the current patchset to quickly understand updates, speeding the code review process.

Fig. 2.7: Gerrit diff menu

Iterative feedback continues until consensus is reached (typically: all active reviewers +1/+2 and no -1s, definitely no -2s), at least one committer +2s and a committer merges the change.

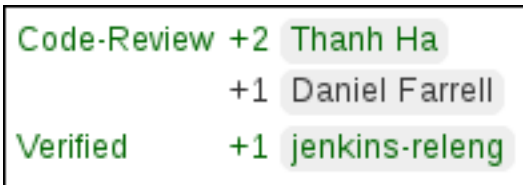


Fig. 2.8: Gerrit code review votes

Merge

Once a patch has gotten a +2 from a committer and they have clicked the submit button the project's merge job should run and publish the project's artifacts to Nexus. Once this is done other projects will be able to see the results of that patch.

This is particularly important when merging dependent patches across multiple projects. You will need to wait for the merge job to run on one patch before any patches in other projects depending on it will successful verify.

Setting up Gerrit

Creating an OpenDaylight Account

1. Using a Google Chrome or Mozilla Firefox browser, go to https://git.opendaylight.org/gerrit

   The main page shows existing Gerrit requests. These are patches that have been pushed to the repository and not yet verified, reviewed, and merged.

   Note: If you already have an OpenDaylight account, you can click **Sign In** in the top right corner of the page and follow the instructions to enter the OpenDaylight page.

   ---

2. If you do not have an existing OpenDaylight account, click **Account signup/management** on the top bar of the main Gerrit page.

   The **WS02 Identity Server** page is displayed.

3. In the **WS02 Identity Server** page, click **Sign-up** in the left pane.

   There is also an option to authenticate your sign in with OpenID. This option is not described in this document.

4. Click on the **Sign-up with User Name/Password** image on the right pane to continue to the actual sign-up page.

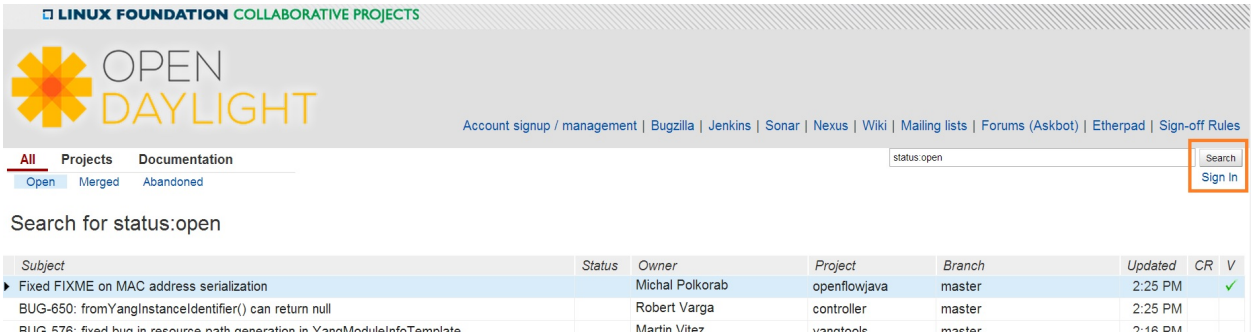


Fig. 2.9: Signing in to OpenDaylight account

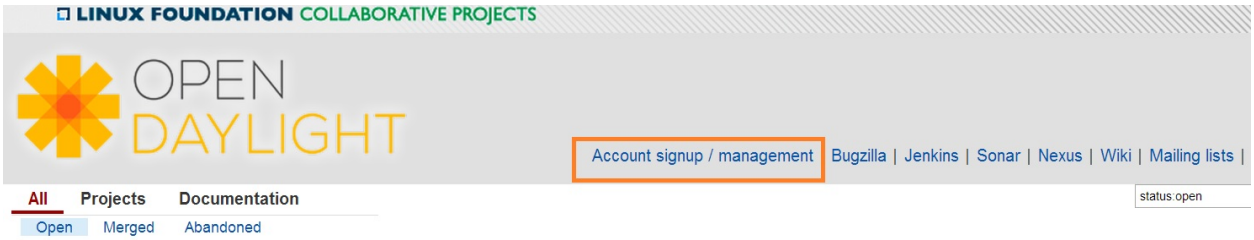


Fig. 2.10: Gerrit Account signup/management link

Fig. 2.11: Sign-up link for Gerrit account

Fig. 2.12: Sign-up with User Name/Password Image

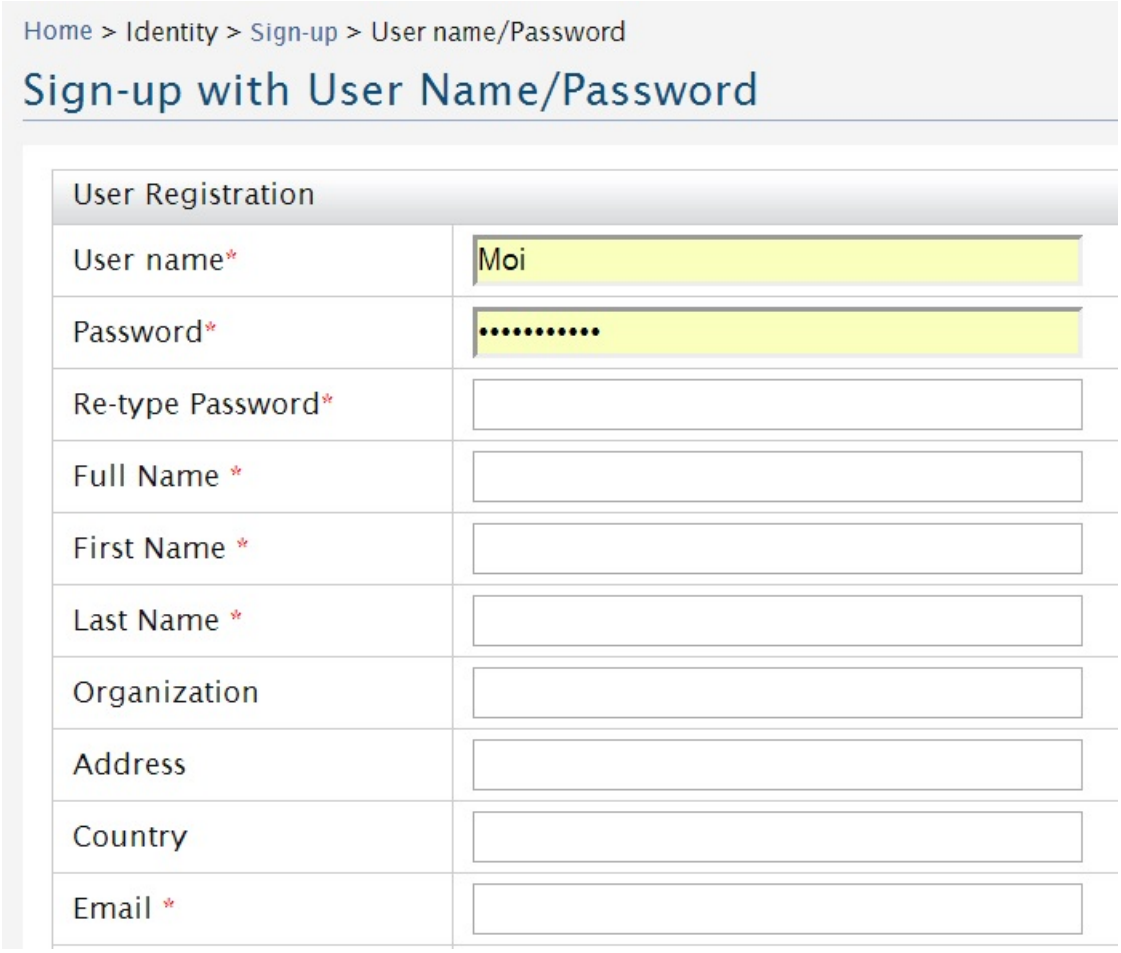


Fig. 2.13: Filling out the details

5. Fill out the details in the account creation form and then click **Submit**.

You now have an OpenDaylight account that can be used with Gerrit to pull the OpenDaylight code.

Generating SSH keys for your system

You must have SSH keys for your system to register with your Gerrit account. The method for generating SSH keys is different for different types of operating systems.

The key you register with Gerrit must be identical to the one you will use later to pull or edit the code. For example, if you have a development VM which has a different UID login and keygen than that of your laptop, the SSH key you generate for the VM is different from the laptop. If you register the SSH key generated on your VM with Gerrit and do not reuse it on your laptop when using Git on the laptop, the pull fails.

Note: For more information on SSH keys for Ubuntu, see https://help.ubuntu.com/community/SSH/OpenSSH/Keys. For generating SSH keys for Windows, see https://help.github.com/articles/generating-ssh-keys.

For a system running Ubuntu operating system, follow the steps below:

1. Run the following command:

```
mkdir ~/.ssh
chmod 700 ~/.ssh
ssh-keygen -t rsa
```

2. You are prompted for a location to save the keys, and a passphrase for the keys.

   This passphrase protects your private key while it is stored on the hard drive. You must use the passphrase to use the keys every time you need to login to a key-based system:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/b/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/b/.ssh/id_rsa.
Your public key has been saved in /home/b/.ssh/id_rsa.pub.
```

Your public key is now available as **.ssh/id_rsa.pub** in your home folder.

Registering your SSH key with Gerrit

1. Using a Google Chrome or Mozilla Firefox browser, go to https://git.opendaylight.org/gerrit.
2. Click **Sign In** to access the OpenDaylight repository.
3. Click your name in the top right corner of the window and then click **Settings**.

   The **Settings** page is displayed.
4. Click **SSH Public Keys** under **Settings**.
5. Click **Add Key**.
6. In the **Add SSH Public Key** text box, paste the contents of your **id_rsa.pub** file and then click **Add**.

To verify your SSH key is working correctly, try using an SSH client to connect to Gerrit's SSHD port:

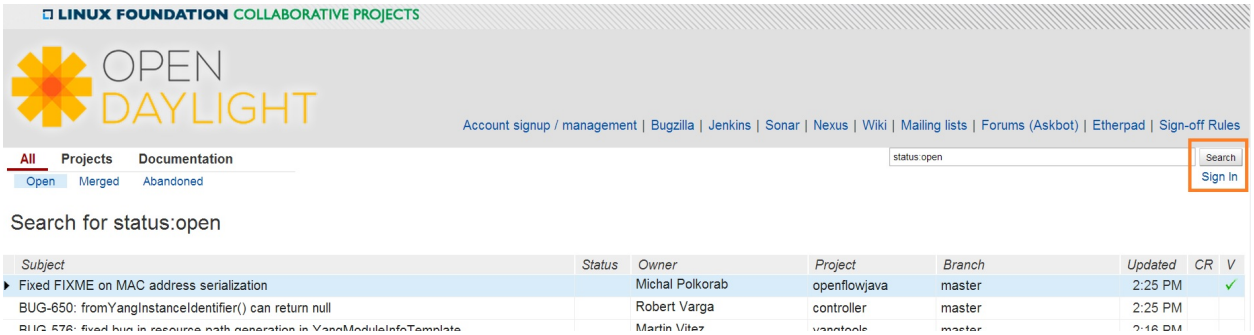


Fig. 2.14: Signin in to OpenDaylight repository

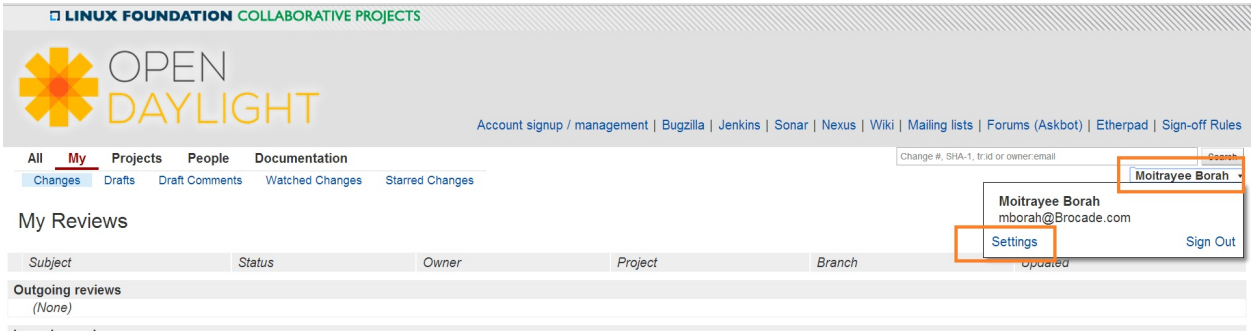


Fig. 2.15: Settings page for your Gerrit account

Fig. 2.16: Adding your SSH key

```
$ ssh -p 29418 <sshusername>@git.opendaylight.org
Enter passphrase for key '/home/cisco/.ssh/id_rsa':
****    Welcome to Gerrit Code Review    ****
Hi <user>, you have successfully connected over SSH.
Unfortunately, interactive shells are disabled.
To clone a hosted Git repository, use: git clone ssh://<user>@git.opendaylight.
↪org:29418/REPOSITORY_NAME.git
Connection to git.opendaylight.org closed.
```

You can now proceed to either Pulling, Hacking, and Pushing the Code from the CLI or Pulling, Hacking, and Pushing the Code from Eclipse depending on your implementation.

Using https to push to Gerrit

It is highly recommended to use ssh to push to Gerrit. In the event that you cannot use ssh, e.g., a corporate firewall is blocking blocking you, then falling back to pushing via https should work.

Gerrit does not allow you to use your regular account credentials when pushing via https. Instead it requires you to first generate a http password via the Gerrit Web UI and use that as the password when pushing via https.

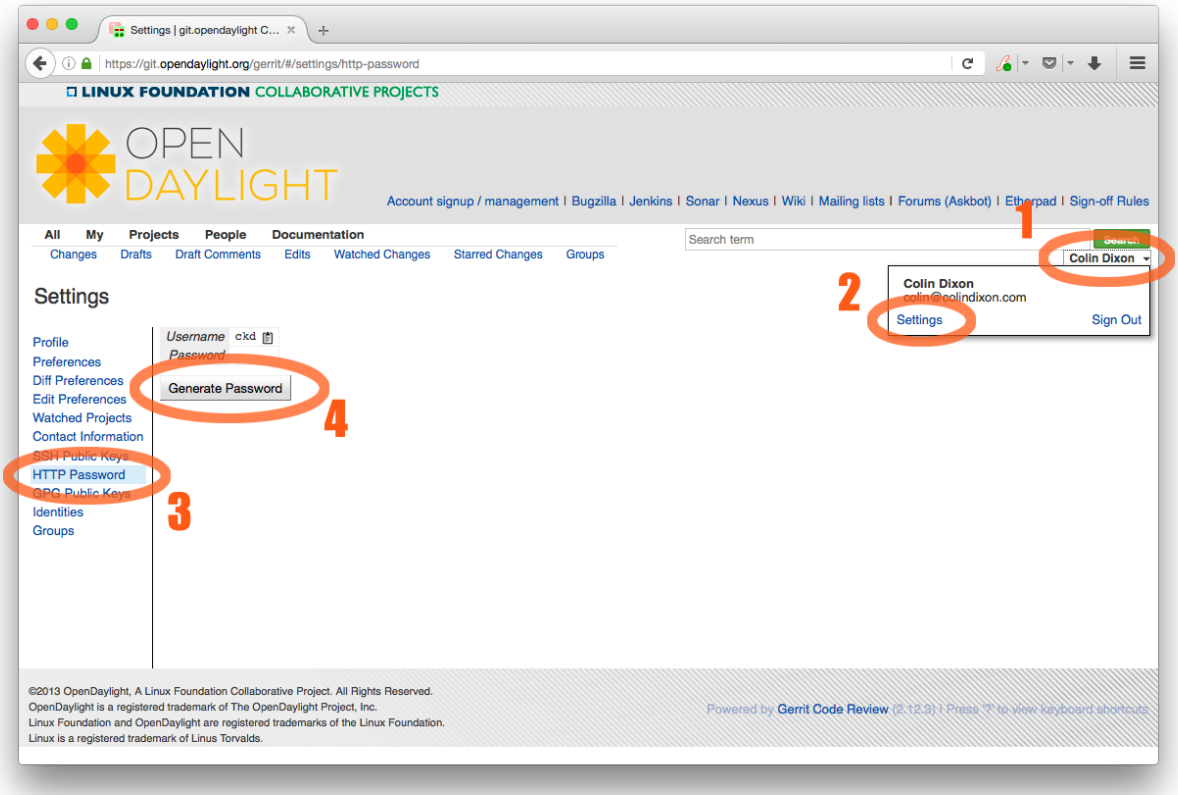


Fig. 2.17: Setting up an https password to push using https instead of ssh.

To do this:

1. navigate to https://git.opendaylight.org/gerrit/#/settings/http-password (Steps 1, 2 and 3 in the image above.)

2. click the **Generate Password** button.

Gerrit will then generate a random password which you will need to use as your password when using Git to push code to Gerrit via https.

To push using git over https, do the following

```
git remote add https https://git.opendaylight.org/gerrit/p/${repo_name}  # adds an␣
↪https version of the git server
git push https HEAD:refs/for/${branch_name}                              # will ask you for␣
↪your username and password
                                                                         #   ${branch_name}␣
↪should usually be master,
                                                                         #   but can be␣
↪stable/carbon or something else
```

Signing Gerrit Commits

1. Generate your GPG key.

   The following instructions work on a Mac, but the general approach should be the same on other OSes.

   ```
   brew install gpg2  # If you don't have homebrew, get that here: http://brew.sh/
   gpg2 --gen-key
   # pick 1 for "RSA and RSA"
   # enter 4096 to creat a 4096-bit key
   # enter an expiration time, I picked 2y for 2 years
   # enter y to accept the expiration time
   # pick O or Q to accept your name/email/comment
   # enter a pass phrase twice. it seems like backspace doesn't work, so type␣
   ↪carefully
   gpg2 --fingerprint
   # you'll get something like this:
   # spectre:~ ckd$ gpg2 --fingerprint
   # /Users/ckd/.gnupg/pubring.gpg
   # ------------------------------
   # pub   4096R/F566C9B1 2015-04-06 [expires: 2017-04-05]
   #       Key fingerprint = 7C37 02AC D651 1FA7 9209  48D3 5DD5 0C4B F566 C9B1
   # uid       [ultimate] Colin Dixon <colin at colindixon.com>
   # sub   4096R/DC1497E1 2015-04-06 [expires: 2017-04-05]
   # you're looking for the part after 4096R, which is your key ID
   gpg2 --send-keys $KEY_ID
   # in the above example, the $KEY_ID would be F566C9B1
   # you should see output like this:
   # gpg: sending key F566C9B1 to hkp server keys.gnupg.net
   ```

   If you're trying to participate in an OpenDaylight keysigning, then send the output of `gpg2 --fingerprint $KEY_ID` to keysigning@opendaylight.org

   ```
   gpg2 --fingerprint $KEY_ID
   # in the above example, the $KEY_ID would be F566C9B1
   # in my case, the output was:
   # pub   4096R/F566C9B1 2015-04-06 [expires: 2017-04-05]
   #       Key fingerprint = 7C37 02AC D651 1FA7 9209  48D3 5DD5 0C4B F566 C9B1
   # uid       [ultimate] Colin Dixon <colin at colindixon.com>
   # sub   4096R/DC1497E1 2015-04-06 [expires: 2017-04-05]
   ```

2. Install gpg, instead of or addition to gpg2. It appears as though gpg2 has annoying things that it does when asking for your passphrase, which I haven't debugged yet.

Note: you can tell Git to use gpg by doing: `git config --global gpg.program gpg2` but that then will seem to struggle asking for your passphrase unless you have your gpg-agent set up right.

---

3. Add you GPG to Gerrit

   (a) Run the following at the CLI:

```
gpg --export -a $FINGER_PRINT
# e.g., gpg --export -a F566C9B1
# in my case the output looked like:
# -----BEGIN PGP PUBLIC KEY BLOCK-----
# Version: GnuPG v2
#
# mQINBFUisGABEAC/DkcjNUhxQkRLdfbfdlq9NlfDusWri0cXLVz4YN1cTUTF5HiW
# ...
# gJT+FwDvCGgaE+JGlmXgjv0WSd4f9cNXkgYqfb6mpji0F3TF2HXXiVPqbwJ1V3I2
# NA+l+/koCW0aMReK
# =A/ql
# -----END PGP PUBLIC KEY BLOCK-----
```

   (b) Browse to https://git.opendaylight.org/gerrit/#/settings/gpg-keys

   (c) Click Add Key...

   (d) Copy the output from the above command, paste it into the box, and click Add

4. Set up your Git to sign commits and push signatures

```
git config commit.gpgsign true
git config push.gpgsign true
git config user.signingkey $FINGER_PRINT
# e.g., git config user.signingkey F566C9B1
```

Note: you can do this instead with `git commit -S` You can use `git commit -S` and `git push --signed` on the CLI instead of configuring it in config if you want to control which commits use your signature.

---

5. Commit and push a change

   (a) change a file

   (b) `git commit -asm "test commit"`

   Note: this should result in Git asking you for your passphrase

   ---

   (c) `git review`

   Note: this should result in Git asking you for your passphrase

   ---

   Note: annoyingly, the presence of a gpgp signature or pushing of a gpg signature isn't recognized as a

"change" by Gerrit, so if you forget to do either, you need to change something about the commit to get Gerrit to accept the patch again. Slightly tweaking the commit message is a good way.

Note: this assumes you have git review set up and push.gpgsign set to true. Otherwise:

```
git push --signed gerrit HEAD:refs/for/master
```

Note: this assumes you have your gerrit remote set up, if not it's something like: `ssh://ckd@git.opendaylight.org:29418/<repo>.git` where repo is something like docs or controller

---

6. Verify that your commit is signed by going to the change in Gerrit and checking for a green check (instead of a blue ?) next to your name.

Setting up gpg-agent on a Mac

1. Install `gpg-agent` and `pinentry-mac` using `brew`:

```
brew install gpg-agent pinentry-mac
```

2. Edit your `~/.gnupg/gpg.conf` contain the line:

```
use-agent
```

3. Edit your `~/.gnupg/gpg-agent.conf` to something like:

```
use-standard-socket
enable-ssh-support
default-cache-ttl 600
max-cache-ttl 7200
pinentry-program /usr/local/bin/pinentry-mac
```

4. Edit your `.bash_profile` or equivalent file to contain the following:

```
[ -f ~/.gpg-agent-info ] && source ~/.gpg-agent-info
if [ -S "${GPG_AGENT_INFO%%:*}" ]; then
  export GPG_AGENT_INFO
else
  eval $( gpg-agent --daemon --write-env-file ~/.gpg-agent-info )
fi
```

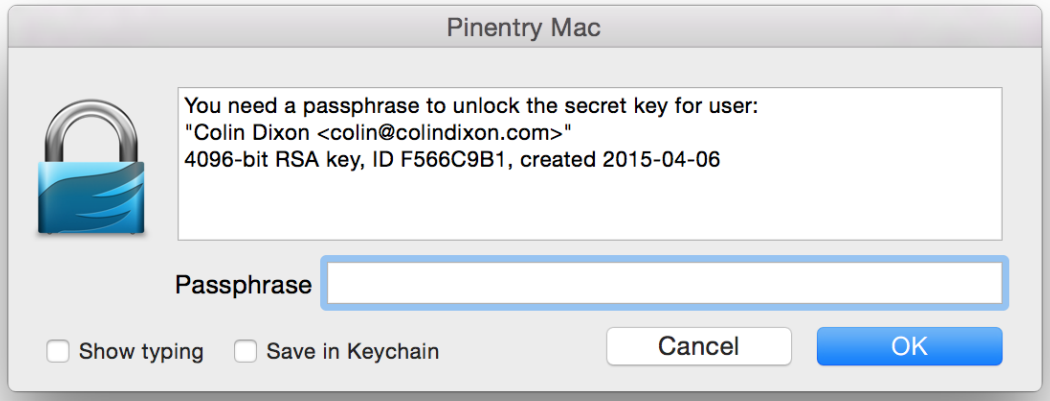
5. Kill any stray `gpg-agent` daemons running:

```
sudo killall gpg-agent
```

6. Restart your terminal (or log in and out) to reload the your `.bash_profile` or equivalent file
7. The next time a Git operation makes a call to gpg, it should use your gpg-agent to run a GUI window to ask for your passphrase and give you an option to save your passphrase in the keychain.

Developing Apps on the OpenDaylight controller

This section provides information that is required to develop apps on the OpenDaylight controller.



You can either develop apps within the controller using the model-driven SAL (MD-SAL) archetype or develop external apps and use the RESTCONF to communicate with the controller.

Overview

This section enables you to get started with app development within the OpenDaylight controller. In this example, you perform the following steps to develop an app.

1. Create a local repository for the code using a simple build process.
2. Start the OpenDaylight controller.
3. Test a simple remote procedure call (RPC) which you have created based on the principle of *hello world*.

Pre requisites

This example requires the following.

- A development environment with following set up and working correctly from the shell:
  - Maven 3.1.1 or later
  - Java 7- or Java 8-compliant JDK
  - An appropriate Maven settings.xml file. A simple way to get the default OpenDaylight settings.xml file is:

```
cp -n ~/.m2/settings.xml{,.orig} ; \wget -q -O - https://raw.
↪githubusercontent.com/opendaylight/odlparent/stable/boron/settings.xml > ~/.
↪m2/settings.xml
```

Note: If you are using Linux or Mac OS X as your development OS, your local repository is ~/.m2/repository. For other platforms the local repository location will vary.

Building an example module

To develop an app perform the following steps.

1. Create an *Example* project using Maven and an archetype called the *opendaylight-startup-archetype*. If you are downloading this project for the first time, then it will take sometime to pull all the code from the remote repository.

```
mvn archetype:generate -DarchetypeGroupId=org.opendaylight.controller -
↪DarchetypeArtifactId=opendaylight-startup-archetype \
-DarchetypeRepository=https://nexus.opendaylight.org/content/repositories/public/␣
↪\
-DarchetypeCatalog=https://nexus.opendaylight.org/content/repositories/public/
↪archetype-catalog.xml
```

2. Update the properties values as follows. Ensure that the groupid and the artifactid is lower case.

```
Define value for property 'groupId': : org.opendaylight.example
Define value for property 'artifactId': : example
Define value for property 'version':  1.0-SNAPSHOT: : 1.0.0-SNAPSHOT
Define value for property 'package':  org.opendaylight.example: :
Define value for property 'classPrefix':  ${artifactId.substring(0,1).
↪toUpperCase()}${artifactId.substring(1)}
Define value for property 'copyright': : Copyright (c) 2015 Yoyodyne, Inc.
```

3. Accept the default value of classPrefix that is, `(${artifactId.substring(0,1).toUpperCase()}${artifactId.substring(1)})`. The classPrefix creates a Java Class Prefix by capitalizing the first character of the artifactId.

Note: In this scenario, the classPrefix used is "Example". Create a top-level directory for the archetype.

```
${artifactId}/
example/
cd example/
api/
artifacts/
features/
impl/
karaf/
pom.xml
```

4. Build the *example* project.

Note: Depending on your development machine's specification this might take a little while. Ensure that you are in the project's root directory, example/, and then issue the build command, shown below.

```
mvn clean install
```

5. Start the *example* project for the first time.

```
cd karaf/target/assembly/bin
ls
./karaf
```

6. Wait for the karaf cli that appears as follows. Wait for OpenDaylight to fully load all the components. This can take a minute or two after the prompt appears. Check the CPU on your dev machine, specifically the Java process to see when it calms down.

```
opendaylight-user@root>
```

7. Verify if the "example" module is built and search for the log entry which includes the entry *ExampleProvider Session Initiated*.

```
log:display | grep Example
```

8. Shutdown the OpenDaylight through the console by using the following command.

```
shutdown -f
```

Defining a Simple Hello World RPC

1. Run the maven archetype *opendaylight-startup-archetype*, and create the *hello* project.

```
mvn archetype:generate -DarchetypeGroupId=org.opendaylight.controller -
↪DarchetypeArtifactId=opendaylight-startup-archetype \
-DarchetypeRepository=http://nexus.opendaylight.org/content/repositories/
↪opendaylight.snapshot/ \
-DarchetypeCatalog=http://nexus.opendaylight.org/content/repositories/
↪opendaylight.snapshot/archetype-catalog.xml
```

2. Update the Properties values as follows.

```
Define value for property 'groupId': : org.opendaylight.hello
Define value for property 'artifactId': : hello
Define value for property 'version':  1.0-SNAPSHOT: : 1.0.0-SNAPSHOT
Define value for property 'package':  org.opendaylight.hello: :
Define value for property 'classPrefix':  ${artifactId.substring(0,1).
↪toUpperCase()}${artifactId.substring(1)}
Define value for property 'copyright': : Copyright(c) Yoyodyne, Inc.
```

3. View the *hello* project.

```
cd hello/
ls -1
api
artifacts
features
impl
karaf
pom.xml
```

4. Build *hello* project by using the following command.

```
mvn clean install
```

5. Verify that the project is functioning by executing karaf.

```
cd karaf/target/assembly/bin
./karaf
```

6. The karaf cli appears as follows.
   NOTE: Remember to wait for OpenDaylight to load completely. Verify that the Java process CPU has stabilized.+

```
opendaylight-user@root>
```

7. Verify that the *hello* module is loaded by checking the log.

```
log:display | grep Hello
```

8. Shutdown karaf.

```
shutdown -f
```

9. Return to the top of the directory structure:

```
cd ../../../../
```

10. View the entry point to understand where the log line came from. The entry point is in the impl project:

```
impl/src/main/java/org/opendaylight/hello/impl/HelloProvider.java
```

11. Add any new things that you are doing in your implementation by using the HelloProvider.onSessionInitiate method. Its analogous to an Activator.

```
@Override
    public void onSessionInitiated(ProviderContext session) {
        LOG.info("HelloProvider Session Initiated");
    }
```

Add a simple HelloWorld RPC API

1. Navigate to the file.

```
Edit
api/src/main/yang/hello.yang
```

2. Edit this file as follows. In the following example, we are adding the code in a YANG module to define the *hello-world* RPC:
3. Return to the hello/api directory and build your API as follows.

```
cd ../../../
mvn clean install
```

Implement the HelloWorld RPC API

1. Define the HelloService, which is invoked through the *hello-world* API.

```
cd ../impl/src/main/java/org/opendaylight/hello/impl/
```

2. Create a new file called HelloWorldImpl.java and add in the code below.

```
package org.opendaylight.hello.impl;
import java.util.concurrent.Future;
import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.
↪rev150105.HelloService;
import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.
↪rev150105.HelloWorldInput;
import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.
↪rev150105.HelloWorldOutput;
import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.
↪rev150105.HelloWorldOutputBuilder;
import org.opendaylight.yangtools.yang.common.RpcResult;
import org.opendaylight.yangtools.yang.common.RpcResultBuilder;
public class HelloWorldImpl implements HelloService {
    @Override
    public Future<RpcResult<HelloWorldOutput>> helloWorld(HelloWorldInput input) {
        HelloWorldOutputBuilder helloBuilder = new HelloWorldOutputBuilder();
        helloBuilder.setGreating("Hello " + input.getName());
        return RpcResultBuilder.success(helloBuilder.build()).buildFuture();
    }
}
```

3. The HelloProvider.java file is in the current directory. Register the RPC that you created in the *hello.yang* file in the HelloProvider.java file. You can either edit the HelloProvider.java to match what is below or you can simple replace it with the code below.

```
/*
 * Copyright(c) Yoyodyne, Inc. and others.  All rights reserved.
 *
 * This program and the accompanying materials are made available under the
 * terms of the Eclipse Public License v1.0 which accompanies this distribution,
 * and is available at http://www.eclipse.org/legal/epl-v10.html
 */
package org.opendaylight.hello.impl;

import org.opendaylight.controller.sal.binding.api.BindingAwareBroker.
↪ProviderContext;
import org.opendaylight.controller.sal.binding.api.BindingAwareBroker.
↪RpcRegistration;
import org.opendaylight.controller.sal.binding.api.BindingAwareProvider;
import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.
↪rev150105.HelloService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class HelloProvider implements BindingAwareProvider, AutoCloseable {
    private static final Logger LOG = LoggerFactory.getLogger(HelloProvider.
↪class);
    private RpcRegistration<HelloService> helloService;
    @Override
    public void onSessionInitiated(ProviderContext session) {
        LOG.info("HelloProvider Session Initiated");
        helloService = session.addRpcImplementation(HelloService.class, new␣
↪HelloWorldImpl());
    }
    @Override
    public void close() throws Exception {
        LOG.info("HelloProvider Closed");
        if (helloService != null) {
```

```
            helloService.close();
        }
    }
}
```

4. Optionally, you can also build the Java classes which will register the new RPC. This is useful to test the edits you have made to HelloProvider.java and HelloWorldImpl.java.

```
cd ../../../../../../../
mvn clean install
```

5. Return to the top level directory

```
cd ../
```

6. Build the entire *hello* again, which will pickup the changes you have made and build them into your project:

```
mvn clean install
```

Execute the *hello* project for the first time

1. Run karaf

```
cd ../karaf/target/assembly/bin
./karaf
```

2. Wait for the project to load completely. Then view the log to see the loaded *Hello* Module:

```
log:display | grep Hello
```

Test the *hello-world* RPC via REST

There are a lot of ways to test your RPC. Following are some examples.

1. Using the API Explorer through HTTP
2. Using a browser REST client

Using the API Explorer through HTTP

1. Navigate to apidoc UI with your web browser.
   NOTE: In the URL mentioned above, Change *localhost* to the IP/Host name to reflect your development machine's network address.
2. Select

```
hello(2015-01-05)
```

3. Select

```
POST /operations/hello:hello-world
```

4. Provide the required value.

```
{"hello:input": { "name":"Your Name"}}
```

5. Click the button.
6. Enter the username and password, by default the credentials are admin/admin.
7. In the response body you should see.

```
{
  "output": {
    "greating": "Hello Your Name"
  }
}
```

Using a browser REST client

For example, use the following information in the Firefox plugin *RESTClient*
[https://github.com/chao/RESTClient\protect\T1\textbraceright

```
POST: http://192.168.1.43:8181/restconf/operations/hello:hello-world
```

Header:

```
application/json
```

Body:

```
{"input": {
    "name": "Andrew"
  }
}
```

Troubleshooting

If you get a response code 501 while attempting to POST /operations/hello:hello-world, check the file: HelloProvider.java and make sure the helloService member is being set. By not invoking "session.addRpcImplementation()" the REST API will be unable to map /operations/hello:hello-world url to HelloWorldImpl.

2.1.2 Project-specific Developer Guides

ALTO Developer Guide

Overview

The topics of this guide are:

1. How to add alto projects as dependencies;
2. How to put/fetch data from ALTO;
3. Basic API and DataType;

4. How to use customized service implementations.

Adding ALTO Projects as Dependencies

Most ALTO packages can be added as dependencies in Maven projects by putting the following code in the *pom.xml* file.

```
<dependency>
    <groupId>org.opendaylight.alto</groupId>
    <artifactId>${THE_NAME_OF_THE_PACKAGE_YOU_NEED}</artifactId>
    <version>${ALTO_VERSION}</version>
</dependency>
```

The current stable version for ALTO is `0.3.0-Boron`.

Putting/Fetching data from ALTO

Using RESTful API

There are two kinds of RESTful APIs for ALTO: the one provided by `alto-northbound` which follows the formats defined in RFC 7285, and the one provided by RESTCONF whose format is defined by the YANG model proposed in this draft.

One way to get the URLs for the resources from `alto-northbound` is to visit the IRD service first where there is a `uri` field for every entry. However, the IRD service is not yet implemented so currently the developers have to construct the URLs themselves. The base URL is `/alto` and below is a list of the specific paths defined in `alto-core/standard-northbound-route` using Jersey `@Path` annotation:

- `/ird/{rid}`: the path to access *IRD* services;
- `/networkmap/{rid}[/{tag}]`: the path to access *Network Map* and *Filtered Network Map* services;
- `/costmap/{rid}[/{tag}[/{mode}/{metric}]]`: the path to access *Cost Map* and *Filtered Cost Map* services;
- `/endpointprop`: the path to access *Endpoint Property* services;
- `/endpointcost`: the path to access *Endpoint Cost* services.

Note: The segments in brackets are optional.

If you want to fetch the data using RESTCONF, it is highly recommended to take a look at the `apidoc` page (http://\protect\T1\textbraceleftcontroller_ip\protect\T1\textbraceright:8181/apidoc/explorer/index.html) after installing the `odl-alto-release` feature in karaf.

It is also worth pointing out that `alto-northbound` only supports `GET` and `POST` operations so it is impossible to manipulate the data through its RESTful APIs. To modify the data, use `PUT` and `DELETE` methods with RESTCONF.

Note: The current implementation uses the `configuration` data store and that enables the developers to modify the data directly through RESTCONF. In the future this approach might be disabled in the core packages of ALTO but may still be available as an extension.

Using MD-SAL

You can also fetch data from the datastore directly.

First you must get the access to the datastore by registering your module with a data broker.

Then an `InstanceIdentifier` must be created. Here is an example of how to build an `InstanceIdentifier` for a *network map*:

```
import org.opendaylight...alto...Resources;
import org.opendaylight...alto...resources.NetworkMaps;
import org.opendaylight...alto...resources.network.maps.NetworkMap;
import org.opendaylight...alto...resources.network.maps.NetworkMapKey;
...
protected
InstanceIdentifier<NetworkMap> getNetworkMapIID(String resource_id) {
  ResourceId rid = ResourceId.getDefaultInstance(resource_id);
  NetworkMapKey key = new NetworkMapKey(rid);
  InstanceIdentifier<NetworkMap> iid = null;
  iid = InstanceIdentifier.builder(Resources.class)
                          .child(NetworkMaps.class)
                          .child(NetworkMap.class, key)
                          .build();
  return iid;
}
...
```

With the `InstanceIdentifier` you can use `ReadOnlyTransaction`, `WriteTransaction` and `ReadWriteTransaction` to manipulate the data accordingly. The `simple-impl` package, which provides some of the AD-SAL APIs mentioned above, is using this method to get data from the datastore and then convert them into RFC7285-compatible objects.

Basic API and DataType

1. alto-basic-types: Defines basic types of ALTO protocol.
2. alto-service-model-api: Includes the YANG models for the five basic ALTO services defined in RFC 7285.
3. alto-resourcepool: Manages the meta data of each ALTO service, including capabilities and versions.
4. alto-northbound: Provides the root of RFC7285-compatible services at http://localhost:8080/alto.
5. alto-northbound-route: Provides the root of the network map resources at http://localhost:8080/alto/networkmap/.

How to customize service

Define new service API

Add a new module in `alto-core/standard-service-models`. For example, we named our service model module as `model-example`.

Implement service RPC

Add a new module in `alto-basic` to implement a service RPC in `alto-core`.

Currently `alto-core/standard-service-models/model-base` has defined a template of the service RPC. You can define your own RPC using `augment` in YANG. Here is an example in `alto-simpleird`.

Register northbound route

If necessary, you can add a northbound route module in `alto-core/standard-northbound-routes`.

Authentication, Authorization and Accounting (AAA) Services

Overview

Authentication, Authorization and Accounting (AAA) is a term for a framework controlling access to resources, enforcing policies to use those resources and auditing their usage. These processes are the fundamental building blocks for effective network management and security.

Authentication provides a way of identifying a user, typically by having the user enter a valid user name and valid password before access is granted. The process of authentication is based on each user having a unique set of criteria for gaining access. The AAA framework compares a user's authentication credentials with other user credentials stored in a database. If the credentials match, the user is granted access to the network. If the credentials don't match, authentication fails and access is denied.

Authorization is the process of finding out what an authenticated user is allowed to do within the system, which tasks can do, which API can call, etc. The authorization process determines whether the user has the authority to perform such actions.

Accounting is the process of logging the activity of an authenticated user, for example, the amount of data a user has sent and/or received during a session, which APIs called, etc.

Terms And Definitions

AAA Authentication, Authorization and Accounting.

Token A claim of access to a group of resources on the controller.

Domain A group of resources, direct or indirect, physical, logical, or virtual, for the purpose of access control.

User A person who either owns or has access to a resource or group of resources on the controller.

Role Opaque representation of a set of permissions, which is merely a unique string as admin or guest.

Credential Proof of identity such as user name and password, OTP, biometrics, or others.

Client A service or application that requires access to the controller.

Claim A data set of validated assertions regarding a user, e.g. the role, domain, name, etc.

IdP Identity Provider.

Quick Start

Building

Get the code:

```
git clone https://git.opendaylight.org/gerrit/aaa
```

Build it:

```
cd aaa && mvn clean install
```

Installing

AAA is automatically installed upon installation of odl-restconf, but you can install it yourself directly from the Karaf console through the following command:

```
feature:install odl-aaa-shiro
```

Pushing changes

The following are basic instructions to push your contributions to the project's GIT repository:

```
git add .
git commit -s
# make changes, add change id, etc.
git commit --amend
git push ssh://{username}@git.opendaylight.org:29418/aaa.git HEAD:refs/for/master
```

AAA Framework implementations

Since Boron release, the OpenDaylight's AAA services are based on the Apache Shiro Java Security Framework. The main configuration file for AAA is located at "etc/shiro.ini" relative to the OpenDaylight Karaf home directory.

Known limitations

The database (H2) used by ODL AAA Authentication store is not-cluster enabled. When deployed in a clustered environment each node needs to have its AAA user file synchronized using out of band means.

How to enable AAA

AAA is enabled through installing the odl-aaa-shiro feature. The vast majority of OpenDaylight's northbound APIs (and all RESTCONF APIs) are protected by AAA by default when installing the +odl-restconf+ feature, since the odl-aaa-shiro is automatically installed as part of them.

How to disable AAA

Edit the "etc/shiro.ini" file and replace the following:

```
/** = authcBasic
```

with

```
/** = anon
```

Then, restart the Karaf process.

How application developers can leverage AAA to provide servlet security

In order to provide security to a servlet, add the following to the servlet's web.xml file as the first filter definition:

```
<context-param>
  <param-name>shiroEnvironmentClass</param-name>
  <param-value>org.opendaylight.aaa.shiro.web.env.KarafIniWebEnvironment</param-value>
</context-param>

<listener>
    <listener-class>org.apache.shiro.web.env.EnvironmentLoaderListener</listener-
↪class>
</listener>

<filter>
    <filter-name>ShiroFilter</filter-name>
    <filter-class>org.opendaylight.aaa.shiro.filters.AAAShiroFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>AAAShiroFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Note: It is very important to place this AAAShiroFilter as the first javax.servlet.Filter, as Jersey applies Filters in the order they appear within web.xml. Placing the AAAShiroFilter first ensures incoming HTTP/HTTPS requests have proper credentials before any other filtering is attempted.

AAA Realms

AAA plugin utilizes the Shiro Realms to support pluggable authentication & authorization schemes. There are two parent types of realms:

- AuthenticatingRealm
  - Provides no Authorization capability.
  - Users authenticated through this type of realm are treated equally.
- AuthorizingRealm
  - AuthorizingRealm is a more sophisticated AuthenticatingRealm, which provides the additional mechanisms to distinguish users based on roles.
  - Useful for applications in which roles determine allowed capabilities.

OpenDaylight contains five implementations:

- TokenAuthRealm
  - An AuthorizingRealm built to bridge the Shiro-based AAA service with the h2-based AAA implementation.

  - Exposes a RESTful web service to manipulate IdM policy on a per-node basis. If identical AAA policy is desired across a cluster, the backing data store must be synchronized using an out of band method.
  - A python script located at "etc/idmtool" is included to help manipulate data contained in the TokenAuthRealm.
  - Enabled out of the box. This is the realm configured by default.
- ODLJndiLdapRealm
  - An AuthorizingRealm built to extract identity information from IdM data contained on an LDAP server.
  - Extracts group information from LDAP, which is translated into OpenDaylight roles.
  - Useful when federating against an existing LDAP server, in which only certain types of users should have certain access privileges.
  - Disabled out of the box.
- ODLJndiLdapRealmAuthNOnly
  - The same as ODLJndiLdapRealm, except without role extraction. Thus, all LDAP users have equal authentication and authorization rights.
  - Disabled out of the box.
- ODLActiveDirectoryRealm
  - Wraps the generic ActiveDirectoryRealm provided by Shiro. This allows for enhanced logging as well as isolation of all realms in a single package, which enables easier import by consuming servlets.
  - Disabled out of the box.
- KeystoneAuthRealm
  - This realm authenticates OpenDaylight users against the OpenStack's Keystone server by using the Keystone's Identity API v3 or later.
  - Disabled out of the box.

Note: More than one Realm implementation can be specified. Realms are attempted in order until authentication succeeds or all realm sources are exhausted. Edit the **securityManager.realms = $tokenAuthRealm** property in shiro.ini and add all the realms needed separated by commas.

TokenAuthRealm

How it works

The TokenAuthRealm is the default Authorization Realm deployed in OpenDaylight. TokenAuthRealm uses a direct authentication mechanism as shown in the following picture:

A user presents some credentials (e.g., username/password) directly to the OpenDaylight controller token endpoint /oauth2/token and receives an access token, which then can be used to access protected resources on the controller.

How to access the H2 database

The H2 database provides an optional front-end Web interface, which can be very useful for new users. From the KARAF_HOME directory, you can run the following command to enable the user interface:

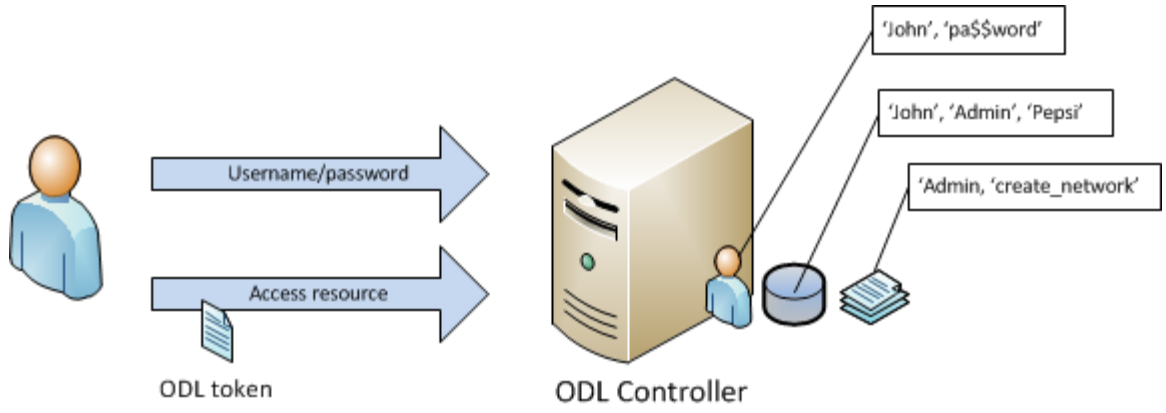


Fig. 2.18: TokenAuthRealm direct authentication mechanism

```
java -cp ./data/cache/org.eclipse.osgi/bundles/217/1/.cp/h2-1.4.185.jar
    org.h2.tools.Server -trace -pg -web -webAllowOthers -baseDir `pwd`
```

You can navigate to the following and login via the browser:

```
http://{IP}:8082/
```

ODLJndiLdapRealm

How it works

LDAP integration is provided in order to externalize identity management. This configuration allows federation with an external LDAP server. The user's OpenDaylight role parameters are mapped to corresponding LDAP attributes as specified by the groupRolesMap. Thus, an LDAP operator can provision attributes for LDAP users that support different OpenDaylight role structures.

ODLJndiLdapRealmAuthNOnly

How it works

This is useful for setups where all LDAP users are allowed equal access.

KeystoneAuthRealm

How it works

This realm authenticates OpenDaylight users against the OpenStack's Keystone server. This realm uses the Keystone's Identity API v3 or later.

As can shown on the above diagram, once configured, all the RESTCONF APIs calls will require sending **user**, **password** and optionally **domain** (1). Those credentials are used to authenticate the call against the Keystone server (2) and, if the authentication succeeds, the call will proceed to the MDSAL (3). The credentials must be provisioned in advance within the Keystone Server. The user and password are mandatory, while the domain is optional, in case

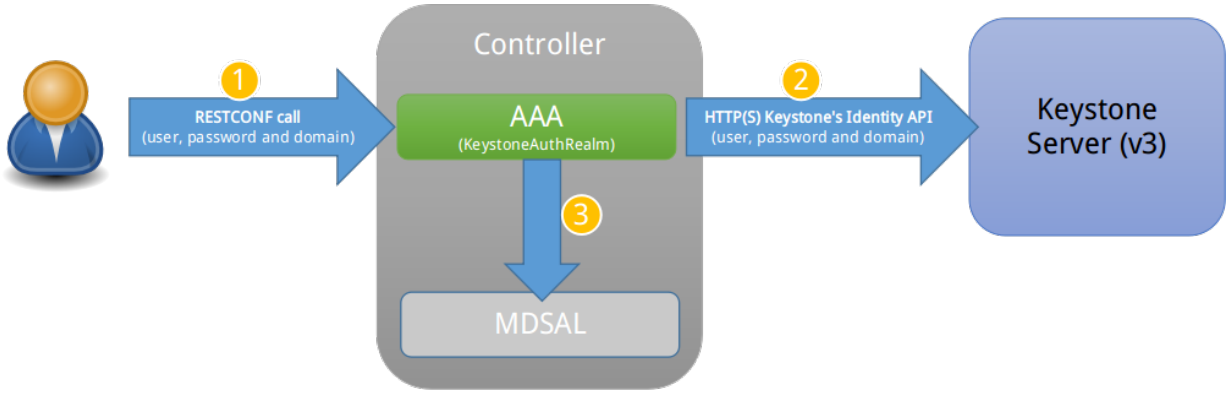


Fig. 2.19: KeystoneAuthRealm authentication/authorization mechanism

it is not provided within the REST call, the realm will default to (**Default**), which is hard-coded. The default domain can be also configured through the *shiro.ini* file (see the *AAA User Guide*).

The protocol between the Controller and the Keystone Server (2) can be either HTTPS or HTTP. In order to use HTTPS the Keystone Server's certificate must be exported and imported on the Controller (see the Certificate Management section).

Authorization Configuration

OpenDaylight supports two authorization engines at present, both of which are roughly similar in behavior:

- Shiro-Based Authorization
- MDSAL-Based Dynamic Authorization

Note: The preferred mechanism for configuring AAA Authentication is the MDSAL-Based Dynamic Authorization. Read the following section.

Shiro-Based Static Authorization

OpenDaylight AAA has support for Role Based Access Control (RBAC) based on the Apache Shiro permissions system. Configuration of the authorization system is done off-line; authorization currently cannot be configured after the controller is started. The Authorization provided by this mechanism is aimed towards supporting coarse-grained security policies, the MDSAL-Based mechanism allows for a more robust configuration capabilities. Shiro-based Authorization describes how to configure the Authentication feature in detail.

MDSAL-Based Dynamic Authorization

The MDSAL-Based Dynamic authorization uses the MDSALDynamicAuthorizationFilter engine to restrict access to particular URL endpoint patterns. Users may define a list of policies that are insertion-ordered. Order matters for that list of policies, since the first matching policy is applied. This choice was made to emulate behavior of the Shiro-Based Authorization mechanism.

A **policy** is a key/value pair, where the key is a **resource** (i.e., a "URL pattern") and the value is a list of **permissions** for the resource. The following describes the various elements of a policy:

- **Resource**: the resource is a string URL pattern as outlined by Apache Shiro. For more information, see http://shiro.apache.org/web.html.
- **Description**: an optional description of the URL endpoint and why it is being secured.
- **Permissions list**: a list of permissions for a particular policy. If more than one permission exists in the permissions list they are evaluated using logical "OR". A permission describes the prerequisites to perform HTTP operations on a particular endpoint. The following describes the various elements of a permission:
  - **Role**: the role required to access the target URL endpoint.
  - **Actions list**: a leaf-list of HTTP permissions that are allowed for a Subject possessing the required role.

This an example on how to limit access to the modules endpoint:

```
HTTP Operation:
put URL: /restconf/config/aaa:http-authorization/policies

headers: Content-Type: application/json Accept: application/json

body:
  { "aaa:policies":
    { "aaa:policies":
      [ { "aaa:resource": "/restconf/modules/**",
        "aaa:permissions": [ { "aaa:role": "admin",
                               "aaa:actions": [ "get",
                                                "post",
                                                "put",
                                                "patch",
                                                "delete"
                                              ]
                             }
                           ]
      }
    ]
    }
  }
```

The above example locks down access to the modules endpoint (and any URLS available past modules) to the "admin" role. Thus, an attempt from the OOB *admin* user will succeed with 2XX HTTP status code, while an attempt from the OOB *user* user will fail with HTTP status code 401, as the user *user* is not granted the "admin" role.

Accounting Configuration

Accounting is handled through the standard slf4j logging mechanisms used by the rest of OpenDaylight. Thus, one can control logging verbosity through manipulating the log levels for individual packages and classes directly through the Karaf console, JMX, or etc/org.ops4j.pax.logging.cfg. In normal operations, the default levels exposed do not provide much information about AAA services; this is due to the fact that logging can severely degrade performance.

All AAA logging is output to the standard karaf.log file. For debugging purposes (i.e., to enable maximum verbosity), issue the following command:

```
log:set TRACE org.opendaylight.aaa
```

Enable Successful/Unsuccessful Authentication Attempts Logging

By default, successful/unsuccessful authentication attempts are NOT logged. This is due to the fact that logging can severely decrease REST performance.

It is possible to add custom AuthenticationListener(s) to the Shiro-based configuration, allowing different ways to listen for successful/unsuccessful authentication attempts. Custom AuthenticationListener(s) must implement the org.apache.shiro.authc.AuthenticationListener interface.

Certificate Management

The **Certificate Management Service** is used to manage the keystores and certificates at the OpenDaylight distribution to easily provides the TLS communication.

The Certificate Management Service managing two keystores:

1. **OpenDaylight Keystore** which holds the OpenDaylight distribution certificate self sign certificate or signed certificate from a root CA based on generated certificate request.
2. **Trust Keystore** which holds all the network nodes certificates that shall to communicate with the OpenDaylight distribution through TLS communication.

The Certificate Management Service stores the keystores (OpenDaylight & Trust) as *.jks* files under configuration/ssl/ directory. Also the keystores could be stored at the MD-SAL datastore in case OpenDaylight distribution running at cluster environment. When the keystores are stored at MD-SAL, the Certificate Management Service rely on the **Encryption-Service** to encrypt the keystore data before storing it to MD-SAL and decrypted at runtime.

How to use the Certificate Management Service to manage the TLS communication

The following are the steps to configure the TLS communication within your feature or module:

1. It is assumed that there exists an already created OpenDaylight distribution project following this guide.
2. In the implementation bundle the following artifact must be added to its *pom.xml* file as dependency.

```
<dependency>
  <groupId>org.opendaylight.aaa</groupId>
  <artifactId>aaa-cert</artifactId>
  <version>0.5.0-SNAPSHOT</version>
</dependency>
```

3. Using the provider class in the implementation bundle needs to define a variable holding the Certificate Manager Service as in the following example:

```
import org.opendaylight.aaa.cert.api.ICertificateManager;
import org.opendaylight.controller.md.sal.binding.api.DataBroker;

public class UseCertManagerExampleProvider {
  private final DataBroker dataBroker;
  private final ICertificateManager caManager;

  public EncSrvExampleProvider(final DataBroker dataBroker, final ICertificateManager␣
↪caManager) {
    this.dataBroker = dataBroker;
    this.caManager = caManager;
  }
  public SSLEngine createSSLEngine() {
```

```
    final SSLContext sslContext = caManager.getServerContext();
    if (sslContext != null) {
      final SSLEngine sslEngine = sslContext.createSSLEngine();
      sslEngine.setEnabledCipherSuites(caManager.getCipherSuites());
      // DO the Implementation
      return sslEngine;
    }
  }
  public void init() {
      // TODO
  }
  public void close() {
      // TODO
  }
}
```

4. The Certificate Manager Service provides two main methods that are needed to establish the *SSLEngine* object, *getServerContext()* and *getCipherSuites()* as the above example shows. It also provides other methods such as *getODLKeyStore()* and *getTrustKeyStore()* that gives access to the OpenDaylight and Trust keystores.
5. Now the *ICertificateManager* need to be passed as an argument to the *UseCertManagerExampleProvider* within the implementation bundle blueprint configuration. The following example shows how:

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:odl="http://opendaylight.org/xmlns/blueprint/v1.0.0"
  odl:use-default-for-reference-types="true">
  <reference id="dataBroker"
    interface="org.opendaylight.controller.md.sal.binding.api.DataBroker"
    odl:type="default" />
  <reference id="aaaCertificateManager"
    interface="org.opendaylight.aaa.cert.api.ICertificateManager"
    odl:type="default-certificate-manager" />
  <bean id="provider"
    class="org.opendaylight.UseCertManagerExample.impl.UseCertManagerExampleProvider"
    init-method="init" destroy-method="close">
    <argument ref="dataBroker" />
    <argument ref="aaaCertificateManager" />
  </bean>
</blueprint>
```

6. After finishing the bundle implementation the feature module needs to be updated to include the *aaa-cert* feature in its feature bundle pom.xml file.

```
<properties>
  <aaa.version>0.5.0-SNAPSHOT</aaa.version>
</properties>
<dependency>
  <groupId>org.opendaylight.aaa</groupId>
  <artifactId>features-aaa</artifactId>
  <version>${aaa.version}</version>
  <classifier>features</classifier>
  <type>xml</type>
</dependency>
```

7. Now, in the feature.xml file add the Certificate Manager Service feature and its repository.

```
<repository>mvn:org.opendaylight.aaa/features-aaa/{VERSION}/xml/features</repository>
```

The Certificate Manager Service feature can be included inside the implementation bundle feature as shown in the following example:

```
<feature name='odl-UseCertManagerExample' version='${project.version}'
  description='OpenDaylight :: UseCertManagerExample'>
  <feature version='${mdsal.version}'>odl-mdsal-broker</feature>
  <feature version='${aaa.version}'>odl-aaa-cert</feature>
  <bundle>mvn:org.opendaylight.UseCertManagerExample/UseCertManagerExample-impl/
↪{VERSION}</bundle>
</feature>
```

8. Now the project can be built and the OpenDaylight distribution started to continue with the configuration process. See the User Guide for more details.

Encryption Service

The **AAA Encryption Service** is used to encrypt the OpenDaylight users' passwords and TLS communication certificates. This section shows how to use the AAA Encryption Service with an OpenDaylight distribution project to encrypt data.

1. It is assumed that there exists an already created OpenDaylight distribution project following this guide.
2. In the implementation bundle the following artifact must be added to its *pom.xml* file as dependency.

```
<dependency>
  <groupId>org.opendaylight.aaa</groupId>
  <artifactId>aaa-encrypt-service</artifactId>
  <version>0.5.0-SNAPSHOT</version>
</dependency>
```

3. Using the provider class in the implementation bundle needs to define a variable holding the Encryption Service as in the following example:

```
import org.opendaylight.aaa.encrypt.AAAEncryptionService;
import org.opendaylight.controller.md.sal.binding.api.DataBroker;

public class EncSrvExampleProvider {
private final DataBroker dataBroker;
  private final AAAEncryptionService encryService;

  public EncSrvExampleProvider(final DataBroker dataBroker, final␣
↪AAAEncryptionService encryService) {
    this.dataBroker = dataBroker;
    this.encryService = encryService;
  }
  public void init() {
    // TODO
  }
  public void close() {
    // TODO
  }
}
```

The AAAEncryptionService can be used to encrypt and decrypt any data based on project's needs.

4. Now the *AAAEncryptionService* needs to be passed as an argument to the *EncSrvExampleProvider* within the implementation bundle blueprint configuration. The following example shows how:

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:odl="http://opendaylight.org/xmlns/blueprint/v1.0.0"
  odl:use-default-for-reference-types="true">
  <reference id="dataBroker"
    interface="org.opendaylight.controller.md.sal.binding.api.DataBroker"
    odl:type="default" />
  <reference id="encryService" interface="org.opendaylight.aaa.encrypt.
↪AAAEncryptionService"/>
  <bean id="provider"
    class="org.opendaylight.EncSrvExample.impl.EncSrvExampleProvider"
    init-method="init" destroy-method="close">
    <argument ref="dataBroker" />
    <argument ref="encryService" />
  </bean>
</blueprint>
```

5. After finishing the bundle implementation the feature module needs to be updated to include the *aaa-encryption-service* feature in its feature bundle pom.xml file.

```
<dependency>
  <groupId>org.opendaylight.aaa</groupId>
  <artifactId>features-aaa</artifactId>
  <version>${aaa.version}</version>
  <classifier>features</classifier>
  <type>xml</type>
</dependency>
```

It is also necessary to add the *aaa.version* in the properties section:

```
<properties>
  <aaa.version>0.5.0-SNAPSHOT</aaa.version>
</properties>
```

6. Now, in the feature.xml file add the Encryption Service feature and its repository.

```
<repository>mvn:org.opendaylight.aaa/features-aaa/{VERSION}/xml/features</repository>
```

The Encryption Service feature can be included inside the implementation bundle feature as shown in the following example:

```
<feature name='odl-EncSrvExample' version='${project.version}' description=
↪'OpenDaylight :: EncSrvExample'>
  <feature version='${mdsal.version}'>odl-mdsal-broker</feature>
  <feature version='${aaa.version}'>odl-aaa-encryption-service</feature>
  <feature version='${project.version}'>odl-EncSrvExample-api</feature>
  <bundle>mvn:org.opendaylight.EncSrvExample/EncSrvExample-impl/{VERSION}</bundle>
</feature>
```

7. Now the project can be built and the OpenDaylight distribution started to continue with the configuration process. See the User Guide for more details.

BGP Developer Guide

Overview

This section provides an overview of the `odl-bgpcep-bgp-all` Karaf feature. This feature will install everything needed for BGP (Border Gateway Protocol) from establishing the connection, storing the data in RIBs (Route Information Base) and displaying data in network-topology overview.

BGP Architecture

Each feature represents a module in the BGPCEP codebase. The following diagram illustrates how the features are related.

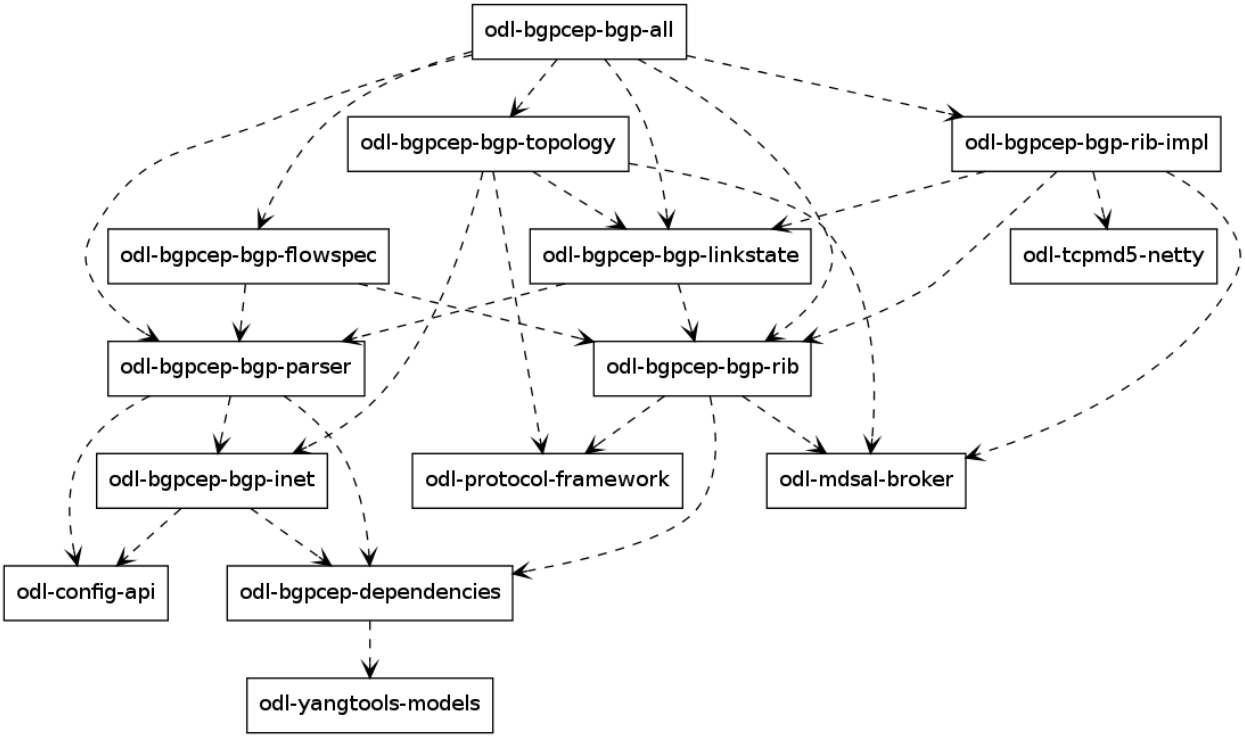


Fig. 2.20: BGP Dependency Tree

Key APIs and Interfaces

BGP concepts

This module contains the base BGP concepts contained in RFC 4271, RFC 4760, RFC 4456, RFC 1997 and RFC 4360.

All the concepts are described in one yang model: bgp-types.yang.

Outside generated classes, there is just one class NextHopUtil that contains methods for serializing and parsing NextHop.

BGP parser

Base BGP parser includes messages and attributes from RFC 4271, RFC 4760, RFC 1997 and RFC 4360.

API module defines BGP messages in YANG.

IMPL module contains actual parsers and serializers for BGP messages and Activator class

SPI module contains helper classes needed for registering parsers into activators

Registration

All parsers and serializers need to be registered into the *Extension provider*. This *Extension provider* is configured in initial configuration of the parser-spi module (`31-bgp.xml`).

```
<module>
 <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:bgp:parser:spi">
↪prefix:bgp-extensions-impl</type>
 <name>global-bgp-extensions</name>
 <extension>
  <type xmlns:bgpspi="urn:opendaylight:params:xml:ns:yang:controller:bgp:parser:spi">
↪bgpspi:extension</type>
  <name>base-bgp-parser</name>
 </extension>
 <extension>
  <type xmlns:bgpspi="urn:opendaylight:params:xml:ns:yang:controller:bgp:parser:spi">
↪bgpspi:extension</type>
  <name>bgp-linkstate</name>
 </extension>
</module>
```

- *base-bgp-parser* - will register parsers and serializers implemented in the bgp-parser-impl module
- *bgp-linkstate* - will register parsers and serializers implemented in the bgp-linkstate module

The bgp-linkstate module is a good example of a BGP parser extension.

The configuration of bgp-parser-spi specifies one implementation of *Extension provider* that will take care of registering mentioned parser extensions: SimpleBGPExtensionProviderContext. All registries are implemented in package bgp-parser-spi.

Serializing

The serializing of BGP elements is mostly done in the same way as in *PCEP*, the only exception is the serialization of path attributes, which is described here. Path attributes are different from any other BGP element, as path attributes don't implement one common interface, but this interface contains getters for individual path attributes (this structure is because update message can contain exactly one instance of each path attribute). This means, that a given *PathAttributes* object, you can only get to the specific type of the path attribute through checking its presence. Therefore method *serialize()* in *AttributeRegistry*, won't look up the registered class, instead it will go through the registrations and offer this object to the each registered parser. This way the object will be passed also to serializers unknown to module bgp-parser, for example to LinkstateAttributeParser. RFC 4271 recommends ordering path attributes, hence the serializers are ordered in a list as they are registered in the *Activator*. In other words, this is the only case, where registration ordering matters.

serialize() method in each Path Attribute parser contains check for presence of its attribute in the PathAttributes object, which simply returns, if the attribute is not there:

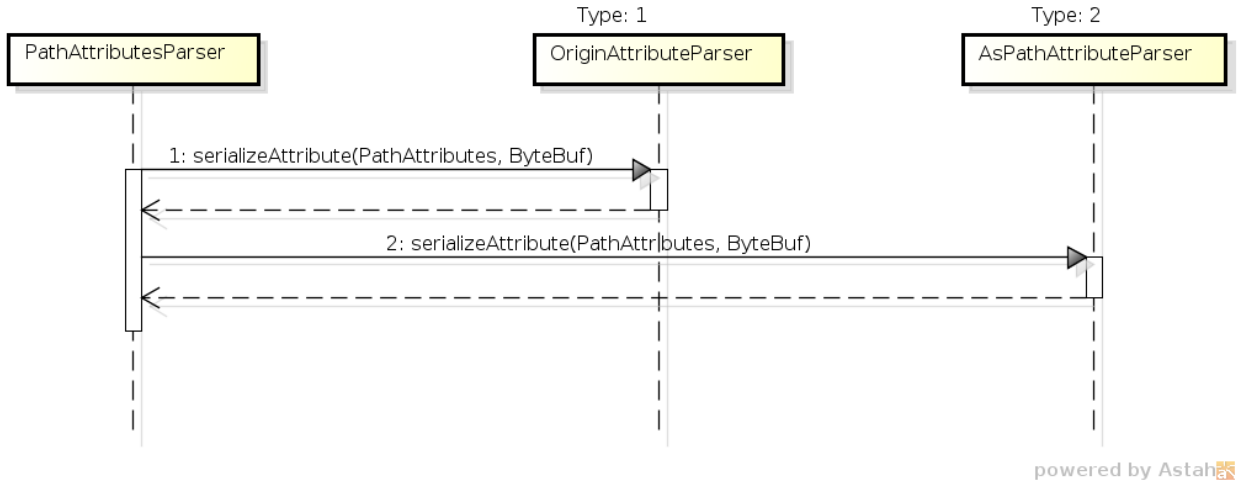


Fig. 2.21: PathAttributesSerialization

```
if (pathAttributes.getAtomicAggregate() == null) {
    return;
}
//continue with serialization of Atomic Aggregate
```

BGP RIB

The BGP RIB module can be divided into two parts:

- BGP listener and speaker session handling
- RIB handling.

Session handling

`31-bgp.xml` defines only bgp-dispatcher and the parser it should be using (global-bgp-extensions).

```
<module>
 <type>prefix:bgp-dispatcher-impl</type>
 <name>global-bgp-dispatcher</name>
 <bgp-extensions>
  <type>bgpspi:extensions</type>
  <name>global-bgp-extensions</name>
 </bgp-extensions>
 <boss-group>
  <type>netty:netty-threadgroup</type>
  <name>global-boss-group</name>
 </boss-group>
 <worker-group>
  <type>netty:netty-threadgroup</type>
  <name>global-worker-group</name>
 </worker-group>
</module>
```

For user configuration of BGP, check User Guide.

Synchronization

Synchronization is a phase, where upon connection, a BGP speaker sends all available data about topology to its new client. After the whole topology has been advertised, the synchronization is over. For the listener, the synchronization is over when the RIB receives End-of-RIB (EOR) messages. There is a special EOR message for each AFI (Address Family Identifier).

- IPv4 EOR is an empty Update message.
- Ipv6 EOR is an Update message with empty MP_UNREACH attribute where AFI and SAFI (Subsequent Address Family Identifier) are set to Ipv6. OpenDaylight also supports EOR for IPv4 in this format.
- Linkstate EOR is an Update message with empty MP_UNREACH attribute where AFI and SAFI are set to Linkstate.

For BGP connections, where both peers support graceful restart, the EORs are sent by the BGP speaker and are redirected to RIB, where the specific AFI/SAFI table is set to *true*. Without graceful restart, the messages are generated by OpenDaylight itself and sent after second keepalive for each AFI/SAFI. This is done in BGPSynchronization.

Peers

BGPPeer has various meanings. If you configure BGP listener, *BGPPeer* represents the BGP listener itself. If you are configuring BGP speaker, you need to provide a list of peers, that are allowed to connect to this speaker. Unknown peer represents, in this case, a peer that is allowed to be refused. *BGPPeer* represents in this case peer, that is supposed to connect to your speaker. *BGPPeer* is stored in BGPPeerRegistry. This registry controls the number of sessions. Our strict implementation limits sessions to one per peer.

ApplicationPeer is a special case of peer, that has it's own RIB. This RIB is populated from RESTCONF. The RIB is synchronized with default BGP RIB. Incoming routes to the default RIB are treated in the same way as they were from a BGP peer (speaker or listener) in the network.

RIB handling

RIB (Route Information Base) is defined as a concept in RFC 4271. RFC does not define how it should be implemented. In our implementation, the routes are stored in the MD-SAL datastore. There are four supported routes - *Ipv4Routes*, *Ipv6Routes*, *LinkstateRoutes* and *FlowspecRoutes*.

Each route type needs to provide a RIBSupport.java implementation. *RIBSupport* tells RIB how to parse binding-aware data (BGP Update message) to binding-independent (datastore format).

Following picture describes the data flow from BGP message that is sent to *BGPPeer* to datastore and various types of RIB.

AdjRibInWriter - represents the first step in putting data to datastore. This writer is notified whenever a peer receives an Update message. The message is transformed into binding-independent format and pushed into datastore to *adj-rib-in*. This RIB is associated with a peer.

EffectiveRibInWriter - this writer is notified whenever *adj-rib-in* is updated. It applies all configured import policies to the routes and stores them in *effective-rib-in*. This RIB is also associated with a peer.

LocRibWriter - this writer is notified whenever **any** *effective-rib-in* is updated (in any peer). Performs best path selection filtering and stores the routes in *loc-rib*. It also determines which routes need to be advertised and fills in *adj-rib-out* that is per peer as well.

AdjRibOutListener - listens for changes in *adj-rib-out*, transforms the routes into BGPUpdate messages and sends them to its associated peer.

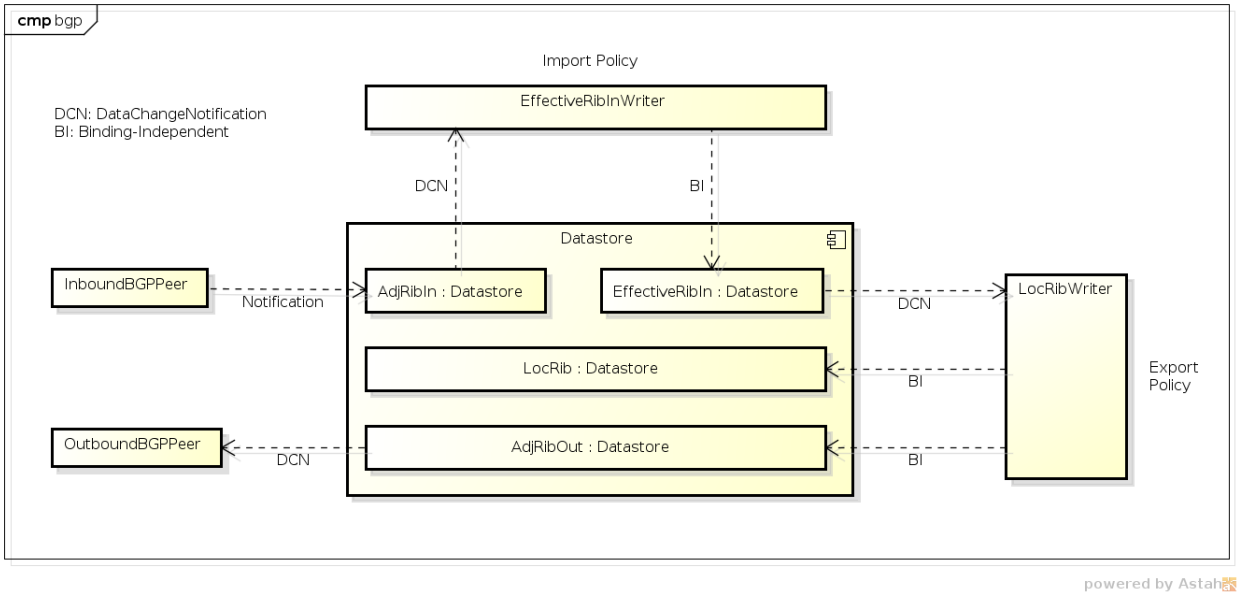


Fig. 2.22: RIB

BGP inet

This module contains only one YANG model bgp-inet.yang that summarizes the ipv4 and ipv6 extensions to RIB routes and BGP messages.

BGP flowspec

BGP flowspec is a module that implements RFC 5575 for IPv4 AFI and draft-ietf-idr-flow-spec-v6-06 for IPv6 AFI. The RFC defines an extension to BGP in form of a new subsequent address family, NLRI and extended communities. All of those are defined in the bgp-flowspec.yang model. In addition to generated sources, the module contains parsers for newly defined elements and RIBSupport for flowspec-routes. The route key of flowspec routes is a string representing human-readable flowspec request.

BGP linkstate

BGP linkstate is a module that implements draft-ietf-idr-ls-distribution version 04. The draft defines an extension to BGP in form of a new address family, subsequent address family, NLRI and path attribute. All of those are defined in the bgp-linkstate.yang model. In addition to generated sources, the module contains LinkstateAttributeParser, LinkstateNlriParser, activators for both, parser and RIB, and RIBSupport handler for linkstate address family. As each route needs a key, in case of linkstate, the route key is defined as a binary string, containing all the NLRI serialized to byte format. The BGP linkstate extension also supports distribution of MPLS TE state as defined in draft-ietf-idr-te-lsp-distribution-03, extension for Segment Routing draft-gredler-idr-bgp-ls-segment-routing-ext-00 and Segment Routing Egress Peer Engineering draft-ietf-idr-bgpls-segment-routing-epe-02.

BGP labeled-unicast

BGP labeled unicast is a module that implements RFC 3107. The RFC defines an extension to the BGP MP to carry Label Mapping Information as a part of the NLRI. The AFI indicates, as usual, the address family of the associated

route. The fact that the NLRI contains a label is indicated by using SAFI value 4. All of those are defined in bgp-labeled-unicast.yang model. In addition to the generated sources, the module contains new NLRI codec and RIBSupport. The route key is defined as a binary, where whole NLRI information is encoded.

BGP topology provider

BGP data besides RIB, is stored in network-topology view. The format of how the data is displayed there conforms to draft-clemm-netmod-yang-network-topo.

API Reference Documentation

Javadocs are generated while creating mvn:site and they are located in target/ directory in each module.

BGP Monitoring Protocol Developer Guide

Overview

This section provides an overview of **feature odl-bgpcep-bmp**. This feature will install everything needed for BMP (BGP Monitoring Protocol) including establishing the connection, processing messages, storing information about monitored routers, peers and their Adj-RIB-In (unprocessed routing information) and Post-Policy Adj-RIB-In and displaying data in BGP RIBs overview. The OpenDaylight BMP plugin plays the role of a monitoring station.

Key APIs and Interfaces

Session handling

32-bmp.xml defines only bmp-dispatcher the parser should be using (global-bmp-extensions).

```
<module>
 <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:bmp:impl">
↪prefix:bmp-dispatcher-impl</type>
 <name>global-bmp-dispatcher</name>
  <bmp-extensions>
   <type xmlns:bmp-spi="urn:opendaylight:params:xml:ns:yang:controller:bmp:spi">bmp-
↪spi:extensions</type>
   <name>global-bmp-extensions</name>
  </bmp-extensions>
  <boss-group>
   <type xmlns:netty="urn:opendaylight:params:xml:ns:yang:controller:netty">
↪netty:netty-threadgroup</type>
   <name>global-boss-group</name>
  </boss-group>
  <worker-group>
   <type xmlns:netty="urn:opendaylight:params:xml:ns:yang:controller:netty">
↪netty:netty-threadgroup</type>
   <name>global-worker-group</name>
 </worker-group>
</module>
```

For user configuration of BMP, check User Guide.

Parser

The base BMP parser includes messages and attributes from https://tools.ietf.org/html/draft-ietf-grow-bmp-15

Registration

All parsers and serializers need to be registered into *Extension provider*. This *Extension provider* is configured in initial configuration of the parser (*32-bmp.xml*).

```
<module>
 <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:bmp:spi">
↪prefix:bmp-extensions-impl</type>
 <name>global-bmp-extensions</name>
 <extension>
  <type xmlns:bmp-spi="urn:opendaylight:params:xml:ns:yang:controller:bmp:spi">bmp-
↪spi:extension</type>
  <name>bmp-parser-base</name>
 </extension>
</module>
```

- *bmp-parser-base* - will register parsers and serializers implemented in bmp-impl module

Parsing

Parsing of BMP elements is mostly done equally to BGP. Some of the BMP messages includes wrapped BGP messages.

BMP Monitoring Station

The BMP application (Monitoring Station) serves as message processor incoming from monitored routers. The processed message is transformed and relevant information is stored. Route information is stored in a BGP RIB data structure.

BMP data is displayed only through one URL that is accessible from the base BMP URL:

'http://<controllerIP>:8181/restconf/operational/bmp-monitor:bmp-monitor <http://<controllerIP>:8181/restconf/operational/bmp-monitor:bmp-monitor>'__

Each Monitor station will be displayed and it may contains multiple monitored routers and peers within:

```
<bmp-monitor xmlns="urn:opendaylight:params:xml:ns:yang:bmp-monitor">
 <monitor>
 <monitor-id>example-bmp-monitor</monitor-id>
  <router>
  <router-id>127.0.0.11</router-id>
   <status>up</status>
   <peer>
    <peer-id>20.20.20.20</peer-id>
    <as>72</as>
    <type>global</type>
    <peer-session>
     <remote-port>5000</remote-port>
     <timestamp-sec>5</timestamp-sec>
     <status>up</status>
```

```
     <local-address>10.10.10.10</local-address>
     <local-port>220</local-port>
    </peer-session>
    <pre-policy-rib>
     <tables>
      <afi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:ipv4-address-
↪family</afi>
      <safi xmlns:x="urn:opendaylight:params:xml:ns:yang:bgp-types">x:unicast-
↪subsequent-address-family</safi>
      <ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-inet">
       <ipv4-route>
        <prefix>10.10.10.0/24</prefix>
         <attributes>
          ...
         </attributes>
       </ipv4-route>
      </ipv4-routes>
      <attributes>
       <uptodate>true</uptodate>
      </attributes>
     </tables>
    </pre-policy-rib>
    <address>10.10.10.10</address>
    <post-policy-rib>
     ...
    </post-policy-rib>
    <bgp-id>20.20.20.20</bgp-id>
    <stats>
     <timestamp-sec>5</timestamp-sec>
     <invalidated-cluster-list-loop>53</invalidated-cluster-list-loop>
     <duplicate-prefix-advertisements>16</duplicate-prefix-advertisements>
     <loc-rib-routes>100</loc-rib-routes>
     <duplicate-withdraws>11</duplicate-withdraws>
     <invalidated-as-confed-loop>55</invalidated-as-confed-loop>
     <adj-ribs-in-routes>10</adj-ribs-in-routes>
     <invalidated-as-path-loop>66</invalidated-as-path-loop>
     <invalidated-originator-id>70</invalidated-originator-id>
     <rejected-prefixes>8</rejected-prefixes>
    </stats>
   </peer>
   <name>name</name>
   <description>description</description>
   <info>some info;</info>
  </router>
 </monitor>
</bmp-monitor>
</source>
```

API Reference Documentation

Javadocs are generated while creating mvn:site and they are located in target/ directory in each module.

BIER Developer Guide

BIER Architecture

- **Channel**
  - Channel (multicast flow) configuration and deploying information management.
- **Common**
  - Common YANG models collection.
- **Drivers**
  - South-bound NETCONF interface for BIER, it has implemented standard interface (ietf-bier). If your BFR's NETCONF interface is Non-standard, you should add your own interface for driver.
- **Sbi-Adapter**
  - Adapter for different BIER south-bound NETCONF interfaces.
- **Service**
  - Major processor function for BIER.
- **Bierman**
  - BIER topology management, and BIER information (BIER, BIER-TE, lable info) configuration.
- **Pce**
  - Path computation element for BIER-TE.
- **Bierapp**
  - BIER UI, show topology and configure BIER/BIER-TE and channel.

APIs in BIER

The sections below give details about the configuration settings for the components that can be configured.

BIER Information Manager

API Description

- bier/bierman/api/src/main/yang/bier-topology-api.yang
  - **load-topology**
    - Load BIER topology, and list all topo-name in all BIER topologies.
  - **configure-domain**
    - Configure domain in given BIER topology.
  - **configure-subdomain**
    - Configure sub-domain in given BIER domain and topology.
  - **delete-domain**
    - Delete given domain in given topology.

  - **delete-subdomain**
    - Delete given sub-domain in given domain and topology.
  - **query-topology**
    - Query given topology in BIER topology, and then display this topology's detail, such as information of node and link.
  - **query-node**
    - Query given nodes in given topology, and then display these nodes' detail, such as information of node-name, router-id, termination-point list, BIER domain and sub-domain list, etc.
  - **query-link**
    - Query given link in given topology, and then display this link's detail.
  - **query-domain**
    - Query domain in given BIER topology, and then display the domain-id list.
  - **query-subdomain**
    - Query sub-domain in given domain and given topology, and then display the sub-domain-id list.
  - **query-subdomain-node**
    - Query nodes which have been assigned to given sub-domain and domain in given topology, and then display these nodes' details.
  - **query-subdomain-link**
    - Query links which have been assigned to given sub-domain and domain in given topology, and then display these links' details.
  - **query-te-subdomain-node**
    - Query te-nodes which have been assigned to given sub-domain and domain in given topology, and then display these te-nodes' details.
  - **query-te-subdomain-link**
    - Query te-links which have been assigned to given sub-domain and domain in given topology, and then display these te-links' details.
- bier/bierman/api/src/main/yang/bier-config-api.yang
  - **configure-node**
    - Configure node information in given topology, which defined in ietf-bier, such as domains, sub-domains, bitstringlength, bfr-id, encapsulation-type, etc.
  - **delete-node**
    - Delete given node which be assigned to given sub-domain and domain in given topology.
  - **delete-ipv4**
    - Delete bier mapping entry of ipv4.
  - **delete-ipv6**
    - Delete bier mapping entry of ipv6.
- bier/bierman/api/src/main/yang/bier-te-config-api.yang
  - **configure-te-node**

        * Configure adjancency information for node, such as domains, sub-domains, si, bitstringlength, tpid, bitposition, etc.
    - **configure-te-label**
        * Configure BIER-TE label range for node.
    - **delete-te-babel**
        * Delete BIER-TE label range of node.
    - **delete-te-bsl**
        * Delete BIER-TE bitstringlength, including all SIs which belongs to this bitstringlenght.
    - **delete-te-si**
        * Delete BIER-TE SI, including all bitpositions which belongs to this SI.
    - **delete-te-bp**
        * Delete BIER-TE bitposition of an adjancency.

Parameters Description

- **topology-id**
    - BIER topology identifier.
- **node-id**
    - Node identifier in network topology.
- **latitude**
    - Node's latitude, default value is 0.
- **longitude**
    - Node's longitude, default value is 0.
- **tp-id**
    - Termination point identifier.
- **domain-id**
    - BIER domain identifier.
- **encapsulation-type**
    - Base identity for BIER encapsulation. Default value is "bier-encapsulation-mpls".
- **bitstringlength**
    - The bitstringlength type for imposition mode. It's value can be chosen from 64, 128, 256, 512, 1024, 2048, and 4096.
    - The BitStringLength ("Imposition BitStringLength") and sub-domain ("Imposition sub-domain") to use when it imposes (as a BFIR) a BIER encapsulation on a particular set of packets.
- **bfr-id**
    - BIER bfr identifier. BFR-id is a number in the range [1, 65535].

  - Bfr-id is unique within the sub-domain. A BFR-id is a small unstructured positive integer. For instance, if a particular BIER sub-domain contains 1, 374 BFRs, each one could be given a BFR-id in the range 1-1374.
  - If a given BFR belongs to more than one sub-domain, it may (though it need not) have a different BFR-id for each sub-domain.

- **ipv4-bfr-prefix**
  - BIER BFR IPv4 prefix.
  - A BFR's BFR-Prefix MUST be an IP address (either IPv4 or IPv6) of the BFR, and MUST be unique and routable within the BIER domain. It is RECOMMENDED that the BFR-prefix be a loopback address of the BFR. Two BFRs in the same BIER domain MUST NOT be assigned the same BFR-Prefix. Note that a BFR in a given BIER domain has the same BFR-prefix in all the sub-domains of that BIER domain.
- **ipv6-bfr-prefix**
  - BIER BFR IPv6 prefix.
- **sub-domain-id**
  - Sub-domain identifier. Each sub-domain is identified by a sub-domain-id in the range [0, 255].
  - A BIER domain may contain one or more sub-domains. Each BIER domain MUST contain at least one sub-domain, the "default sub-domain" (also denoted "sub-domain zero"). If a BIER domain contains more than one sub-domain, each BFR in the domain MUST be provisioned to know the set of sub-domains to which it belongs.
- **igp-type**
  - The IGP type. Enum type contains OSPF and ISIS.
- **mt-id**
  - Multi-topology associated with BIER sub-domain.
- **bitstringlength**
  - Disposition bitstringlength.
  - The BitStringLengths ("Disposition BitStringLengths") that it will process when (as a BFR or BFER) it receives packets from a particular sub-domain.
- **bier-mpls-label-base**
  - BIER mpls-label, range in [0, 1048575].
- **bier-mpls-label-range-size**
  - BIER mpls-label range size.
- **link-id**
  - The identifier of a link in the topology.
  - A link is specific to a topology to which it belongs.
- **source-node**
  - Source node identifier, must be in same topology.
- **source-tp**
  - Termination point within source node that terminates the link.
- **dest-node**

  - Destination node identifier and must be in same topology.
- **dest-tp**
  - Termination point within destination node that terminates the link.
- **delay**
  - The link delay, default value is 0.
- **loss**
  - The number of packet loss on the link and default value is 0.

Channel Manager

API Description

- bier/channel/api/src/main/yang/bier-channel-api.yang
  - **get-channel**
    - Display all channel's names in given BIER topology.
  - **query-channel**
    - Query specific channel in given topology and display this channel's information (multicast flow information and related BFIR,BFER information).
  - **add-channel**
    - Create channel with multicast information in given BIER topology.
  - **modify-channel**
    - Modify the channel's information which created above.
  - **remove-channel**
    - Remove given channel in given topology.
  - **deploy-channel**
    - Deploy channel, and configure BFIR and BFERs about this multicast flow in given topology.

Parameters Description

- **topology-id**
  - BIER topology identifier.
- **channel-name**
  - BIER channel (multicast flow information) name.
- **src-ip**
  - The IPv4 of multicast source. The value set to zero means that the receiver interests in all source that relevant to one group.
- **dst-group**
  - The IPv4 of multicast group.
- **domain-id**

  - BIER domain identifier.
- **sub-domain-id**
  - BIER sub-domain identifier.
- **source-wildcard**
  - The wildcard information of source, in the range [1, 32].
- **group-wildcard**
  - The wildcard information of multi-cast group, in the range [1, 32].
- **ingress-node**
  - BFIR (Bit-Forwarding Ingress Router).
- **ingress-bfr-id**
  - The bfr-id of BRIR.
- **egress-node**
  - BFER (Bit-Forwarding Egress Router).
- **egress-bfr-id**
  - The bfr-id of BRER.
- **bier-forwarding-type**
  - The forwarding type, enum type contains BIER and BIER-TE.

Note: For more information about BIER terminology, see YANG Data Model for BIER Protocol.

Sample Configurations

1. Configure Domain And Sub-domain

1.1. Configure Domain

REST API : *POST /restconf/operations/bier-topology-api:configure-domain*

Sample JSON Data

```
{
     "input": {
        "topo-id": " bier-topo" ,
        "domain ":[
           {
              "domain-id": " 1",
           },
           {
             "domain-id": " 2",
           }
        ]
    }
}
```

1.2. Configure Sub-domain

REST API : *POST /restconf/operations/bier-topology-api:configure-subdomain*

Sample JSON Data

```
{
    "input": {
        "topo-id": " bier-topo" ,
        "domain-id":" 1",
        "sub-domain":[
            {
                "sub-domain-id":" 0",
            },
            {
                "sub-domain-id":"1",
            }
        ]
    }
}
```

2. Configure Node

2.1. Configure BIER Parameters

REST API : *POST /restconf/operations/bier-config-api:configure-node*

Sample JSON Data

```
{
    "input": {
        "topology-id": "bier-topo",
        "node-id": "node1",
        "domain": [
            {
                "domain-id": "2",
                "bier-global": {
                    "sub-domain": [
                        {
                            "sub-domain-id": "0",
                            "igp-type": "ISIS",
                            "mt-id": "1",
                            "bfr-id": "3",
                            "bitstringlength": "64-bit",
                            "af": {
                                "ipv4": [
                                    {
                                        "bitstringlength": "64",
                                        "bier-mpls-label-base": "56",
                                        "bier-mpls-label-range-size": "100"
                                    }
                                ]
                            }
                        }
                    ],
                    "encapsulation-type": "bier-encapsulation-mpls",
```

```
                    "bitstringlength": "64-bit",
                    "bfr-id": "33",
                    "ipv4-bfr-prefix": "192.168.1.1/24",
                    "ipv6-bfr-prefix": "1030:0:0:0:C9B4:FF12:48AA:1A2B/60"
                }
            }
        ]
    }
}
```

2.2. Configure BIER-TE label

REST API : *POST /restconf/operations/bier-te-config-api:configure-te-label*

Sample JSON Data

```
{
    "input": {
        "topology-id": "bier-topo",
        "node-id": "node1",
        "label-base": "100",
        "label-range-size": "20"
    }
}
```

2.3. Configure BIER-TE Parameters

REST API : *POST /restconf/operations/bier-te-config-api:configure-te-node*

Sample JSON Data

```
{
    "input": {
        "topology-id": "bier-topo",
        "node-id": "node1",
        "te-domain": [
            {
                "domain-id": "1",
                "te-sub-domain": [
                    {
                        "sub-domain-id": "0",
                        "te-bsl": [
                            {
                                "bitstringlength": "64-bit",
                                "te-si": [
                                    {
                                        "si": "1",
                                        "te-bp": [
                                            {
                                                "tp-id":"tp1",
                                                "bitposition": "1"
                                            }
                                        ]
                                    }
                                ]
```

```
                                }
                            ]
                        }
                    ]
                }
            ]
        }
}
```

3. Query BIER Topology Information

3.1. Load Topology

REST API : *POST /restconf/operations/bier-topology-api:load-topology*

no request body.

3.2. Query Topology

REST API : *POST /restconf/operations/bier-topology-api:query-topology*

Sample JSON Data

```
{
    "input": {
        "topo-id": "bier-topo"
    }
}
```

3.3. Query BIER Node

REST API : *POST /restconf/operations/bier-topology-api:query-node*

Sample JSON Data

```
{
    "input": {
         "topo-id": "bier-topo",
         "node-id": "node1"
     }
}
```

3.4. Query BIER Link

REST API : *POST /restconf/operations/bier-topology-api:query-link*

Sample JSON Data

```
{
    "input": {
         "topo-id": "bier-topo",
         "node-id": "node1"
```

```
      }
}
```

3.5. Query Domain

REST API : *POST /restconf/operations/bier-topology-api:query-domain*

Sample JSON Data

```
{
    "input": {
         "topo-id": "bier-topo"
     }
}
```

3.6. Query Sub-domain

REST API : *POST /restconf/operations/bier-topology-api:query-subdomain*

Sample JSON Data

```
{
    "input": {
         "topo-id": "bier-topo",
         "domain-id": "1"
     }
}
```

3.7. Query Sub-domain Node

REST API : *POST /restconf/operations/bier-topology-api:query-subdomain-node*

Sample JSON Data

```
{
    "input": {
        "topology-id": "bier-topo",
        "domain-id": "1",
        "sub-domain-id": "0"
    }
}
```

3.8. Query Sub-domain Link

REST API : *POST /restconf/operations/bier-topology-api:query-subdomain-link*

Sample JSON Data

```
{
    "input": {
        "topology-id": "bier-topo",
        "domain-id": "1",
```

```
        "sub-domain-id": "0"
    }
}
```

3.9. Query BIER-TE Sub-domain Node

REST API : *POST /restconf/operations/bier-topology-api:query-te-subdomain-node*

Sample JSON Data

```
{
    "input": {
        "topology-id": "bier-topo",
        "domain-id": "1",
        "sub-domain-id": "0"
    }
}
```

3.10. Query BIER-TE Sub-domain Link

REST API : *POST /restconf/operations/bier-topology-api:query-te-subdomain-link*

Sample JSON Data

```
{
    "input": {
        "topology-id": "bier-topo",
        "domain-id": "1",
        "sub-domain-id": "0"
    }
}
```

4. BIER Channel Configuration

4.1. Configure Channel

REST API : *POST /restconf/operations/bier-channel-api:add-channel*

Sample JSON Data

```
{
    "input": {
        "topology-id": "bier-topo",
        "name": "channel-1",
        "src-ip": "1.1.1.1",
        "dst-group": "224.1.1.1",
        "domain-id": "1",
        "sub-domain-id": "11",
        "source-wildcard": "24",
        "group-wildcard": "30"
    }
}
```

4.2. Modify Channel

REST API : *POST /restconf/operations/bier-channel-api:modify-channel*

Sample JSON Data

```
{
    "input": {
        "topology-id": "bier-topo",
        "name": "channel-1",
        "src-ip": "2.2.2.2",
        "dst-group": "225.1.1.1",
        "domain-id": "1",
        "sub-domain-id": "11",
        "source-wildcard": "24",
        "group-wildcard": "30"
    }
}
```

5. Deploy Channel

REST API : *POST /restconf/operations/bier-channel-api:deploy-channel*

Sample JSON Data

```
{
    "input": {
        "topology-id": "bier-topo",
        "channel-name": "channel-1",
        "bier-forwarding-type":"bier-te"
        "ingress-node": "node1",
        "egress-node": [
            {
                "node-id": "node2"
            },
            {
                "node-id": "node3"
            }
        ]
    }
}
```

6. Query Channel Information

6.1. Get Channel

REST API : *POST /restconf/operations/bier-channel-api:get-channel*

Sample JSON Data

```
{
    "input": {
        "topology-id": "bier-topo"
    }
}
```

6.2. Query Channel

REST API : *POST /restconf/operations/bier-channel-api:query-channel*

Sample JSON Data

```
{
    "input": {
        "topology-id": "bier-topo",
        "channel-name": [
            "channel-1",
            "channel-2"
        ]
    }
}
```

7. Remove Channel

REST API : *POST /restconf/operations/bier-channel-api:remove-channel*

Sample JSON Data

```
{
    "input": {
        "topology-id": "bier-topo",
        "channel-name": "channel-1"
    }
}
```

8. Delete BIER and BIER-TE Configuration

8.1. Delete BIER Node

REST API : *POST /restconf/operations/bier-config-api:delete-node*

Sample JSON Data

```
{
     "input": {
             "topo-id": "bier-topo",
             "node-id": "node3",
             "domain-id": "1",
             "subdomain-id": "0"
    }
}
```

8.2. Delete IPv4 of BIER Node

REST API : *POST /restconf/operations/bier-config-api:delete-ipv4*

Sample JSON Data

```
{
    input: {
        "topology-id": "bier-topo",
        "domain-id": "1",
        "sub-domain-id": "0",
        "node-id": "node1",
        "ipv4": {
            "bier-mpls-label-base": "10",
            "bier-mpls-label-range-size": "16",
            "bitstringlength": "64"
        }
    }
}
```

8.3. Delete IPv6 of BIER Node

REST API : *POST /restconf/operations/bier-config-api:delete-ipv6*

Sample JSON Data

```
{
    input: {
        "topology-id": "bier-topo",
        "domain-id": "1",
        "sub-domain-id": "0",
        "node-id": "node1",
        "ipv6": {
            "bier-mpls-label-base": "10",
            "bier-mpls-label-range-size": "16",
            "bitstringlength": "64"
        }
    }
}
```

8.4. Delete BIER-TE BSL

REST API : *POST /restconf/operations/bier-te-config-api:delete-te-bsl*

Sample JSON Data

```
{
    input:{
        "topology-id": "bier-topo",
        "node-id": "node1",
        "domain-id": "1",
        "sub-domain-id": "0",
        "bitstringlength": "64-bit"
    }
}
```

8.5. Delete BIER-TE SI

REST API : *POST /restconf/operations/bier-te-config-api:delete-te-si*

Sample JSON Data

```
{
    input:{
        "topology-id": "bier-topo",
        "node-id": "node1",
        "domain-id": "1",
        "sub-domain-id": "0",
        "bitstringlength": "64-bit",
        "si": "1"
    }
}
```

8.6. Delete BIER-TE BP

REST API : *POST /restconf/operations/bier-te-config-api:delete-te-bp*

Sample JSON Data

```
{
    input: {
        "topology-id": "bier-topo",
        "node-id": "node1",
        "domain-id": "1",
        "sub-domain-id": "0",
        "bitstringlength": "64-bit",
        "si": "1",
        "tp-id": "tp1"
    }
}
```

8.7. Delete BIER-TE Label

REST API : *POST /restconf/operations/bier-te-config-api:delete-te-label*

Sample JSON Data

```
{
     "input": {
             "topo-id": "bier-topo",
             "node-id": "node1"
    }
}
```

8.8. Delete Sub-domain

REST API : *POST /restconf/operations/bier-topology-api:delete-subdomian*

Sample JSON Data

```
{
     "input": {
             "topo-id": "bier-topo",
             "domain-id": "1",
```

```
            "subdomain-id": "0"
    }
}
```

8.9. Delete Domain

REST API : *POST /restconf/operations/bier-topology-api:delete-domian*

Sample JSON Data

```
{
     "input": {
            "topo-id": "bier-topo",
            "domain-id": "1"
    }
}
```

CAPWAP Developer Guide

Overview

The Control And Provisioning of Wireless Access Points (CAPWAP) plugin project aims to provide new southbound interface for controller to be able to monitor and manage CAPWAP compliant wireless termination point (WTP) network devices. The CAPWAP feature will provide REST based northbound APIs.

CAPWAP Architecture

The CAPWAP feature is implemented as an MD-SAL based provider module, which helps discover WTP devices and update their states in the MD-SAL operational datastore.

CAPWAP APIs and Interfaces

This section describes the APIs for interacting with the CAPWAP plugin.

Discovered WTPs

The CAPWAP project maintains list of discovered CAPWAP WTPs that is YANG-based in MD-SAL. These models are available via RESTCONF.

- Name: Discovered-WTPs
- URL: http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operational/capwap-impl:capwap-ac-root/
- Description: Displays list of discovered WTPs and their basic attributes

API Reference Documentation

Go to http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/apidoc/explorer/index.html, sign in, and expand the capwap-impl panel. From there, users can execute various API calls to test their CAPWAP deployment.

Cardinal: OpenDaylight Monitoring as a Service

Overview

Cardinal (OpenDaylight Monitoring as a Service) enables OpenDaylight and the underlying software defined network to be remotely monitored by deployed Network Management Systems (NMS) or Analytics suite. In the Boron release, Cardinal adds:

1. OpenDaylight MIB.
2. Enable ODL diagnostics/monitoring to be exposed across SNMP (v2c, v3) and REST north-bound.
3. Extend ODL System health, Karaf parameter and feature info, ODL plugin scalability and network parameters.
4. Support autonomous notifications (SNMP Traps).

Cardinal Architecture

The Cardinal architecture can be found at the below link:

https://wiki.opendaylight.org/images/8/89/Cardinal-ODL_Monitoring_as_a_Service_V2.pdf

Key APIs and Interfaces

There are 6 main APIs for requesting snmpget request of the Karaf info, System info, Openflow devices and Netconf Devices. To expose these APIs, it assumes that you already have the `odl-cardinal` and `odl-restconf` features installed. You can do that by entering the following at the Karaf console:

```
feature:install odl-cardinal
feature:install odl-restconf-all
feature:install odl-l2switch-switch
feature:install odl-netconf-all
feature:install odl-netconf-connector-all
feature:install odl-netconf-mdsal
feature:install cardinal-features4
feature:install odl-cardinal-api
feature:install odl-cardinal-ui
feature:install odl-cardinal-rest
```

System Info APIs

Open the REST interface and using the basic authentication, execute REST APIs for system info as:

```
http://localhost:8181/restconf/operational/cardinal:CardinalSystemInfo/
```

You should get the response code of the same as 200 OK with the following output as:

```
{
  "CardinalSystemInfo": {
    "odlSystemMemUsage": " 9",
    "odlSystemSysInfo": " OpenDaylight Node Information",
    "odlSystemOdlUptime": " 00:29",
    "odlSystemCpuUsage": " 271",
    "odlSystemHostAddress": " Address of the Host should come up"
  }
}
```


Karaf Info APIs

Open the REST interface and using the basic authentication, execute REST APIs for system info as:

```
http://localhost:8181/restconf/operational/cardinal-karaf:CardinalKarafInfo/
```

You should get the response code of the same as 200 OK with the following output as:


```
  {
  "CardinalKarafInfo": {
    "odlKarafBundleListActive1": " org.ops4j.pax.url.mvn_2.4.5 [1]",
    "odlKarafBundleListActive2": " org.ops4j.pax.url.wrap_2.4.5 [2]",
    "odlKarafBundleListActive3": " org.ops4j.pax.logging.pax-logging-api_1.8.4 [3]",
    "odlKarafBundleListActive4": " org.ops4j.pax.logging.pax-logging-service_1.8.4 [4]
↪",
    "odlKarafBundleListActive5": " org.apache.karaf.service.guard_3.0.6 [5]",
    "odlKarafBundleListActive6": " org.apache.felix.configadmin_1.8.4 [6]",
    "odlKarafBundleListActive7": " org.apache.felix.fileinstall_3.5.2 [7]",
    "odlKarafBundleListActive8": " org.objectweb.asm.all_5.0.3 [8]",
    "odlKarafBundleListActive9": " org.apache.aries.util_1.1.1 [9]",
    "odlKarafBundleListActive10": " org.apache.aries.proxy.api_1.0.1 [10]",
    "odlKarafBundleListInstalled1": " org.ops4j.pax.url.mvn_2.4.5 [1]",
    "odlKarafBundleListInstalled2": " org.ops4j.pax.url.wrap_2.4.5 [2]",
    "odlKarafBundleListInstalled3": " org.ops4j.pax.logging.pax-logging-api_1.8.4 [3]
↪",
    "odlKarafBundleListInstalled4": " org.ops4j.pax.logging.pax-logging-service_1.8.4␣
↪[4]",
    "odlKarafBundleListInstalled5": " org.apache.karaf.service.guard_3.0.6 [5]",
    "odlKarafFeatureListInstalled1": " config",
    "odlKarafFeatureListInstalled2": " region",
    "odlKarafFeatureListInstalled3": " package",
    "odlKarafFeatureListInstalled4": " http",
    "odlKarafFeatureListInstalled5": " war",
    "odlKarafFeatureListInstalled6": " kar",
    "odlKarafFeatureListInstalled7": " ssh",
    "odlKarafFeatureListInstalled8": " management",
    "odlKarafFeatureListInstalled9": " odl-netty",
    "odlKarafFeatureListInstalled10": " odl-lmax",
    "odlKarafBundleListResolved1": " org.ops4j.pax.url.mvn_2.4.5 [1]",
    "odlKarafBundleListResolved2": " org.ops4j.pax.url.wrap_2.4.5 [2]",
    "odlKarafBundleListResolved3": " org.ops4j.pax.logging.pax-logging-api_1.8.4 [3]",
    "odlKarafBundleListResolved4": " org.ops4j.pax.logging.pax-logging-service_1.8.4␣
↪[4]",
    "odlKarafBundleListResolved5": " org.apache.karaf.service.guard_3.0.6 [5]",
    "odlKarafFeatureListUnInstalled1": " aries-annotation",
    "odlKarafFeatureListUnInstalled2": " wrapper",
```

```
    "odlKarafFeatureListUnInstalled3": " service-wrapper",
    "odlKarafFeatureListUnInstalled4": " obr",
    "odlKarafFeatureListUnInstalled5": " http-whiteboard",
    "odlKarafFeatureListUnInstalled6": " jetty",
    "odlKarafFeatureListUnInstalled7": " webconsole",
    "odlKarafFeatureListUnInstalled8": " scheduler",
    "odlKarafFeatureListUnInstalled9": " eventadmin",
    "odlKarafFeatureListUnInstalled10": " jasypt-encryption"
  }
}
```


OpenFlowInfo Apis

Open the REST interface and using the basic authentication, execute REST APIs for system info as:

http://localhost:8181/restconf/operational/cardinal-openflow:Devices

You should get the response code of the same as 200 OK with the following output as:


```
{
    "Devices": {
        "openflow": [
            {
                "macAddress": "6a:80:ef:06:d3:46",
                "status": "Connected",
                "flowStats": " ",
                "interface": "s1",
                "manufacturer": "Nicira, Inc.",
                "nodeName": "openflow:1:LOCAL",
                "meterStats": " "
            },
            {
                "macAddress": "32:56:c7:41:5d:9a",
                "status": "Connected",
                "flowStats": " ",
                "interface": "s2-eth2",
                "manufacturer": "Nicira, Inc.",
                "nodeName": "openflow:2:2",
                "meterStats": " "
            },
            {
                "macAddress": "36:a8:3b:fe:e2:21",
                "status": "Connected",
                "flowStats": " ",
                "interface": "s3-eth1",
                "manufacturer": "Nicira, Inc.",
                "nodeName": "openflow:3:1",
                "meterStats": " "
            }
        ]
    }
}
```


Configuration for Netconf Devices:-

1. To configure or update a netconf-connector via topology you need to send following request to Restconf:

```
Method: PUT
URI: http://localhost:8181/restconf/config/network-topology:network-topology/topology/
↪topology-netconf/node/new-netconf-device
Headers:
Accept: application/xml
Content-Type: application/xml
```

Payload:

```
<node xmlns="urn:TBD:params:xml:ns:yang:network-topology">
  <node-id>new-netconf-device</node-id>
  <host xmlns="urn:opendaylight:netconf-node-topology">127.0.0.1</host>
  <port xmlns="urn:opendaylight:netconf-node-topology">17830</port>
  <username xmlns="urn:opendaylight:netconf-node-topology">admin</username>
  <password xmlns="urn:opendaylight:netconf-node-topology">admin</password>
  <tcp-only xmlns="urn:opendaylight:netconf-node-topology">false</tcp-only>
  <keepalive-delay xmlns="urn:opendaylight:netconf-node-topology">0</keepalive-delay>
</node>
```

2. To delete a netconf connector issue a DELETE request to the following url: URI:http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/new-netconf-device

NetConf Info Apis Open the REST interface and using the basic authentication, execute REST APIs for system info as:

http://localhost:8181/restconf/operational/cardinal-netconf:Devices

You should get the response code of the same as 200 OK with the following output as:

```
{
    "Devices": {
        "netconf": [
            {
                "status": "connecting",
                "host": "127.0.0.1",
                "nodeId": "new-netconf-device1",
                "port": "17830"
            },
            {
                "status": "connecting",
                "host": "127.0.0.1",
                "nodeId": "new-netconf-device",
                "port": "17830"
            },
            {
                "status": "connecting",
                "host": "127.0.0.1",
                "nodeId": "controller-config",
                "port": "1830"
            }
        ]
    }
}
```

Controller

Overview

OpenDaylight Controller is Java-based, model-driven controller using YANG as its modeling language for various aspects of the system and applications and with its components serves as a base platform for other OpenDaylight applications.

The OpenDaylight Controller relies on the following technologies:

- **OSGI** - This framework is the back-end of OpenDaylight as it allows dynamically loading of bundles and packages JAR files, and binding bundles together for exchanging information.
- **Karaf** - Application container built on top of OSGI, which simplifies operational aspects of packaging and installing applications.
- **YANG** - a data modeling language used to model configuration and state data manipulated by the applications, remote procedure calls, and notifications.

The OpenDaylight Controller provides following model-driven subsystems as a foundation for Java applications:

- *Config Subsystem* - an activation, dependency-injection and configuration framework, which allows two-phase commits of configuration and dependency-injection, and allows for run-time rewiring.
- *MD-SAL* - messaging and data storage functionality for data, notifications and RPCs modeled by application developers. MD-SAL uses YANG as the modeling for both interface and data definitions, and provides a messaging and data-centric runtime for such services based on YANG modeling.
- **MD-SAL Clustering** - enables cluster support for core MD-SAL functionality and provides location-transparent accesss to YANG-modeled data.

The OpenDaylight Controller supports external access to applications and data using following model-driven protocols:

- **NETCONF** - XML-based RPC protocol, which provides abilities for client to invoke YANG-modeled RPCs, receive notifications and to read, modify and manipulate YANG modeled data.
- **RESTCONF** - HTTP-based protocol, which provides REST-like APIs to manipulate YANG modeled data and invoke YANG modeled RPCs, using XML or JSON as payload format.

MD-SAL Overview

The Model-Driven Service Adaptation Layer (MD-SAL) is message-bus inspired extensible middleware component that provides messaging and data storage functionality based on data and interface models defined by application developers (i.e. user-defined models).

The MD-SAL:

- Defines a **common-layer, concepts, data model building blocks and messaging patterns** and provides infrastructure / framework for applications and inter-application communication.
- Provide common support for user-defined transport and payload formats, including payload serialization and adaptation (e.g. binary, XML or JSON).

The MD-SAL uses **YANG** as the modeling language for both interface and data definitions, and provides a messaging and data-centric runtime for such services based on YANG modeling.

The MD-SAL provides two different API types (flavours):

- **MD-SAL Binding:** MD-SAL APIs which extensively uses APIs and classes generated from YANG models, which provides compile-time safety.
- **MD-SAL DOM:** (Document Object Model) APIs which uses DOM-like representation of data, which makes them more powerful, but provides less compile-time safety.

Note: Model-driven nature of the MD-SAL and **DOM**-based APIs allows for behind-the-scene API and payload type mediation and transformation to facilitate seamless communication between applications - this enables for other components and applications to provide connectors / expose different set of APIs and derive most of its functionality purely from models, which all existing code can benefit from without modification. For example **RESTCONF Connector** is an application built on top of MD-SAL and exposes YANG-modeled application APIs transparently via HTTP and adds support for XML and JSON payload type.

Basic concepts

Basic concepts are building blocks which are used by applications, and from which MD-SAL uses to define messaging patterns and to provide services and behavior based on developer-supplied YANG models.

Data Tree All state-related data are modeled and represented as data tree, with possibility to address any element / subtree

- **Operational Data Tree** - Reported state of the system, published by the providers using MD-SAL. Represents a feedback loop for applications to observe state of the network / system.
- **Configuration Data Tree** - Intended state of the system or network, populated by consumers, which expresses their intention.

Instance Identifier Unique identifier of node / subtree in data tree, which provides unambiguous information, how to reference and retrieve node / subtree from conceptual data trees.

Notification Asynchronous transient event which may be consumed by subscribers and they may act upon it

RPC asynchronous request-reply message pair, when request is triggered by consumer, send to the provider, which in future replies with reply message.

Note: In MD-SAL terminology, the term *RPC* is used to define the input and output for a procedure (function) that is to be provided by a provider, and mediated by the MD-SAL, that means it may not result in remote call.

Messaging Patterns

MD-SAL provides several messaging patterns using broker derived from basic concepts, which are intended to transfer YANG modeled data between applications to provide data-centric integration between applications instead of API-centric integration.

- **Unicast communication**
  - **Remote Procedure Calls** - unicast between consumer and provider, where consumer sends **request** message to provider, which asynchronously responds with **reply** message
- **Publish / Subscribe**
  - **Notifications** - multicast transient message which is published by provider and is delivered to subscribers
  - **Data Change Events** - multicast asynchronous event, which is sent by data broker if there is change in conceptual data tree, and is delivered to subscribers

- **Transactional access to Data Tree**
  - Transactional **reads** from conceptual **data tree** - read-only transactions with isolation from other running transactions.
  - Transactional **modification** to conceptual **data tree** - write transactions with isolation from other running transactions.
  - **Transaction chaining**

MD-SAL Data Transactions

MD-SAL **Data Broker** provides transactional access to conceptual **data trees** representing configuration and operational state.

Note: **Data tree** usually represents state of the modeled data, usually this is state of controller, applications and also external systems (network devices).

Transactions provide *stable and isolated view* from other currently running transactions. The state of running transaction and underlying data tree is not affected by other concurrently running transactions.

Write-Only Transaction provides only modification capabilities, but does not provide read capabilities. Write-only transaction is allocated using `newWriteOnlyTransaction()`.

> **Note:** This allows less state tracking for write-only transactions and allows MD-SAL Clustering to optimize internal representation of transaction in cluster.

Read-Write Transaction provides both read and write capabilities. It is allocated using `newReadWriteTransaction()`.

Read-Only Transaction provides stable read-only view based on current data tree. Read-only view is not affected by any subsequent write transactions. Read-only transaction is allocated using `newReadOnlyTransaction()`.

> **Note:** If an application needs to observe changes itself in data tree, it should use **data tree listeners** instead of read-only transactions and polling data tree.

Transactions may be allocated using the **data broker** itself or using **transaction chain**. In the case of **transaction chain**, the new allocated transaction is not based on current state of data tree, but rather on state introduced by previous transaction from the same chain, even if the commit for previous transaction has not yet occurred (but transaction was submitted).

Write-Only & Read-Write Transaction

Write-Only and Read-Write transactions provide modification capabilities for the conceptual data trees.

1. application allocates new transactions using `newWriteOnlyTransaction()` or `newReadWriteTransaction()`.
2. application *modifies data tree* using `put`, `merge` and/or `delete`.
3. application finishes transaction using `submit()`, which *seals transaction and submits* it to be processed.
4. application observes the result of the transaction commit using either blocking or asynchronous calls.

The **initial state** of the write transaction is a **stable snapshot** of the current data tree state captured when transaction was created and it's state and underlying data tree are not affected by other concurrently running transactions.

Write transactions are **isolated** from other concurrent write transactions. All *writes are local* to the transaction and represents only a **proposal of state change** for data tree and **are not visible** to any other concurrently running transactions (including read-only transactions).

The transaction *commit may fail* due to failing verification of data or concurrent transaction modifying and affected data in an incompatible way.

Modification of Data Tree

Write-only and read-write transaction provides following methods to modify data tree:

put

```
<T> void put(LogicalDatastoreType store, InstanceIdentifier<T> path, T data);
```

Stores a piece of data at a specified path. This acts as an **add / replace** operation, which is to say that whole subtree will be replaced by the specified data.

merge

```
<T> void merge(LogicalDatastoreType store, InstanceIdentifier<T> path, T data);
```

Merges a piece of data with the existing data at a specified path. Any **pre-existing data** which are not explicitly overwritten **will be preserved**. This means that if you store a container, its child subtrees will be merged.

delete

```
void delete(LogicalDatastoreType store, InstanceIdentifier<?> path);
```

Removes a whole subtree from a specified path.

Submitting transaction

Transaction is submitted to be processed and committed using following method:

```
CheckedFuture<Void,TransactionCommitFailedException> submit();
```

Applications publish the changes proposed in the transaction by calling `submit()` on the transaction. This **seals the transaction** (preventing any further writes using this transaction) and submits it to be processed and applied to global conceptual data tree. The `submit()` method does not block, but rather returns `ListenableFuture`, which will complete successfully once processing of transaction is finished and changes are applied to data tree. If **commit** of data failed, the future will fail with `TransactionFailedException`.

Application may listen on commit state asynchronously using `ListenableFuture`.

```
Futures.addCallback( writeTx.submit(), new FutureCallback<Void>() {
        public void onSuccess( Void result ) {
            LOG.debug("Transaction committed successfully.");
        }

        public void onFailure( Throwable t ) {
            LOG.error("Commit failed.",e);
        }
    });
```

- Submits `writeTx` and registers application provided `FutureCallback` on returned future.
- Invoked when future completed successfully - transaction `writeTx` was successfully committed to data tree.
- Invoked when future failed - commit of transaction `writeTx` failed. Supplied exception provides additional details and cause of failure.

If application need to block till commit is finished it may use `checkedGet()` to wait till commit is finished.

```
try {
    writeTx.submit().checkedGet();
} catch (TransactionCommitFailedException e) {
    LOG.error("Commit failed.",e);
}
```

- Submits `writeTx` and blocks till commit of `writeTx` is finished. If commit fails `TransactionCommitFailedException` will be thrown.
- Catches `TransactionCommitFailedException` and logs it.

Transaction local state

Read-Write transactions maintain transaction-local state, which renders all modifications as if they happened, but this is only local to transaction.

Reads from the transaction returns data as if the previous modifications in transaction already happened.

Let assume initial state of data tree for `PATH` is `A`.

```
ReadWriteTransaction rwTx = broker.newReadWriteTransaction();

rwRx.read(OPERATIONAL,PATH).get();
rwRx.put(OPERATIONAL,PATH,B);
rwRx.read(OPERATIONAL,PATH).get();
rwRx.put(OPERATIONAL,PATH,C);
rwRx.read(OPERATIONAL,PATH).get();
```

- Allocates new `ReadWriteTransaction`.
- Read from `rwTx` will return value `A` for `PATH`.
- Writes value `B` to `PATH` using `rwTx`.
- Read will return value `B` for `PATH`, since previous write occurred in same transaction.
- Writes value `C` to `PATH` using `rwTx`.
- Read will return value `C` for `PATH`, since previous write occurred in same transaction.

Transaction isolation

Running (not submitted) transactions are isolated from each other and changes done in one transaction are not observable in other currently running transaction.

Lets assume initial state of data tree for `PATH` is `A`.

```
ReadOnlyTransaction txRead = broker.newReadOnlyTransaction();
ReadWriteTransaction txWrite = broker.newReadWriteTransaction();

txRead.read(OPERATIONAL,PATH).get();
```

```
txWrite.put(OPERATIONAL,PATH,B);
txWrite.read(OPERATIONAL,PATH).get();
txWrite.submit().get();
txRead.read(OPERATIONAL,PATH).get();
txAfterCommit = broker.newReadOnlyTransaction();
txAfterCommit.read(OPERATIONAL,PATH).get();
```

- Allocates read only transaction, which is based on data tree which contains value `A` for `PATH`.
- Allocates read write transaction, which is based on data tree which contains value `A` for `PATH`.
- Read from read-only transaction returns value `A` for `PATH`.
- Data tree is updated using read-write transaction, `PATH` contains `B`. Change is not public and only local to transaction.
- Read from read-write transaction returns value `B` for `PATH`.
- Submits changes in read-write transaction to be committed to data tree. Once commit will finish, changes will be published and `PATH` will be updated for value `B`. Previously allocated transactions are not affected by this change.
- Read from previously allocated read-only transaction still returns value `A` for `PATH`, since it provides stable and isolated view.
- Allocates new read-only transaction, which is based on data tree, which contains value `B` for `PATH`.
- Read from new read-only transaction return value `B` for `PATH` since read-write transaction was committed.

Note: Examples contain blocking calls on future only to illustrate that action happened after other asynchronous action. The use of the blocking call `ListenableFuture#get()` is discouraged for most use-cases and you should use `Futures#addCallback(ListenableFuture, FutureCallback)` to listen asynchronously for result.

Commit failure scenarios

A transaction commit may fail because of following reasons:

Optimistic Lock Failure Another transaction finished earlier and **modified the same node in a non-compatible way**. The commit (and the returned future) will fail with an `OptimisticLockFailedException`.

It is the responsibility of the caller to create a new transaction and submit the same modification again in order to update data tree.

> **Warning:** `OptimisticLockFailedException` usually exposes **multiple writers** to the same data subtree, which may conflict on same resources.
>
> In most cases, retrying may result in a probability of success.
>
> There are scenarios, albeit unusual, where any number of retries will not succeed. Therefore it is strongly recommended to limit the number of retries (2 or 3) to avoid an endless loop.

Data Validation The data change introduced by this transaction **did not pass validation** by commit handlers or data was incorrectly structured. The returned future will fail with a `DataValidationFailedException`. User **should not retry** to create new transaction with same data, since it probably will fail again.

Example conflict of two transactions

This example illustrates two concurrent transactions, which derived from same initial state of data tree and proposes conflicting modifications.

```
WriteTransaction txA = broker.newWriteTransaction();
WriteTransaction txB = broker.newWriteTransaction();

txA.put(CONFIGURATION, PATH, A);
txB.put(CONFIGURATION, PATH, B);

CheckedFuture<?,?> futureA = txA.submit();
CheckedFuture<?,?> futureB = txB.submit();
```

- Updates `PATH` to value `A` using `txA`
- Updates `PATH` to value `B` using `txB`
- Seals & submits `txA`. The commit will be processed asynchronously and data tree will be updated to contain value `A` for `PATH`. The returned 'ListenableFuture' will complete successfully once state is applied to data tree.
- Seals & submits `txB`. Commit of `txB` will fail, because previous transaction also modified path in a concurrent way. The state introduced by `txB` will not be applied. The returned `ListenableFuture` will fail with `OptimisticLockFailedException` exception, which indicates that concurrent transaction prevented the submitted transaction from being applied.

Example asynchronous retry-loop

```
private void doWrite( final int tries ) {
    WriteTransaction writeTx = dataBroker.newWriteOnlyTransaction();

    MyDataObject data = ...;
    InstanceIdentifier<MyDataObject> path = ...;
    writeTx.put( LogicalDatastoreType.OPERATIONAL, path, data );

    Futures.addCallback( writeTx.submit(), new FutureCallback<Void>() {
        public void onSuccess( Void result ) {
            // succeeded
        }

        public void onFailure( Throwable t ) {
            if( t instanceof OptimisticLockFailedException && (( tries - 1 ) > 0)) {
                doWrite( tries - 1 );
            }
        }
      });
}
...
doWrite( 2 );
```

Concurrent change compatibility

There are several sets of changes which could be considered incompatible between two transactions which are derived from same initial state. Rules for conflict detection applies recursively for each subtree level.

Following table shows state changes and failures between two concurrent transactions, which are based on same initial state, `tx1` is submitted before `tx2`.

INFO: Following tables stores numeric values and shows data using `toString()` to simplify examples.

Initial state	tx1	tx2	Observable Result
Empty	`put(A,1)`	`put(A,2)`	`tx2` will fail, value of `A` is `1`
Empty	`put(A,1)`	`merge(A,2)`	value of `A` is `2`
Empty	`merge(A,1)`	`put(A,2)`	`tx2` will fail, value of `A` is `1`
Empty	`merge(A,1)`	`merge(A,2)`	`A` is `2`
A=0	`put(A,1)`	`put(A,2)`	`tx2` will fail, `A` is `1`
A=0	`put(A,1)`	`merge(A,2)`	`A` is `2`
A=0	`merge(A,1)`	`put(A,2)`	`tx2` will fail, value of `A` is `1`
A=0	`merge(A,1)`	`merge(A,2)`	`A` is `2`
A=0	`delete(A)`	`put(A,2)`	`tx2` will fail, `A` does not exists
A=0	`delete(A)`	`merge(A,2)`	`A` is `2`

Table: Concurrent change resolution for leaves and leaf-list items

Initial state	`tx1`	`tx2`	Result
Empty	put(TOP,[])	put(TOP,[])	`tx2` will fail, state is TOP=[]
Empty	put(TOP,[])	merge(TOP,[])	TOP=[]
Empty	put(TOP,[FOO=1])	put(TOP,[BAR=1])	`tx2` will fail, state is TOP=[FOO=1]
Empty	put(TOP,[FOO=1])	merge(TOP,[BAR=1])	TOP=[FOO=1,BAR=1]
Empty	merge(TOP,[FOO=1])	put(TOP,[BAR=1])	`tx2` will fail, state is TOP=[FOO=1]
Empty	merge(TOP,[FOO=1])	merge(TOP,[BAR=1])	TOP=[FOO=1,BAR=1]
TOP=[]	put(TOP,[FOO=1])	put(TOP,[BAR=1])	`tx2` will fail, state is TOP=[FOO=1]
TOP=[]	put(TOP,[FOO=1])	merge(TOP,[BAR=1])	state is TOP=[FOO=1,BAR=1]
TOP=[]	merge(TOP,[FOO=1])	put(TOP,[BAR=1])	`tx2` will fail, state is TOP=[FOO=1]
TOP=[]	merge(TOP,[FOO=1])	merge(TOP,[BAR=1])	state is TOP=[FOO=1,BAR=1]
TOP=[]	delete(TOP)	put(TOP,[BAR=1])	`tx2` will fail, state is empty store
TOP=[]	delete(TOP)	merge(TOP,[BAR=1])	state is TOP=[BAR=1]
TOP=[]	put(TOP/FOO,1)	put(TOP/BAR,1])	state is TOP=[FOO=1,BAR=1]
TOP=[]	put(TOP/FOO,1)	merge(TOP/BAR,1)	state is TOP=[FOO=1,BAR=1]
TOP=[]	merge(TOP/FOO,1)	put(TOP/BAR,1)	state is TOP=[FOO=1,BAR=1]
TOP=[]	merge(TOP/FOO,1)	merge(TOP/BAR,1)	state is TOP=[FOO=1,BAR=1]
TOP=[]	delete(TOP)	put(TOP/BAR,1)	`tx2` will fail, state is empty store
TOP=[]	delete(TOP)	merge(TOP/BAR,1]	`tx2` will fail, state is empty store
TOP=[FOO=1]	put(TOP/FOO,2)	put(TOP/BAR,1)	state is TOP=[FOO=2,BAR=1]
TOP=[FOO=1]	put(TOP/FOO,2)	merge(TOP/BAR,1)	state is TOP=[FOO=2,BAR=1]
TOP=[FOO=1]	merge(TOP/FOO,2)	put(TOP/BAR,1)	state is TOP=[FOO=2,BAR=1]
TOP=[FOO=1]	merge(TOP/FOO,2)	merge(TOP/BAR,1)	state is TOP=[FOO=2,BAR=1]
TOP=[FOO=1]	delete(TOP/FOO)	put(TOP/BAR,1)	state is TOP=[BAR=1]
TOP=[FOO=1]	delete(TOP/FOO)	merge(TOP/BAR,1]	state is TOP=[BAR=1]

Table: Concurrent change resolution for containers, lists, list items

MD-SAL RPC routing

The MD-SAL provides a way to deliver Remote Procedure Calls (RPCs) to a particular implementation based on content in the input as it is modeled in YANG. This part of the the RPC input is referred to as a **context reference**.

The MD-SAL does not dictate the name of the leaf which is used for this RPC routing, but provides necessary functionality for YANG model author to define their **context reference** in their model of RPCs.

MD-SAL routing behavior is modeled using following terminology and its application to YANG models:

Context Type Logical type of RPC routing. Context type is modeled as YANG `identity` and is referenced in model to provide scoping information.

Context Instance Conceptual location in data tree, which represents context in which RPC could be executed. Context instance usually represent logical point to which RPC execution is attached.

Context Reference Field of RPC input payload which contains Instance Identifier referencing **context instance** in which the RPC should be executed.

Modeling a routed RPC

In order to define routed RPCs, the YANG model author needs to declare (or reuse) a **context type**, set of possible **context instances** and finally RPCs which will contain **context reference** on which they will be routed.

Declaring a routing context type

This declares an identity named `node-context`, which is used as marker for node-based routing and is used in other places to reference that routing type.

Declaring possible context instances

In order to define possible values of **context instances** for routed RPCs, we need to model that set accordingly using `context-instance` extension from the `yang-ext` model.

The statement `ext:context-instance "node-context";` marks any element of the `list node` as a possible valid **context instance** in `node-context` based routing.

Note: The existence of a **context instance** node in operational or config data tree is not strongly tied to existence of RPC implementation.

For most routed RPC models, there is relationship between the data present in operational data tree and RPC implementation availability, but this is not enforced by MD-SAL. This provides some flexibility for YANG model writers to better specify their routing model and requirements for implementations. Details when RPC implementations are available should be documented in YANG model.

If user invokes RPC with a **context instance** that has no registered implementation, the RPC invocation will fail with the exception `DOMRpcImplementationNotAvailableException`.

Declaring a routed RPC

To declare RPC to be routed based on `node-context` we need to add leaf of `instance-identifier` type (or type derived from `instance-identifier`) to the RPC and mark it as **context reference**.

This is achieved using YANG extension `context-reference` from `yang-ext` model on leaf, which will be used for RPC routing.

The statement `ext:context-reference "node-context"` marks `leaf node` as **context reference** of type `node-context`. The value of this leaf, will be used by the MD-SAL to select the particular RPC implementation that registered itself as the implementation of the RPC for particular **context instance**.

Using routed RPCs

From a user perspective (e.g. invoking RPCs) there is no difference between routed and non-routed RPCs. Routing information is just an additional leaf in RPC which must be populated.

Implementing a routed RPC

Implementation

Registering implementations

Implementations of a routed RPC (e.g., southbound plugins) will specify an instance-identifier for the **context reference** (in this case a node) for which they want to provide an implementation during registration. Consumers, e.g., those calling the RPC are required to specify that instance-identifier (in this case the identifier of a node) when invoking RPC.

Simple code which showcases that for add-flow via Binding-Aware APIs (RoutedServiceTest.java):

```
61  @Override
62  public void onSessionInitiated(ProviderContext session) {
63      assertNotNull(session);
64      firstReg = session.addRoutedRpcImplementation(SalFlowService.class,␣
↪salFlowService1);
65  }
```

Line 64: We are registering salFlowService1 as implementation of SalFlowService RPC

```
107  NodeRef nodeOne = createNodeRef("foo:node:1");
109  /**
110   * Provider 1 registers path of node 1
111   */
112  firstReg.registerPath(NodeContext.class, nodeOne);
```

Line 107: We are creating NodeRef (encapsulation of InstanceIdentifier) for “foo:node:1”.

Line 112: We register salFlowService1 as implementation for nodeOne.

The salFlowService1 will be executed only for RPCs which contains Instance Identifier for foo:node:1.

RPCs and cluster

In case there is is only a single provider of an RPC in the cluster the RPC registration is propagated to other nodes via Gossip protocol and the RPC calls from other nodes are correctly routed to the provider. Since the registrations are not expected to change rapidly there is a latency of about 1 second until the registration is reflected on the remote nodes.

OpenDaylight Controller MD-SAL: RESTCONF

RESTCONF operations overview

RESTCONF allows access to datastores in the controller.
There are two datastores:

- Config: Contains data inserted via controller
- Operational: Contains other data

Note:

Each request must start with the URI /restconf.
RESTCONF listens on port 8080 for HTTP requests.

RESTCONF supports **OPTIONS**, **GET**, **PUT**, **POST**, and **DELETE** operations. Request and response data can either be in the XML or JSON format. XML structures according to yang are defined at: XML-YANG. JSON structures are defined at: JSON-YANG. Data in the request must have a correctly set **Content-Type** field in the http header with the allowed value of the media type. The media type of the requested data has to be set in the **Accept** field. Get the media types for each resource by calling the OPTIONS operation. Most of the paths of the pathsRestconf endpoints use Instance Identifier. `<identifier>` is used in the explanation of the operations.

<identifier>

- It must start with <moduleName>:<nodeName> where <moduleName> is a name of the module and <nodeName> is the name of a node in the module. It is sufficient to just use <nodeName> after <moduleName>:<nodeName>. Each <nodeName> has to be separated by /.
- <nodeName> can represent a data node which is a list or container yang built-in type. If the data node is a list, there must be defined keys of the list behind the data node name for example, <nodeName>/<valueOfKey1>/<valueOfKey2>.
- The format <moduleName>:<nodeName> has to be used in this case as well:
  Module A has node A1. Module B augments node A1 by adding node X. Module C augments node A1 by adding node X. For clarity, it has to be known which node is X (for example: C:X). For more details about encoding, see: RESTCONF 02 - Encoding YANG Instance Identifiers in the Request URI.

Mount point

A Node can be behind a mount point. In this case, the URI has to be in format <identifier>**/yang-ext:mount/**<identifier>. The first <identifier> is the path to a mount point and the second <identifier> is the path to a node behind the mount point. A URI can end in a mount point itself by using <identifier>**/yang-ext:mount**.
More information on how to actually use mountpoints is available at: OpenDaylight Controller:Config:Examples:Netconf.

HTTP methods

OPTIONS /restconf

- Returns the XML description of the resources with the required request and response media types in Web Application Description Language (WADL)

GET /restconf/config/<identifier>

- Returns a data node from the Config datastore.
- <identifier> points to a data node which must be retrieved.

GET /restconf/operational/<identifier>

- Returns the value of the data node from the Operational datastore.
- <identifier> points to a data node which must be retrieved.

PUT /restconf/config/<identifier>

- Updates or creates data in the Config datastore and returns the state about success.
- <identifier> points to a data node which must be stored.

Example:

```
PUT http://<controllerIP>:8080/restconf/config/module1:foo/bar
Content-Type: applicaton/xml
<bar>
  ...
</bar>
```

Example with mount point:

```
PUT http://<controllerIP>:8080/restconf/config/module1:foo1/foo2/yang-ext:mount/
↪module2:foo/bar
Content-Type: applicaton/xml
<bar>
  ...
</bar>
```

POST /restconf/config

- Creates the data if it does not exist

For example:

```
POST URL: http://localhost:8080/restconf/config/
content-type: application/yang.data+json
JSON payload:
```

```
  {
    "toaster:toaster" :
    {
      "toaster:toasterManufacturer" : "General Electric",
      "toaster:toasterModelNumber" : "123",
      "toaster:toasterStatus" : "up"
    }
  }
```

POST /restconf/config/<identifier>

- Creates the data if it does not exist in the Config datastore, and returns the state about success.
- <identifier> points to a data node where data must be stored.
- The root element of data must have the namespace (data are in XML) or module name (data are in JSON.)

Example:

```
POST http://<controllerIP>:8080/restconf/config/module1:foo
Content-Type: applicaton/xml/
<bar xmlns="module1namespace">
  ...
</bar>
```

Example with mount point:

```
http://<controllerIP>:8080/restconf/config/module1:foo1/foo2/yang-ext:mount/
↪module2:foo
Content-Type: applicaton/xml
<bar xmlns="module2namespace">
  ...
</bar>
```

POST /restconf/operations/<moduleName>:<rpcName>

- Invokes RPC.
- <moduleName>:<rpcName> - <moduleName> is the name of the module and <rpcName> is the name of the RPC in this module.
- The Root element of the data sent to RPC must have the name "input".
- The result can be the status code or the retrieved data having the root element "output".

Example:

```
POST http://<controllerIP>:8080/restconf/operations/module1:fooRpc
Content-Type: applicaton/xml
Accept: applicaton/xml
```

```
<input>
  ...
</input>

The answer from the server could be:
<output>
  ...
</output>
```

An example using a JSON payload:

```
POST http://localhost:8080/restconf/operations/toaster:make-toast
Content-Type: application/yang.data+json
{
  "input" :
  {
     "toaster:toasterDoneness" : "10",
     "toaster:toasterToastType":"wheat-bread"
  }
}
```

Note: Even though this is a default for the toasterToastType value in the yang, you still need to define it.

DELETE /restconf/config/<identifier>

- Removes the data node in the Config datastore and returns the state about success.
- <identifier> points to a data node which must be removed.

More information is available in the RESTCONF RFC.

How RESTCONF works

RESTCONF uses these base classes:

InstanceIdentifier Represents the path in the data tree

ConsumerSession Used for invoking RPCs

DataBrokerService Offers manipulation with transactions and reading data from the datastores

SchemaContext Holds information about yang modules

MountService Returns MountInstance based on the InstanceIdentifier pointing to a mount point

MountInstace Contains the SchemaContext behind the mount point

DataSchemaNode Provides information about the schema node

SimpleNode Possesses the same name as the schema node, and contains the value representing the data node value

CompositeNode Can contain CompositeNode-s and SimpleNode-s

GET in action

Figure 1 shows the GET operation with URI restconf/config/M:N where M is the module name, and N is the node name.

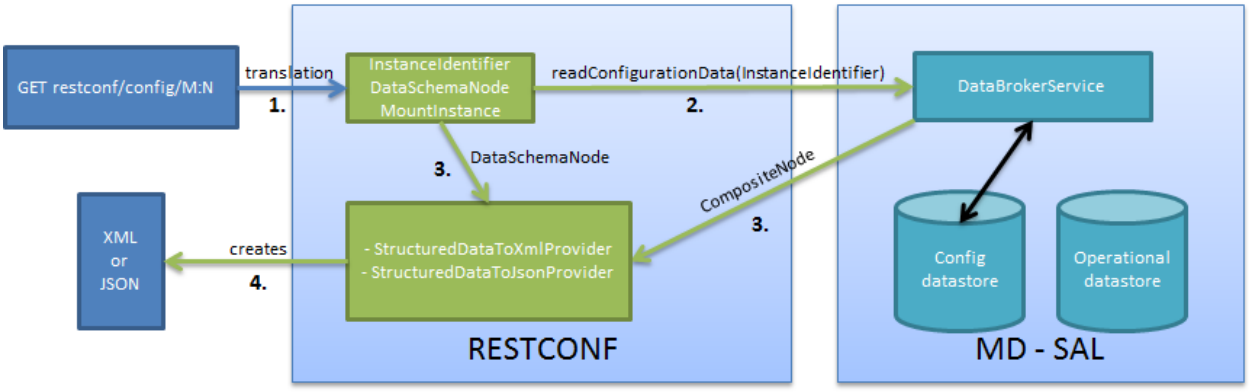


Fig. 2.23: Get

1. The requested URI is translated into the InstanceIdentifier which points to the data node. During this translation, the DataSchemaNode that conforms to the data node is obtained. If the data node is behind the mount point, the MountInstance is obtained as well.
2. RESTCONF asks for the value of the data node from DataBrokerService based on InstanceIdentifier.
3. DataBrokerService returns CompositeNode as data.
4. StructuredDataToXmlProvider or StructuredDataToJsonProvider is called based on the **Accept** field from the http request. These two providers can transform CompositeNode regarding DataSchemaNode to an XML or JSON document.
5. XML or JSON is returned as the answer on the request from the client.

PUT in action

Figure 2 shows the PUT operation with the URI restconf/config/M:N where M is the module name, and N is the node name. Data is sent in the request either in the XML or JSON format.

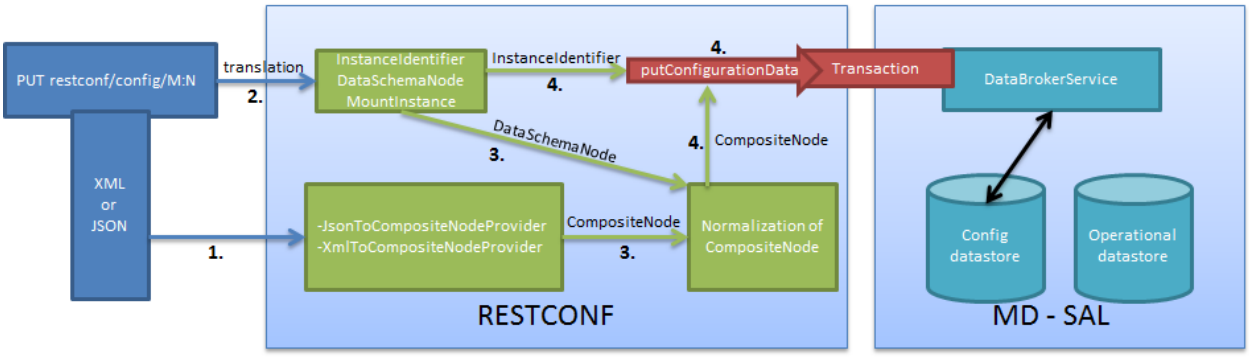


Fig. 2.24: Put

1. Input data is sent to JsonToCompositeNodeProvider or XmlToCompositeNodeProvider. The correct provider is selected based on the Content-Type field from the http request. These two providers can transform input data to CompositeNode. However, this CompositeNode does not contain enough information for transactions.
2. The requested URI is translated into InstanceIdentifier which points to the data node. DataSchemaNode conforming to the data node is obtained during this translation. If the data node is behind the mount point, the MountInstance is obtained as well.
3. CompositeNode can be normalized by adding additional information from DataSchemaNode.
4. RESTCONF begins the transaction, and puts CompositeNode with InstanceIdentifier into it. The response on the request from the client is the status code which depends on the result from the transaction.

Something practical

1. Create a new flow on the switch openflow:1 in table 2.

HTTP request

```
Operation: POST
URI: http://192.168.11.1:8080/restconf/config/opendaylight-inventory:nodes/node/
↪openflow:1/table/2
Content-Type: application/xml
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow
    xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <instructions>
        <instruction>
            <order>1</order>
            <apply-actions>
                <action>
                  <order>1</order>
                    <flood-all-action/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <table_id>2</table_id>
    <id>111</id>
    <cookie_mask>10</cookie_mask>
    <out_port>10</out_port>
    <installHw>false</installHw>
    <out_group>2</out_group>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
        <ipv4-destination>10.0.0.1/24</ipv4-destination>
    </match>
    <hard-timeout>0</hard-timeout>
    <cookie>10</cookie>
```

```
    <idle-timeout>0</idle-timeout>
    <flow-name>FooXf22</flow-name>
    <priority>2</priority>
    <barrier>false</barrier>
</flow>
```

HTTP response

```
Status: 204 No Content
```

1. Change *strict* to *true* in the previous flow.

HTTP request

```
Operation: PUT
URI: http://192.168.11.1:8080/restconf/config/opendaylight-inventory:nodes/node/
↪openflow:1/table/2/flow/111
Content-Type: application/xml
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow
    xmlns="urn:opendaylight:flow:inventory">
    <strict>true</strict>
    <instructions>
        <instruction>
            <order>1</order>
            <apply-actions>
                <action>
                  <order>1</order>
                    <flood-all-action/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <table_id>2</table_id>
    <id>111</id>
    <cookie_mask>10</cookie_mask>
    <out_port>10</out_port>
    <installHw>false</installHw>
    <out_group>2</out_group>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
        <ipv4-destination>10.0.0.1/24</ipv4-destination>
    </match>
    <hard-timeout>0</hard-timeout>
    <cookie>10</cookie>
    <idle-timeout>0</idle-timeout>
```

```
    <flow-name>FooXf22</flow-name>
    <priority>2</priority>
    <barrier>false</barrier>
</flow>
```

HTTP response

```
Status: 200 OK
```

1. Show flow: check that *strict* is *true*.

HTTP request

```
Operation: GET
URI: http://192.168.11.1:8080/restconf/config/opendaylight-inventory:nodes/node/
↪openflow:1/table/2/flow/111
Accept: application/xml
```

HTTP response

```
Status: 200 OK
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow
    xmlns="urn:opendaylight:flow:inventory">
    <strict>true</strict>
    <instructions>
        <instruction>
            <order>1</order>
            <apply-actions>
                <action>
                  <order>1</order>
                    <flood-all-action/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <table_id>2</table_id>
    <id>111</id>
    <cookie_mask>10</cookie_mask>
    <out_port>10</out_port>
    <installHw>false</installHw>
    <out_group>2</out_group>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
```

```
        </ethernet-match>
        <ipv4-destination>10.0.0.1/24</ipv4-destination>
    </match>
    <hard-timeout>0</hard-timeout>
    <cookie>10</cookie>
    <idle-timeout>0</idle-timeout>
    <flow-name>FooXf22</flow-name>
    <priority>2</priority>
    <barrier>false</barrier>
</flow>
```

1. Delete the flow created.

HTTP request

```
Operation: DELETE
URI: http://192.168.11.1:8080/restconf/config/opendaylight-inventory:nodes/node/
↪openflow:1/table/2/flow/111
```

HTTP response

```
Status: 200 OK
```

Websocket change event notification subscription tutorial

Subscribing to data change notifications makes it possible to obtain notifications about data manipulation (insert, change, delete) which are done on any specified **path** of any specified **datastore** with specific **scope**. In following examples *{odlAddress}* is address of server where ODL is running and *{odlPort}* is port on which OpenDaylight is running.

Websocket notifications subscription process

In this section we will learn what steps need to be taken in order to successfully subscribe to data change event notifications.

Create stream

In order to use event notifications you first need to call RPC that creates notification stream that you can later listen to. You need to provide three parameters to this RPC:

- **path**: data store path that you plan to listen to. You can register listener on containers, lists and leaves.
- **datastore**: data store type. *OPERATIONAL* or *CONFIGURATION*.
- **scope**: Represents scope of data change. Possible options are:
  - BASE: only changes directly to the data tree node specified in the path will be reported

  - ONE: changes to the node and to direct child nodes will be reported
  - SUBTREE: changes anywhere in the subtree starting at the node will be reported

The RPC to create the stream can be invoked via RESTCONF like this:

- URI: http://{odlAddress}:{odlPort}/restconf/operations/sal-remote:create-data-change-event-subscription
- HEADER: Content-Type=application/json
- OPERATION: POST
- DATA:

```
{
    "input": {
        "path": "/toaster:toaster/toaster:toasterStatus",
        "sal-remote-augment:datastore": "OPERATIONAL",
        "sal-remote-augment:scope": "ONE"
    }
}
```

The response should look something like this:

```
{
    "output": {
        "stream-name": "data-change-event-subscription/toaster:toaster/
↪toaster:toasterStatus/datastore=CONFIGURATION/scope=SUBTREE"
    }
}
```

stream-name is important because you will need to use it when you subscribe to the stream in the next step.

Note: Internally, this will create a new listener for *stream-name* if it did not already exist.

Subscribe to stream

In order to subscribe to stream and obtain WebSocket location you need to call *GET* on your stream path. The URI should generally be http://{odlAddress}:{odlPort}/restconf/streams/stream/{streamName}, where *{streamName}* is the *stream-name* parameter contained in response from *create-data-change-event-subscription* RPC from the previous step.

- URI: http://{odlAddress}:{odlPort}/restconf/streams/stream/data-change-event-subscription/toaster:toaster/datastore=CONFIGURATION/scope=SUBTREE
- OPERATION: GET

The subscription call may be modified with the following query parameters defined in the RESTCONF RFC:

- filter
- start-time
- end-time

In addition, the following ODL extension query parameter is supported:

odl-leaf-nodes-only If this parameter is set to "true", create and update notifications will only contain the leaf nodes modified instead of the entire subscription subtree. This can help in reducing the size of the notifications.

The expected response status is 200 OK and response body should be empty. You will get your WebSocket location from **Location** header of response. For example in our particular toaster example location header would have this value: *ws://{odlAddress}:8185/toaster:toaster/datastore=CONFIGURATION/scope=SUBTREE*

Note: During this phase there is an internal check for to see if a listener for the *stream-name* from the URI exists. If not, new a new listener is registered with the DOM data broker.

Receive notifications

You should now have a data change notification stream created and have location of a WebSocket. You can use this WebSocket to listen to data change notifications. To listen to notifications you can use a JavaScript client or if you are using chrome browser you can use the Simple WebSocket Client.

Also, for testing purposes, there is simple Java application named WebSocketClient. The application is placed in the *-sal-rest-connector-classes.class* project. It accepts a WebSocket URI as and input parameter. After starting the utility (WebSocketClient class directly in Eclipse/InteliJ Idea) received notifications should be displayed in console.

Notifications are always in XML format and look like this:

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2014-09-11T09:58:23+02:00</eventTime>
    <data-changed-notification xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:remote">
        <data-change-event>
            <path xmlns:meae="http://netconfcentral.org/ns/toaster">/meae:toaster</
↪path>
            <operation>updated</operation>
            <data>
               <!-- updated data -->
            </data>
        </data-change-event>
    </data-changed-notification>
</notification>
```

Example use case

The typical use case is listening to data change events to update web page data in real-time. In this tutorial we will be using toaster as the base.

When you call *make-toast* RPC, it sets *toasterStatus* to "down" to reflect that the toaster is busy making toast. When it finishes, *toasterStatus* is set to "up" again. We will listen to this toaster status changes in data store and will reflect it on our web page in real-time thanks to WebSocket data change notification.

Simple javascript client implementation

We will create simple JavaScript web application that will listen updates on *toasterStatus* leaf and update some element of our web page according to new toaster status state.

Create stream

First you need to create stream that you are planing to subscribe to. This can be achieved by invoking "create-data-change-event-subscription" RPC on RESTCONF via AJAX request. You need to provide data store **path** that you plan to listen on, **data store type** and **scope**. If the request is successful you can extract the **stream-name** from the response and use that to subscribe to the newly created stream. The *{username}* and *{password}* fields represent your credentials that you use to connect to OpenDaylight via RESTCONF:

Note: The default user name and password are "admin".

```
function createStream() {
    $.ajax(
        {
            url: 'http://{odlAddress}:{odlPort}/restconf/operations/sal-remote:create-
↪data-change-event-subscription',
            type: 'POST',
            headers: {
              'Authorization': 'Basic ' + btoa('{username}:{password}'),
              'Content-Type': 'application/json'
            },
            data: JSON.stringify(
                {
                    'input': {
                        'path': '/toaster:toaster/toaster:toasterStatus',
                        'sal-remote-augment:datastore': 'OPERATIONAL',
                        'sal-remote-augment:scope': 'ONE'
                    }
                }
            )
        }).done(function (data) {
            // this function will be called when ajax call is executed successfully
            subscribeToStream(data.output['stream-name']);
        }).fail(function (data) {
            // this function will be called when ajax call fails
            console.log("Create stream call unsuccessful");
        })
}
```

Subscribe to stream

The Next step is to subscribe to the stream. To subscribe to the stream you need to call *GET* on *http://{odlAddress}:{odlPort}/restconf/streams/stream/{stream-name}*. If the call is successful, you get WebSocket address for this stream in **Location** parameter inside response header. You can get response header by calling *getResponseHeader(*Location)** on HttpRequest object inside *done()* function call:

```
function subscribeToStream(streamName) {
    $.ajax(
        {
            url: 'http://{odlAddress}:{odlPort}/restconf/streams/stream/' +␣
↪streamName;
            type: 'GET',
            headers: {
              'Authorization': 'Basic ' + btoa('{username}:{password}'),
            }
```

```
        }
    ).done(function (data, textStatus, httpReq) {
        // we need function that has http request object parameter in order to access␣
↪response headers.
        listenToNotifications(httpReq.getResponseHeader('Location'));
    }).fail(function (data) {
        console.log("Subscribe to stream call unsuccessful");
    });
}
```

Receive notifications

Once you got WebSocket server location you can now connect to it and start receiving data change events. You need to define functions that will handle events on WebSocket. In order to process incoming events from OpenDaylight you need to provide a function that will handle *onmessage* events. The function must have one parameter that represents the received event object. The event data will be stored in *event.data*. The data will be in an XML format that you can then easily parse using jQuery.

```
function listenToNotifications(socketLocation) {
    try {
        var notificatinSocket = new WebSocket(socketLocation);

        notificatinSocket.onmessage = function (event) {
            // we process our received event here
            console.log('Received toaster data change event.');
            $($.parseXML(event.data)).find('data-change-event').each(
                function (index) {
                    var operation = $(this).find('operation').text();
                    if (operation == 'updated') {
                        // toaster status was updated so we call function that gets␣
↪the value of toasterStatus leaf
                        updateToasterStatus();
                        return false;
                    }
                }
            );
        }
        notificatinSocket.onerror = function (error) {
            console.log("Socket error: " + error);
        }
        notificatinSocket.onopen = function (event) {
            console.log("Socket connection opened.");
        }
        notificatinSocket.onclose = function (event) {
            console.log("Socket connection closed.");
        }
        // if there is a problem on socket creation we get exception (i.e. when␣
↪socket address is incorrect)
    } catch(e) {
        alert("Error when creating WebSocket" + e );
    }
}
```

The *updateToasterStatus()* function represents function that calls *GET* on the path that was modified and sets toaster status in some web page element according to received data. After the WebSocket connection has been established you can test events by calling make-toast RPC via RESTCONF.

Note: for more information about WebSockets in JavaScript visit Writing WebSocket client applications

Config Subsystem

Overview

The Controller configuration operation has three stages:

- First, a Proposed configuration is created. Its target is to replace the old configuration.
- Second, the Proposed configuration is validated, and then committed. If it passes validation successfully, the Proposed configuration state will be changed to Validated.
- Finally, a Validated configuration can be Committed, and the affected modules can be reconfigured.

In fact, each configuration operation is wrapped in a transaction. Once a transaction is created, it can be configured, that is to say, a user can abort the transaction during this stage. After the transaction configuration is done, it is committed to the validation stage. In this stage, the validation procedures are invoked. If one or more validations fail, the transaction can be reconfigured. Upon success, the second phase commit is invoked. If this commit is successful, the transaction enters the last stage, committed. After that, the desired modules are reconfigured. If the second phase commit fails, it means that the transaction is unhealthy - basically, a new configuration instance creation failed, and the application can be in an inconsistent state.

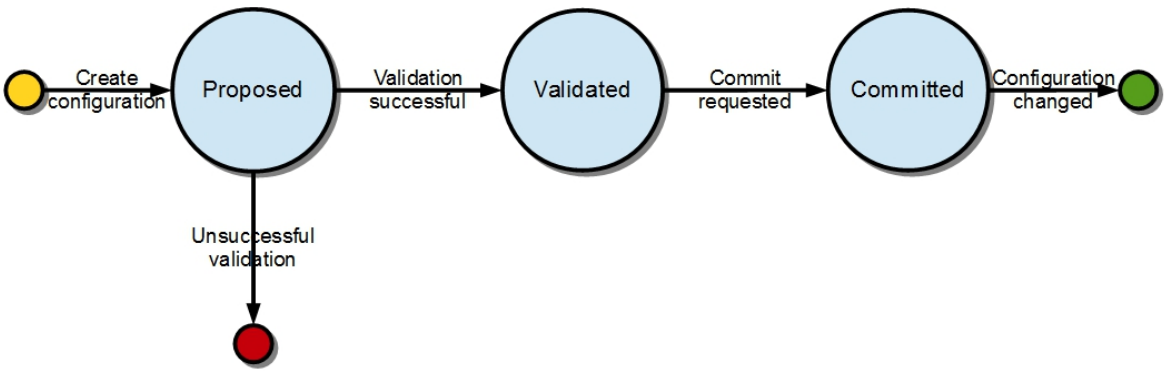


Fig. 2.25: Configuration states

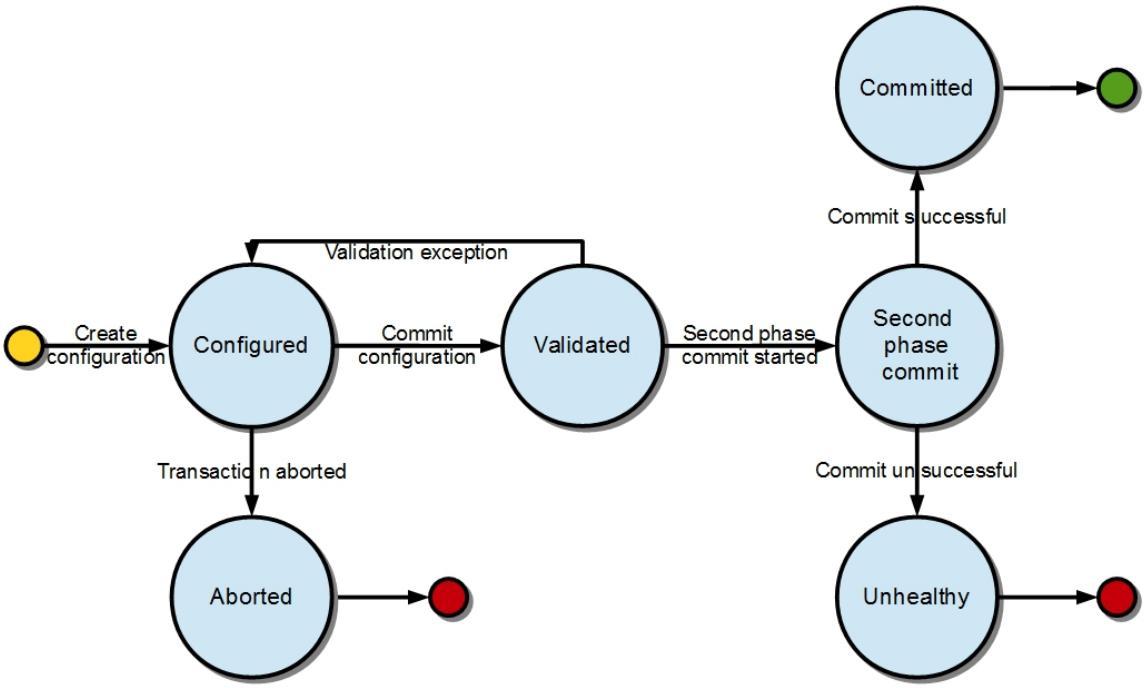


Fig. 2.26: Transaction states

Validation

To secure the consistency and safety of the new configuration and to avoid conflicts, the configuration validation process is necessary. Usually, validation checks the input parameters of a new configuration, and mostly verifies module-specific relationships. The validation procedure results in a decision on whether the proposed configuration is healthy.

Dependency resolver

Since there can be dependencies between modules, a change in a module configuration can affect the state of other modules. Therefore, we need to verify whether dependencies on other modules can be resolved. The Dependency Resolver acts in a manner similar to dependency injectors. Basically, a dependency tree is built.

APIs and SPIs

This section describes configuration system APIs and SPIs.

SPIs

Module org.opendaylight.controller.config.spi. Module is the common interface for all modules: every module must implement it. The module is designated to hold configuration attributes, validate them, and create instances of service based on the attributes. This instance must implement the AutoCloseable interface, owing to resources clean up. If the module was created from an already running instance, it contains an old instance of the module. A module can implement multiple services. If the module depends on other modules, setters need to be annotated with @RequireInterface.

Module creation

1. The module needs to be configured, set with all required attributes.
2. The module is then moved to the commit stage for validation. If the validation fails, the module attributes can be reconfigured. Otherwise, a new instance is either created, or an old instance is reconfigured. A module instance is identified by ModuleIdentifier, consisting of the factory name and instance name.

ModuleFactory org.opendaylight.controller.config.spi. The ModuleFactory interface must be implemented by each module factory.
A module factory can create a new module instance in two ways:

- From an existing module instance
- An entirely new instance
  ModuleFactory can also return default modules, useful for populating registry with already existing configurations. A module factory implementation must have a globally unique name.

APIs

ConfigRegistry	Represents functionality provided by a configuration transaction (create, destroy module, validate, or abort transaction).
ConfigTransactionController	Represents functionality for manipulating with configuration transactions (begin, commit config).
RuntimeBeanRegistratorAwareConfiBean	The module implementing this interface will receive RuntimeBeanRegistrator before getInstance is invoked.

Runtime APIs

RuntimeBean	Common interface for all runtime beans
RootRuntimeBeanRegistrator	Represents functionality for root runtime bean registration, which subsequently allows hierarchical registrations
HierarchicalRuntimeBeanRegistration	Represents functionality for runtime bean registration and unreregistration from hierarchy

JMX APIs

JMX API is purposed as a transition between the Client API and the JMX platform.

ConfigTransaction-ControllerMXBean	Extends ConfigTransactionController, executed by Jolokia clients on configuration transaction.
ConfigReg-istryMXBean	Represents entry point of configuration management for MXBeans.
Object names	Object Name is the pattern used in JMX to locate JMX beans. It consists of domain and key properties (at least one key-value pair). Domain is defined as "org.opendaylight.controller". The only mandatory property is "type".

Use case scenarios

A few samples of successful and unsuccessful transaction scenarios follow:

Successful commit scenario

1. The user creates a transaction calling creteTransaction() method on ConfigRegistry.
2. ConfigRegisty creates a transaction controller, and registers the transaction as a new bean.
3. Runtime configurations are copied to the transaction. The user can create modules and set their attributes.
4. The configuration transaction is to be committed.
5. The validation process is performed.
6. After successful validation, the second phase commit begins.
7. Modules proposed to be destroyed are destroyed, and their service instances are closed.
8. Runtime beans are set to registrator.
9. The transaction controller invokes the method getInstance on each module.
10. The transaction is committed, and resources are either closed or released.

Validation failure scenario

The transaction is the same as the previous case until the validation process.

1. If validation fails, (that is to day, illegal input attributes values or dependency resolver failure), the validationException is thrown and exposed to the user.
2. The user can decide to reconfigure the transaction and commit again, or abort the current transaction.
3. On aborted transactions, TransactionController and JMXRegistrator are properly closed.
4. Unregistration event is sent to ConfigRegistry.

Default module instances

The configuration subsystem provides a way for modules to create default instances. A default instance is an instance of a module, that is created at the module bundle start-up (module becomes visible for configuration subsystem, for example, its bundle is activated in the OSGi environment). By default, no default instances are produced.

The default instance does not differ from instances created later in the module life-cycle. The only difference is that the configuration for the default instance cannot be provided by the configuration subsystem. The module has to acquire the configuration for these instances on its own. It can be acquired from, for example, environment variables. After

the creation of a default instance, it acts as a regular instance and fully participates in the configuration subsystem (It can be reconfigured or deleted in following transactions.).

Data Export/Import Developer Guide

Overview

This feature is used to export the system data tree state, or part of, to the system's file system. It may also be used to import the system data tree state, or part of, from the system's file system.

Data Export/Import Architecture

The daexim feature consists of a single feature, which interacts with MD-SAL to export and import the system data.

Key APIs and Interfaces

The APIs are available via REST. The details are provided are in user-guide.

API Reference Documentation

The details of the API are also available in the YANG model for this feature. This model is accessiable via the APIDOC explorer interface.

DIDM Developer Guide

Overview

The Device Identification and Driver Management (DIDM) project addresses the need to provide device-specific functionality. Device-specific functionality is code that performs a feature, and the code is knowledgeable of the capability and limitations of the device. For example, configuring VLANs and adjusting FlowMods are features, and there may be different implementations for different device types. Device-specific functionality is implemented as Device Drivers. Device Drivers need to be associated with the devices they can be used with. To determine this association requires the ability to identify the device type.

DIDM Architecture

The DIDM project creates the infrastructure to support the following functions:

- **Discovery** - Determination that a device exists in the controller management domain and connectivity to the device can be established. For devices that support the OpenFlow protocol, the existing discovery mechanism in OpenDaylight suffices. Devices that do not support OpenFlow will be discovered through manual means such as the operator entering device information via GUI or REST API.
- **Identification** – Determination of the device type.
- **Driver Registration** – Registration of Device Drivers as routed RPCs.
- **Synchronization** – Collection of device information, device configuration, and link (connection) information.

- **Data Models for Common Features** – Data models will be defined to perform common features such as VLAN configuration. For example, applications can configure a VLAN by writing the VLAN data to the data store as specified by the common data model.
- **RPCs for Common Features** – Configuring VLANs and adjusting FlowMods are example of features. RPCs will be defined that specify the APIs for these features. Drivers implement features for specific devices and support the APIs defined by the RPCs. There may be different Driver implementations for different device types.

Key APIs and Interfaces

FlowObjective API

Following are the list of the APIs to create the flow objectives to install the flow rule in OpenFlow switch in pipeline agnostic way. Currently these APIs are getting consumed by Atrium project.

Install the Forwarding Objective:

`http://<CONTROLLER-IP>:8181/restconf/operations/atrium-flow-objective:forward`

Install the Filter Objective

`http://<CONTROLLER-IP>:8181/restconf/operations/atrium-flow-objective:filter`

Install the Next Objective:

`http://<CONTROLLER-IP>:8181/restconf/operations/atrium-flow-objective:next`

Flow mod driver API

This release includes a flow mod driver for the HP 3800. This driver adjusts the flows and push the same to the device. This API takes the flow to be adjusted as input and displays the adjusted flow as output in the REST output container. Here is the REST API to adjust and push flows to HP 3800 device:

```
http://<CONTROLLER-IP:8181>/restconf/operations/openflow-feature:adjust-flow
```

Here is an example of an ARP flow and how it gets adjusted and pushed to device HP3800:

adjust-flow input.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<input xmlns="urn:opendaylight:params:xml:ns:yang:didm:drivers:openflow"␣
↪xmlns:opendaylight-inventory="urn:opendaylight:inventory">
  <node>/opendaylight-inventory:nodes/opendaylight-inventory:node[opendaylight-
↪inventory:id='openflow:673249119553088']</node>
    <flow>
      <match>
        <ethernet-match>
            <ethernet-type>
                <type>2054</type>
            </ethernet-type>
        </ethernet-match>
      </match>
      <flags>SEND_FLOW_REM</flags>
      <priority>0</priority>
      <flow-name>ARP_FLOW</flow-name>
      <instructions>
```

```
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                        <output-node-connector>CONTROLLER</output-node-connector>
                        <max-length>65535</max-length>
                    </output-action>
                </action>
                <action>
                    <order>1</order>
                    <output-action>
                        <output-node-connector>NORMAL</output-node-connector>
                        <max-length>65535</max-length>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
      </instructions>
      <idle-timeout>180</idle-timeout>
      <hard-timeout>1800</hard-timeout>
      <cookie>10</cookie>
    </flow>
</input>
```

In the output, you can see that the table ID has been identified for the given flow and two flow mods are created as a result of adjustment. The first one is to catch ARP packets in Hardware table 100 with an action to goto table 200. The second flow mod is in table 200 with actions: output normal and output controller.

adjust-flow output.

```
{
  "output": {
    "flow": [
      {
        "idle-timeout": 180,
        "instructions": {
          "instruction": [
            {
              "order": 0,
              "apply-actions": {
                "action": [
                  {
                    "order": 1,
                    "output-action": {
                      "output-node-connector": "NORMAL",
                      "max-length": 65535
                    }
                  },
                  {
                    "order": 0,
                    "output-action": {
                      "output-node-connector": "CONTROLLER",
                      "max-length": 65535
                    }
                  }
                ]
```

```
                }
              }
            ]
          },
          "strict": false,
          "table_id": 200,
          "flags": "SEND_FLOW_REM",
          "cookie": 10,
          "hard-timeout": 1800,
          "match": {
            "ethernet-match": {
              "ethernet-type": {
                "type": 2054
              }
            }
          },
          "flow-name": "ARP_FLOW",
          "priority": 0
        },
        {
          "idle-timeout": 180,
          "instructions": {
            "instruction": [
              {
                "order": 0,
                "go-to-table": {
                  "table_id": 200
                }
              }
            ]
          },
          "strict": false,
          "table_id": 100,
          "flags": "SEND_FLOW_REM",
          "cookie": 10,
          "hard-timeout": 1800,
          "match": {},
          "flow-name": "ARP_FLOW",
          "priority": 0
        }
      ]
    }
}
```

API Reference Documentation

Go to http://${controller-ip}:8181/apidoc/explorer/index.html, and look under DIDM section to see all the available REST calls and tables

Distribution Version reporting

Overview

This section provides an overview of **odl-distribution-version** feature.

A remote user of OpenDaylight usually has access to RESTCONF and NETCONF northbound interfaces, but does not have access to the system OpenDaylight is running on. OpenDaylight has released multiple versions including Service Releases, and there are incompatible changes between them. In order to know which YANG modules to use, which bugs to expect and which workarounds to apply, such user would need to know the exact version of at least one OpenDaylight component.

There are indirect ways to deduce such version, but the direct way is enabled by odl-distribution-version feature. Administrator can specify version strings, which would be available to users via NETCONF, or via RESTCONF if OpenDaylight is configured to initiate NETCONF connection to its config subsystem northbound interface.

By default, users have write access to config subsystem, so they can add, modify or delete any version strings present there. Admins can only influence whether the feature is installed, and initial values.

Config subsystem is local only, not cluster aware, so each member reports versions independently. This is suitable for heterogeneous clusters. On homogeneous clusters, make sure you set and check every member.

Key APIs and Interfaces

Current implementation relies heavily on `config-parent` parent POM file from Controller project.

YANG model for config subsystem

Throughout this chapter, *model* denotes YANG module, and *module* denotes item in config subsystem module list.

Version functionality relies on config subsystem and its `config` YANG model. The YANG model `odl-distribution-version` adds an identity `odl-version` and augments `/config:modules/module/configuration` adding new case for `odl-version` type. This case contains single leaf `version`, which would hold the version string.

Config subsystem can hold multiple modules, the version string should contain version of OpenDaylight component corresponding to the module name. As this is pure metadata with no consequence on OpenDaylight behavior, there is no prescribed scheme for chosing config module names. But see the default configuration file for examples.

Java API

Each config module needs to come with java classes which override `customValidation()` and `createInstance()`. Version related modules have no impact on OpenDaylight internal behavior, so the methods return void and dummy closeable respectively, without any side effect.

Default config file

Initial version values are set via config file `odl-version.xml` which is created in `$KARAF_HOME/etc/opendaylight/karaf/` upon installation of `odl-distribution-version` feature. If admin wants to use different content, the file with desired content has to be created there before feature installation happens.

By default, the config file defines two config modules, named `odl-distribution-version` and `odl-odlparent-version`.

Currently the default version values are set to Maven property strings (as opposed to valid values), as the needed new functionality did not make it into Controller project in Boron. See Bug number 6003.

Karaf Feature

The `odl-distribution-version` feature is currently the only feature defined in feature repository of artifactId `features-distribution`, which is available (transitively) in OpenDaylight Karaf distribution.

RESTCONF usage

Opendaylight config subsystem NETCONF northbound is not made available just by installing `odl-distribution-version`, but most other feature installations would enable it. RESTCONF interfaces are enabled by installing `odl-restconf` feature, but that do not allow access to config subsystem by itself.

On single node deployments, installation of `odl-netconf-connector-ssh` is recommended, which would configure `controller-config` device and its MD-SAL mount point. See documentation for clustering on how to create similar devices for member modes, as `controller-config` name is not unique in that context.

Assuming single node deployment and user located on the same system, here is an example `curl` command accessing `odl-odlparent-version` config module:

```
curl 127.0.0.1:8181/restconf/config/network-topology:network-topology/topology/
↪topology-netconf/node/controller-config/yang-ext:mount/config:modules/module/odl-
↪distribution-version:odl-version/odl-odlparent-version
```

Distribution features

Overview

This section provides an overview of **odl-integration-compatible-with-all** and **odl-integration-all** features.

Integration/Distribution project produces a Karaf 4 distribution which gives users access to many Karaf features provided by upstream OpenDaylight projects. Users are free to install arbitrary subset of those features, but not every feature combination is expected to work properly.

Some features are pro-active, which means OpenDaylight in contact with othe network elements starts diving changes in the network even without prompting by users, in order to satisfy initial conditions their use case expects. Such activity from one feature may in turn affect behavior of another feature.

In some cases, there exists features which offer diferent implementation of the same service, they may fail to initialize properly (e.g. failing to bind a port already bound by the other feature).

Integration/Test project is maintaining system tests (CSIT) jobs. Aside of testing scenarios with only a minimal set of features installed (-only- jobs), the scenarios are also tested with a large set of features installed (-all- jobs).

In order to define a proper set of features to test with, Integration/Distribution project defines two “aggregate” features. Note that these features are not intended for production use, so the feature repository which defines them is not enabled by default.

The content of these features is determined by upstream OpenDaylight contributions, with Integration/Test providing insight on observed compatibuility relations. Integration/Distribution team is focused only on making sure the build process is reliable.

Feature repositories

features-index

This feature repository is enabled by default. It does not refer to any new features directly, instead it refers to upstream feature repositories, enabling any feature contained therein to be available for installation.

features-test

This feature repository defines the two aggregate features. To enable this repository, change the featuresRepositories line of org.apache.karaf.features.cfg file, by copy-pasting the feature-index value and editing the name.

Karaf features

The two aggregate features, defining sets of user-facing features defined by compatibility requirements. Note that is the compatibility relation differs between single node an cluster deployments, single node point of view takes precedence.

odl-integration-all

This feature contains the largest set of user-facing features which may affect each others operation, but the set does not affect usability of Karaf infrastructure.

Note that port binding conflicts and "server is unhealthy" status of config subsystem are considered to affect usability, as is a failure of Restconf to respond to GET on /restconf/modules with HTTP status 200.

This feature is used in verification process for Integration/Distribution contributions.

odl-integration-compatible-with-all

This feature contains the largest set of user-facing features which are not pro-active and do not affect each others operation.

Installing this set together with just one of odl-integration-all features should still result in fully operational installation, as one pro-active feature should not lead to any conflicts. This should also hold if the single added feature is outside odl-integration-all, if it is one of conflicting implementations (and no such implementations is in odl-integration-all).

This feature is used in the aforementioned -all- CSIT jobs.

DLUX

Setup and Run

Required Technology Stack

- AngularJS (JavaScript client-side framework, http://www.angularjs.org)

Run DLUX

To turn on the DLUX UI, install DLUX core feature via running following command on the Karaf console -

```
feature:install odl-dlux-core
```

The above command will install odl-restconf along with core DLUX components. Once this feature is successfully installed, access the UI at http://localhost:8181/index.html. The default credentials for login are admin/admin. After successful login you'll see empty page. For applications, continue with DluxApps project.

DLUX Modules

DLUX modules are the individual features such as nodes and topology. Each module has a defined structure and you can find all existing modules at https://github.com/opendaylight/dlux/tree/stable/boron/modules.

Module Structure

- module_folder
  - <module_name>.module.js
  - <module_name>.controller.js
  - <module_name>.services.js
  - <module_name>.directives.js
  - <module_name>.filter.js
  - index.tpl.html
  - <a_stylesheet>.css

Create New Module

Define the module

1. Create an empty maven project and create your module folder under src/main/resources.
2. Create an empty file with pattern <module_name>.module.js.
3. Next, you need to surround the angular module with a define function. This allows RequireJs to see our module.js files. The first argument is an array which contains all the module's dependencies. The second argument is a callback function, whose body contain the AngularJS code base. The function parameters correspond with the order of dependencies. Each dependency is injected into a parameter, if it is provided.
4. Finally, you will return the angular module to be able to inject it as a parameter in others modules.

For each new module, you must have at least these two dependencies :

- angularAMD : It's a wrapper around AngularJS to provide an AMD (Asynchronous Module Definition) support, which is used by RequireJs. For more information see the AMD documentation.
- app/core/core.services : This one is mandatory, if you want to add content in the navigation menu, the left bar or the top bar.

The following are not mandatory, but very often used.

- angular-ui-router : A library to provide URL routing.
- routingConfig : To set the level access to a page.

Your module.js file might look like this:

```
define(['angularAMD','app/routingConfig', 'angular-ui-router','app/core/core.services
↪'], function(ng) {
   var module = angular.module('app.a_module', ['ui.router.state', 'app.core']);
   // module configuration
   module.config(function() {
       [...]
   });
  return module;
});
```

Set the register function

AngularJS allows lazy registration of a module's components such as controller, factory etc. Once you will install your application, DLUX will load your module javascript, but not your angular component during bootstrap phase. You have to register your angular components to make sure they are available at the runtime.

Here is how to register your module's component for lazy initialization -

```
module.config(function($compileProvider, $controllerProvider, $provide) {
   module.register = {
     controller : $controllerProvider.register,
     directive : $compileProvider.directive,
     factory : $provide.factory,
     service : $provide.service
   };
});
```

Set the route

The next step is to set up the route for your module. This part is also done in the configuration method of the module. We have to add **$stateProvider** as a parameter.

```
module.config(function($stateProvider) {
   var access = routingConfig.accessLevels;
   $stateProvider.state('main.module', {
     url: 'module',
     views : {
       'content' : {
         templateUrl: 'src/app/module/module.tpl.html',
         controller: 'ModuleCtrl'
       }
     }
   });
});
```

Adding element to the navigation menu

To be able to add item to the navigation menu, the module requires the **NavHelperProvider** parameter in the configuration method. **addToMenu** method in **NavMenuHelper** helper allows an item addition to the menu.

```
var module = angular.module('app.a_module', ['app.core']);
module.config(function(NavMenuHelper) {
    NavMenuHelper.addToMenu('myFirstModule', {
        "link" : "#/module/index",
        "active" : "module",
        "title" : "My First Module",
        "icon" : "icon-sitemap",
        "page" : {
            "title" : "My First Module",
            "description" : "My first module"
        }
    });
});
```

The first parameter is an ID that refers to the level of your menu and the second is a object. For now, The ID parameter supports two levels of depth. If your ID looks like *rootNode.childNode*, the helper will look for a node named *rootNode* and it will append the *childNode* to it. If the root node doesn't exist, it will create it.

Link the AngularJS module's controller file

To include the module's controller file, you can use the NavHelperProvider. It contains a method that will load the given file.

```
[...]
   NavHelperProvider.addControllerUrl('<path_to_module_folder>/<module_name>.
↪controller');
```

This completes your module.js file.

Create the controller, factory, directive, etc

Creating the controller and other components is similar to the module.

- First, add the define method.
- Second, add the relative path to the module definition.
- Last, create your methods as you usually do it with AngularJS.

For example -

```
define(['<relative_path_to_module>/<module_name>.module'], function(module) {
   module.register.controller('ModuleCtrl', function($rootScope, $scope) {
   });
});
```

Add new application using DLUX modularity

DLUX works as a Karaf based UI platform, where you can create a new Karaf feature of your UI component and install that UI applications in DLUX using blueprint. This page will help you to create and load a new application for

DLUX. You don't have to add new module in DLUX repository.

Add a new OSGi blueprint bundle

The OSGi Blueprint Container specification allows us to use dependency injection in our OSGi environment. Each DLUX application module registers itself via blueprint configuration. Each application will have its own blueprint.xml to place its configuration.

1. Create a maven project to place blueprint configuration. For reference, take a look at topology bundle, present at https://github.com/opendaylight/dlux/tree/stable/boron/bundles/topology. All the existing DLUX modules' configurations are available under bundles directory of DLUX code.
2. In pom.xml, you have to add a maven plugin to unpack your module code under generated-resources of this project. For reference, you can check pom.xml of dlux/bundles/topology at https://github.com/opendaylight/dlux/tree/stable/boron/bundles/topology. Your bundle will eventually get deployed in Karaf as feature, so your bundle should contain all your module code. If you want to combine module and bundle project, that should not be an issue either.
3. Create a blueprint.xml configuration file under src/main/resources/OSGI-INF/blueprint. Below is the content of the blueprint.xml taken from topology bundles's blueprint.xml. Any new application should create a blueprint.xml in following format -

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
    <reference id="httpService" availability="mandatory" activation="eager" interface=
↪"org.osgi.service.http.HttpService"/>
    <reference id="loader" availability="mandatory" activation="eager" interface="org.
↪opendaylight.dlux.loader.DluxModuleLoader"/>

    <bean id="bundle" init-method="initialize" destroy-method="clean" class="org.
↪opendaylight.dlux.loader.DluxModule">
      <property name="httpService" ref="httpService"/>
      <property name="loader" ref="loader"/>
      <property name="moduleName" value="topology "/>
      <property name="url" value="/src/app/topology"/>
      <property name="directory" value="/topology"/>
      <property name="requireJs" value="app/topology/topology.module"/>
      <property name="angularJs" value="app.topology"/>
      <property name="cssDependencies">
          <list>
              <value>http://yui.yahooapis.com/3.18.1/build/cssreset/cssreset-min.css</
↪value>
              <value>src/app/topology/topology-custom.css</value>
          </list>
      </property>
    </bean>
</blueprint>
```

In above configuration, there are two references with id httpService and loader. These two beans will already be initialized by dlux-core, so any new application can use them. Without these two bean references, a new application will not be able to register.

Next is the initialization of your application bean, which will be an instance of class org.opendaylight.dlux.loader.DluxModule. There are 5 properties that you should provide in this bean besides the references of httpService and loader. Lets talk about those bean properties in little more detail.

moduleName : Name of your module. This name should be unique in DLUX.

url: This is the url via which RequireJS in DLUX will try to load your module JS/HTML files. Also, this is the url

that browser will use to load the static HTML, JS or CSS files. RequireJS in DLUX has a base path of **src**, so all the url should start with /src so RequireJS and the browser can correctly find the files.

directory: In your bundle's pom.xml, you unpack your module code. This is the directory where your actual static files will reside. The above mentioned url is registered with httpService, so when browser makes a call to that url, it will be redirected to the directory mentioned here. In the above example, all the topology files are present under /topology directory and the browser/RequireJS can access those files with uri /src/app/topology.

requireJS: This is the path to your RequireJS module. If you notice closely, you will see the initial path of RequireJS app/topology in the above example matches with the last part of url. This path will be be used by RequireJS. As mentioned above, we have kept **src** as base path in RequireJS, that is the exact reason that url start with /src.

angularJS: name of your AngularJS module.

cssDependencies: If the application has any external/internal css dependencies, then those can be added here. If you create your own css files, just point to those css files here. Use the url path that you mentioned above, so the browser can find your css file.

OSGi understands blueprint.xml, once you will deploy your bundle in karaf (or you can create a new feature for your application), karaf will read your blueprint.xml and it will try to register your application with dlux. Once successful, if you refresh your dlux UI, you will see your application in left hand navigation bar of dlux.

Yang Utils

Yang Utils are used by UI to perform all CRUD operations. All of these utilities are present in yangutils.services.js file. It has following AngularJS factories -

- **arrayUtils** – defines functions for working with arrays.
- **pathUtils** – defines functions for working with xpath (paths to APIs and subAPIs). It divides xpath string to array of elements, so this array can be later used for search functions.
- **syncFact** – provides synchronization between requests to and from OpenDaylight when it's needed.
- **custFunct** – it is linked with apiConnector.createCustomFunctionalityApis in yangui controller in yangui.controller.js. That function makes it possible to create some custom function called by the click on button in index.tpl.html. All custom functions are stored in array and linked to specific subAPI. When particular subAPI is expanded and clicked, its inputs (linked root node with its child nodes) are displayed in the bottom part of the page and its buttons with custom functionality are displayed also.
- **reqBuilder** – Builds object in JSON format from input fields of the UI page. **Show Preview** button on Yang UI use this builder. This request is sent to OpenDaylight when button PUT or POST is clicked.
- **yinParser** – factory for reading .xml files of yang models and creating object hierarchy. Every statement from yang is represented by a node.
- **nodeWrapper** – adds functions to objects in tree hierarchy created with yinParser. These functions provide functionality for every type of node.
- **apiConnector** – the main functionality is filling the main structures and linking them. Structure of APIs and subAPIs which is two level array - first level is filled by main APIs, second level is filled by others sub APIs. Second main structure is array of root nodes, which are objects including root node and its children nodes. Linking these two structures is creating links between every subAPI (second level of APIs array) and its root node, which must be displayed like inputs when subAPI is expanded.
- **yangUtils** – some top level functions which are used by yangui controller for creating the main structures.

eman Developer Guide

Overview

The OpenDaylight Energy Management (eman) plugin implements an abstract Information Model that describes energy measurement and control features that may be supported by a variety of device types. The eman plugin may support a number of southbound interfaces to accommodate a set of protocols, including but not limited to SNMP, NETCONF, IPDR. The plugin presents a northbound REST API. This framework enables any number of applications to interoperate with any number of devices in order to measure and optimize energy usage. The Information Model will be inherited from the SCTE 216 standard – Adaptive Power Systems Interface Specification (APSIS), which in turn inherits definitions within the IETF eman document set.

This documentation is directed to developers who may use the eman features to build other OpenDaylight features or applications.

eman is composed of 3 Karaf features:

- `eman` incudes the YANG model and its implementation
- `eman-api` adds support for REST
- `eman-ui` adds support for DLUX.

Developers will typically interface with `eman-api`.

eman Architecture

`eman` defines a YANG model that represents the IETF energy management Information Model, and includes RPCs. The implementation of the model currently supports an SNMP 'binding' via interfacing with the OpenDaylight SNMP module. In the future, other Southbound protocols may be supported.

Developers my use the `eman-api` feature to read and write energy related data and commands to devices that support the IETF eman MIBS.

Key APIs and Interfaces

The eman API currently supports a subset of the IETF eman Information Model, including the EnergyObjectPowerMeasurement table. Users of the API may get individual attributes or the entire table. When querying the table, the results are written into the MD-SAL, for subsequent access. For example, a developer may periodically poll a device for its powerMeasurements, and fetch a collection of measurements to discover a history of measurements.

Operational API

Via MD-SAL, the following endpoint provides access to previously captured power measurements.

Note: "eo" indicates "energy object" as per the IETF Information Model

operational:

```
eman:eoDevices/eoDevice{id}/eoPowerMeasurement{id}

  id indicates an index into a collection
```

EoDevices may contain a collection of individual eoDevice objects, which in turn may contain a collection of eoPowerMeasurement objects

Operations API

A set of RPCs enable interactions with devices.

get-eoAttribute enables query on an individual attribute of a energy object:

```
get-eoAttribute

  deviceIP indicates IP address of target device
  attribute indicates name of requested attribute
```

Note: Future releases will provide a enumeration of allowed names.

The supported name are:

- eoPower
- eoPowerNameplate
- eoPowerUnitMultiplier
- eoPowerAccuracy
- eoPowerMeasurementCaliber
- eoPowerCurrentType
- eoPowerMeasurementLocal
- eoPowerAdminState
- eoPowerOperState
- eoPowerStateEnterReason

set-eoAttribute enables sending a command to an energy object:

```
set-eoAttribute

  deviceIP. IP address of target device
  attribute. string indicating name of attribute. Currently, no attributes
```

get-eoDevicePowerMeasures reads an eoPowerMEasurements table from a device and stores the result in MD-SAL, making it available vie the operational API:

```
get-eoDevicePowerMeasures

  deviceIP. IP address of target device
```

API Reference Documentation

See eman project page for additional information: https://wiki.opendaylight.org/view/eman:Main

Fabric As A Service

FaaS (Fabric As A service) has two layers of APIs. We describe the top level API in the user guide. This document focuses on the Fabric level API and describes each API's semantics and example implementation. The second layer defines an abstraction layer called *''Fabric''* API. The idea is to abstract network into a topology formed by a collections of fabric objects other than varies of physical devices.Each Fabric object provides a collection of unified services.The top level API enables application developers or users to write applications to map high level model such as GBP, Intent etc... into a logical network model, while the lower level gives the application more control to individual fabric object level. More importantly the Fabric API is more like SP (Service Provider API) a fabric provider or vendor can implement the SPI based on its own Fabric technique such as TRILL, SPB etc ...

For how to use first level API operation, please refer to user guide for more details.

FaaS Architecture

FaaS Architecture is an 3 layered architecture, on the top is the FaaS Application layer, in the middle is the Fabric manager and at the bottom are different types of fabric objects. From bottom up, it is

Fabric and its controller (Fabric Controller) The Fabric object provides an abstraction of a homogeneous network or portion of the network and also has a built in Fabric controller which provides management plane and control plane for the fabric. The fabric controller implements the services required in Fabric Service and monitor and control the fabric operation.

Fabric Manager Fabric Manager manages all the fabric objects. also Fabric manager acts as a Unified Fabric Controller which provides inter-connect fabric control and configuration Also Fabric Manager is FaaS API service via Which FaaS user level logical network API (the top level API as mentioned previously) exposed and implemented.

FaaS renderer for GBP (Group Based Policy) FaaS renderer for GBP is an application of FaaS and provides the rendering service between GBP model and logical network model provided by Fabric Manager.

Fabric APIs and Interfaces

FaaS APIs have 4 groups as defined below

Fabric Provisioning API This set of APIs is used to create and remove Fabric Abstractions, in other words, those APIs is to provision the underlay networks and prepare to create overlay network(the logical network) on top of it.

Fabric Service API This set of APIs is used to create logical network over the Fabrics.

EndPoint API EndPoint API is used to bind a physical port which is the location of the attachment of an EndPoint happens or will happen.

OAM API Those APIs are for Operations, Administration and Maintenance purpose and In current release, OAM API is not implemented yet.

Fabric Provisioning API

- http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric:compose-fabric
- http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric:decompose-fabric

- http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric:get-all-fabrics

Fabric Service API

- RESTCONF for creating Logical port, switch, router, routing entries and link. Among them, both switches and routers have ports. links connect ports.these 5 logical elements are basic building blocks of a logical network.
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-service:create-logical-switch
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-service:rm-logical-switch
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-service:create-logical-router
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-service:rm-logical-router
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-service:add-static-route
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-service:create-logic-port
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-service:rm-logic-port
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-service:create-gateway
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-service:rm-gateway
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-service:port-binding-logical-to-fabric
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-service:port-binding-logical-to-device
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-service:add-port-function
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-service:add-acl
  - http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-service:del-acl

EndPoint API

The following APIs is to bind the physical ports to the logical ports on the logical switches:

- http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-endpoint:register-endpoint
- http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-endpoint:unregister-endpoint

- http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-endpoint:locate-endpoint

Others API

- http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/fabric-resource:create-fabric-port

API Reference Documentation

Go to http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/apidoc/index.html and expand on ‘’*FaaS*‘’ related panel for more APIs.

Infrautils

Overview

Infrautils offer various utilities and infrastructures for other projects to use:

Counters Infrastructure

Create, update and output counters is a basic tool for debugging and generating statistics in any system. We have developed a counter infrastructure integrated into ODL which has already been successfully used with multiple products, and more recently in debugging and fixing the OpenFlow plugin/Java and LACP modules. Getting started with Counters

Async Infrastructure

The decision to split a service into one or more threads with asynchronous interactions between them is frequently dependent on constraints learned late in the development and even the deployment cycle. In order to allow flexibility in making these decisions we have developed an infrastructure which is configuration driven allowing agnostic code to be written under generic constrains which can then later be customized according to the required constraints. Getting started with Async

IoTDM Developer Guide

Overview

The Internet of Things Data Management (IoTDM) on OpenDaylight project is about developing a data-centric middleware that will act as a oneM2M compliant IoT Data Broker and enable authorized applications to retrieve IoT data uploaded by any device. The OpenDaylight platform is used to implement the oneM2M data store which models a hierarchical containment tree, where each node in the tree represents an oneM2M resource. Typically, IoT devices and applications interact with the resource tree over standard protocols such as CoAP, MQTT, and HTTP. Initially, the oneM2M resource tree is used by applications to retrieve data. Possible applications are inventory or device management systems or big data analytic systems designed to make sense of the collected data. But, at some point, applications will need to configure the devices. Features and tools will have to be provided to enable configuration of the devices based on applications responding to user input, network conditions, or some set of programmable rules or policies possibly triggered by the receipt of data collected from the devices. The OpenDaylight platform, with its

rich unique cross-section of SDN capabilities, NFV, and now IoT device and application management, can be bundled with a targeted set of features and deployed anywhere in the network to give the network service provider ultimate control. Depending on the use case, the OpenDaylight IoT platform can be configured with only IoT data collection capabilities where it is deployed near the IoT devices and its footprint needs to be small, or it can be configured to run as a highly scaled up and out distributed cluster with IoT, SDN and NFV functions enabled and deployed in a high traffic data center.

oneM2M Architecture

The architecture provides a framework that enables the support of the oneM2M resource containment tree. The onem2m-core implements the MDSAL RPCs defined in the onem2m-api YANG files. These RPCs enable oneM2M resources to be created, read, updated, and deleted (CRUD), and also enables the management of subscriptions. When resources are CRUDed, the onem2m-notifier issues oneM2M notification events to interested subscribers. TS0001: oneM2M Functional Architecture and TS0004: oneM2M Service Layer Protocol are great reference documents to learn details of oneM2M resource types, message flow, formats, and CRUD/N semantics. Both of these specifications can be found at http://onem2m.org/technical/published-documents

The oneM2M resource tree is modeled in YANG and essentially is a meta-model for the tree. The oneM2M wire protocols allow the resource tree to be constructed via HTTP or CoAP messages that populate nodes in the tree with resource specific attributes. Each oneM2M resource type has semantic behaviour associated with it. For example: a container resource has attributes which control quotas on how many and how big the collection of data or content instance objects that can exist below it in the tree. Depending on the resource type, the oneM2M core software implements and enforces the resource type specific rules to ensure a well-behaved resource tree.

The resource tree can be simultaneously accessed by many concurrent applications wishing to manage or access the tree, and also many devices can be reporting in new data or sensor readings into their appropriate place in the tree.

Key APIs and Interfaces

The API's to access the oneM2M datastore are well documented in TS0004 (referred above) found on onem2m.org

RESTCONF is available too but generally HTTP and CoAP are used to access the oneM2M data tree.

L2Switch Developer Guide

Overview

The L2Switch project provides Layer2 switch functionality.

L2Switch Architecture

- Packet Handler
  - Decodes the packets coming to the controller and dispatches them appropriately
- Loop Remover
  - Removes loops in the network
- Arp Handler
  - Handles the decoded ARP packets
- Address Tracker

  - Learns the Addresses (MAC and IP) of entities in the network
- Host Tracker
  - Tracks the locations of hosts in the network
- L2Switch Main
  - Installs flows on each switch based on network traffic

Key APIs and Interfaces

- Packet Handler
- Loop Remover
- Arp Handler
- Address Tracker
- Host Tracker
- L2Switch Main

Packet Dispatcher

Classes

- AbstractPacketDecoder
  - Defines the methods that all decoders must implement
- EthernetDecoder
  - The base decoder which decodes the packet into an Ethernet packet
- ArpDecoder, Ipv4Decoder, Ipv6Decoder
  - Decodes Ethernet packets into the either an ARP or IPv4 or IPv6 packet

Further development

There is a need for more decoders. A developer can write

- A decoder for another EtherType, i.e. LLDP.
- A higher layer decoder for the body of the IPv4 packet or IPv6 packet, i.e. TCP and UDP.

How to write a new decoder

- extends AbstractDecoder<A, B>
  - A refers to the notification that the new decoder consumes
  - B refers to the notification that the new decoder produces
- implements xPacketListener
  - The new decoder must specify which notification it is listening to
- canDecode method

  - This method should examine the consumed notification to see whether the new decoder can decode the contents of the packet
- decode method
  - This method does the actual decoding of the packet

Loop Remover

Classes

- **LoopRemoverModule**
  - Reads config subsystem value for *is-install-lldp-flow*
    - If *is-install-lldp-flow* is true, then an **InitialFlowWriter** is created
  - Creates and initializes the other LoopRemover classes
- **InitialFlowWriter**
  - Only created when *is-install-lldp-flow* is true
  - Installs a flow, which forwards all LLDP packets to the controller, on each switch
- **TopologyLinkDataChangeHandler**
  - Listens to data change events on the Topology tree
  - When these changes occur, it waits *graph-refresh-delay* seconds and then tells **NetworkGraphImpl** to update
  - Writes an STP (Spanning Tree Protocol) status of "forwarding" or "discarding" to each link in the Topology data tree
    - Forwarding links can forward packets.
    - Discarding links cannot forward packets.
- **NetworkGraphImpl**
  - Creates a loop-free graph of the network

Configuration

- graph-refresh-delay
  - Used in TopologyLinkDataChangeHandler
  - A higher value has the advantage of doing less graph updates, at the potential cost of losing some packets because the graph didn't update immediately.
  - A lower value has the advantage of handling network topology changes quicker, at the cost of doing more computation.
- is-install-lldp-flow
  - Used in LoopRemoverModule
  - "true" means a flow that sends all LLDP packets to the controller will be installed on each switch
  - "false" means this flow will not be installed
- lldp-flow-table-id

  - The LLDP flow will be installed on the specified flow table of each switch
- lldp-flow-priority
  - The LLDP flow will be installed with the specified priority
- lldp-flow-idle-timeout
  - The LLDP flow will timeout (removed from the switch) if the flow doesn't forward a packet for *x* seconds
- lldp-flow-hard-timeout
  - The LLDP flow will timeout (removed from the switch) after *x* seconds, regardless of how many packets it is forwarding

Further development

No suggestions at the moment.

Validating changes to Loop Remover

STP Status information is added to the Inventory data tree.

- A status of "forwarding" means the link is active and packets are flowing on it.
- A status of "discarding" means the link is inactive and packets are not sent over it.

The STP status of a link can be checked through a browser or a REST Client.

```
http://10.194.126.91:8080/restconf/operational/opendaylight-inventory:nodes/node/
↪openflow:1/node-connector/openflow:1:2
```

The STP status should still be there after changes are made.

Arp Handler

Classes

- **ArpHandlerModule**
  - Reads config subsystem value for *is-proactive-flood-mode*
    - If *is-proactive-flood-mode* is true, then a ProactiveFloodFlowWriter is created
    - If *is-proactive-flood-mode* is false, then an InitialFlowWriter is created
- **ProactiveFloodFlowWriter**
  - Only created when *is-proactive-flood-mode* is true
  - Installs a flood flow on each switch. With this flood flow, a packet that doesn't match any other flows will be flooded/broadcast from that switch.
- **InitialFlowWriter**
  - Only created when *is-proactive-flood-mode* is false
  - Installs a flow, which sends all ARP packets to the controller, on each switch
- **ArpPacketHandler**

  - Only created when *is-proactive-flood-mode* is false
  - Handles and processes the controller's incoming ARP packets
  - Uses **PacketDispatcher** to send the ARP packet back into the network
- **PacketDispatcher**
  - Only created when *is-proactive-flood-mode* is false
  - Sends packets out to the network
  - Uses **InventoryReader** to determine which node-connector to a send a packet on
- **InventoryReader**
  - Only created when *is-proactive-flood-mode* is false
  - Maintains a list of each switch's node-connectors

Configuration

- is-proactive-flood-mode
  - "true" means that flood flows will be installed on each switch. With this flood flow, each switch will flood a packet that doesn't match any other flows.
    * Advantage: Fewer packets are sent to the controller because those packets are flooded to the network.
    * Disadvantage: A lot of network traffic is generated.
  - "false" means the previously mentioned flood flows will not be installed. Instead an ARP flow will be installed on each switch that sends all ARP packets to the controller.
    * Advantage: Less network traffic is generated.
    * Disadvantage: The controller handles more packets (ARP requests & replies) and the ARP process takes longer than if there were flood flows.
- flood-flow-table-id
  - The flood flow will be installed on the specified flow table of each switch
- flood-flow-priority
  - The flood flow will be installed with the specified priority
- flood-flow-idle-timeout
  - The flood flow will timeout (removed from the switch) if the flow doesn't forward a packet for *x* seconds
- flood-flow-hard-timeout
  - The flood flow will timeout (removed from the switch) after *x* seconds, regardless of how many packets it is forwarding
- arp-flow-table-id
  - The ARP flow will be installed on the specified flow table of each switch
- arp-flow-priority
  - The ARP flow will be installed with the specified priority
- arp-flow-idle-timeout
  - The ARP flow will timeout (removed from the switch) if the flow doesn't forward a packet for *x* seconds

- arp-flow-hard-timeout
  - The ARP flow will timeout (removed from the switch) after *arp-flow-hard-timeout* seconds, regardless of how many packets it is forwarding

Further development

The **ProactiveFloodFlowWriter** needs to be improved. It does have the advantage of having less traffic come to the controller; however, it generates too much network traffic.

Address Tracker

Classes

- AddressTrackerModule
  - Reads config subsystem value for *observe-addresses-from*
  - If *observe-addresses-from* contains "arp", then an AddressObserverUsingArp is created
  - If *observe-addresses-from* contains "ipv4", then an AddressObserverUsingIpv4 is created
  - If *observe-addresses-from* contains "ipv6", then an AddressObserverUsingIpv6 is created
- AddressObserverUsingArp
  - Registers for ARP packet notifications
  - Uses **AddressObservationWriter** to write address observations from ARP packets
- AddressObserverUsingIpv4
  - Registers for IPv4 packet notifications
  - Uses **AddressObservationWriter** to write address observations from IPv4 packets
- AddressObserverUsingIpv6
  - Registers for IPv6 packet notifications
  - Uses **AddressObservationWriter** to write address observations from IPv6 packets
- AddressObservationWriter
  - Writes new Address Observations to the Inventory data tree
  - Updates existing Address Observations with updated "last seen" timestamps
    - Uses the *timestamp-update-intervval* configuration variable to determine whether or not to update

Configuration

- timestamp-update-interval
  - A last-seen timestamp is associated with each address. This last-seen timestamp will only be updated after *timestamp-update-interval* milliseconds.
  - A higher value has the advantage of performing less writes to the database.
  - A lower value has the advantage of knowing how fresh an address is.
- observe-addresses-from

  - IP and MAC addresses can be observed/learned from ARP, IPv4, and IPv6 packets. Set which packets to make these observations from.

Further development

Further improvements can be made to the **AddressObservationWriter** so that it (1) doesn't make any unnecessary writes to the DB and (2) is optimized for multi-threaded environments.

Validating changes to Address Tracker

Address Observations are added to the Inventory data tree.

The Address Observations on a Node Connector can be checked through a browser or a REST Client.

```
http://10.194.126.91:8080/restconf/operational/opendaylight-inventory:nodes/node/
↪openflow:1/node-connector/openflow:1:1
```

The Address Observations should still be there after changes.

Developer's Guide for Host Tracker

Validationg changes to Host Tracker

Host information is added to the Topology data tree.

- Host address
- Attachment point (link) to a node/switch

This host information and attachment point information can be checked through a browser or a REST Client.

```
http://10.194.126.91:8080/restconf/operational/network-topology:network-topology/
↪topology/flow:1/
```

Host information should still be there after changes.

L2Switch Main

Classes

- L2SwitchMainModule
  - Reads config subsystem value for *is-install-dropall-flow*
    - If *is-install-dropall-flow* is true, then an **InitialFlowWriter** is created
  - Reads config subsystem value for *is-learning-only-mode*
    - If *is-learning-only-mode* is false, then a **ReactiveFlowWriter** is created
- InitialFlowWriter
  - Only created when *is-install-dropall-flow* is true
  - Installs a flow, which drops all packets, on each switch. This flow has low priority and means that packets that don't match any higher-priority flows will simply be dropped.

- ReactiveFlowWriter
  - Reacts to network traffic and installs MAC-to-MAC flows on switches. These flows have matches based on MAC source and MAC destination.
  - Uses **FlowWriterServiceImpl** to write these flows to the switches
- FlowWriterService / FlowWriterServiceImpl
  - Writes flows to switches

Configuration

- is-install-dropall-flow
  - "true" means a drop-all flow will be installed on each switch, so the default action will be to drop a packet instead of sending it to the controller
  - "false" means this flow will not be installed
- dropall-flow-table-id
  - The dropall flow will be installed on the specified flow table of each switch
  - This field is only relevant when "is-install-dropall-flow" is set to "true"
- dropall-flow-priority
  - The dropall flow will be installed with the specified priority
  - This field is only relevant when "is-install-dropall-flow" is set to "true"
- dropall-flow-idle-timeout
  - The dropall flow will timeout (removed from the switch) if the flow doesn't forward a packet for *x* seconds
  - This field is only relevant when "is-install-dropall-flow" is set to "true"
- dropall-flow-hard-timeout
  - The dropall flow will timeout (removed from the switch) after *x* seconds, regardless of how many packets it is forwarding
  - This field is only relevant when "is-install-dropall-flow" is set to "true"
- is-learning-only-mode
  - "true" means that the L2Switch will only be learning addresses. No additional flows to optimize network traffic will be installed.
  - "false" means that the L2Switch will react to network traffic and install flows on the switches to optimize traffic. Currently, MAC-to-MAC flows are installed.
- reactive-flow-table-id
  - The reactive flow will be installed on the specified flow table of each switch
  - This field is only relevant when "is-learning-only-mode" is set to "false"
- reactive-flow-priority
  - The reactive flow will be installed with the specified priority
  - This field is only relevant when "is-learning-only-mode" is set to "false"
- reactive-flow-idle-timeout

  - The reactive flow will timeout (removed from the switch) if the flow doesn't forward a packet for *x* seconds
  - This field is only relevant when "is-learning-only-mode" is set to "false"
- reactive-flow-hard-timeout
  - The reactive flow will timeout (removed from the switch) after *x* seconds, regardless of how many packets it is forwarding
  - This field is only relevant when "is-learning-only-mode" is set to "false"

Further development

The **ReactiveFlowWriter** needs to be improved to install the MAC-to-MAC flows faster. For the first ping, the ARP request and reply are successful. However, then the ping packets are sent out. The first ping packet is dropped sometimes because the MAC-to-MAC flow isn't installed quickly enough. The second, third, and following ping packets are successful though.

API Reference Documentation

Further documentation can be found by checking out the L2Switch project.

Checking out the L2Switch project

```
git clone https://git.opendaylight.org/gerrit/p/l2switch.git
```

The above command will create a directory called "l2switch" with the project.

Testing your changes to the L2Switch project

Running the L2Switch project

To run the base distribution, you can use the following command

```
./distribution/base/target/distributions-l2switch-base-0.1.0-SNAPSHOT-osgipackage/
↪opendaylight/run.sh
```

If you need additional resources, you can use these command line arguments:

```
-Xms1024m -Xmx2048m -XX:PermSize=512m -XX:MaxPermSize=1024m'
```

To run the karaf distribution, you can use the following command:

```
./distribution/karaf/target/assembly/bin/karaf
```

Create a network using mininet

```
sudo mn --controller=remote,ip=<Controller IP> --topo=linear,3 --switch ovsk,
↪protocols=OpenFlow13
sudo mn --controller=remote,ip=127.0.0.1 --topo=linear,3 --switch ovsk,
↪protocols=OpenFlow13
```

The above command will create a virtual network consisting of 3 switches. Each switch will connect to the controller located at the specified IP, i.e. 127.0.0.1

```
sudo mn --controller=remote,ip=127.0.0.1 --mac --topo=linear,3 --switch ovsk,
↪protocols=OpenFlow13
```

The above command has the "mac" option, which makes it easier to distinguish between Host MAC addresses and Switch MAC addresses.

Generating network traffic using mininet

```
h1 ping h2
```

The above command will cause host1 (h1) to ping host2 (h2)

```
pingall
```

pingall will cause each host to ping every other host.

Miscellaneous mininet commands

```
link s1 s2 down
```

This will bring the link between switch1 (s1) and switch2 (s2) down

```
link s1 s2 up
```

This will bring the link between switch1 (s1) and switch2 (s2) up

```
link s1 h1 down
```

This will bring the link between switch1 (s1) and host1 (h1) down

LACP Developer Guide

LACP Overview

The OpenDaylight LACP (Link Aggregation Control Protocol) project can be used to aggregate multiple links between OpenDaylight controlled network switches and LACP enabled legacy switches or hosts operating in active LACP mode.

OpenDaylight LACP passively negotiates automatic bundling of multiple links to form a single LAG (Link Aggregation Group). LAGs are realised in the OpenDaylight controlled switches using OpenFlow 1.3+ group table functionality.

LACP Architecture

- **inventory**
  - Maintains list of OpenDaylight controlled switches and port information
  - List of LAGs created and physical ports that are part of the LAG

  - Interacts with MD-SAL to update LACP related information

- **inventorylistener**
  - This module interacts with MD-SAL for receiving node/node-connector notifications
- **flow**
  - Programs the switch to punt LACP PDU (Protocol Data Unit) to controller
- **packethandler**
  - Receives and transmits LACP PDUs to the LACP enabled endpoint
  - Provides infrastructure services for group table programming
- **core**
  - Performs LACP state machine processing

How LAG programming is implemented

The LAG representing the aggregated multiple physical ports are realized in the OpenDaylight controlled switches by creating a group table entry (Group table supported from OpenFlow 1.3 onwards). The group table entry has a group type **Select** and action referring to the aggregated physical ports. Any data traffic to be sent out through the LAG can be sent through the **group entry** available for the LAG.

Suppose there are ports P1-P8 in a node. When LACP project is installed, a group table entry for handling broadcast traffic is automatically created on all the switches that have registered to the controller.

GroupID	GroupType	EgressPorts
<B'castgID>	ALL	P1,P2,...P8

Now, assume P1 & P2 are now part of LAG1. The group table would be programmed as follows:

GroupID	GroupType	EgressPorts
<B'castgID>	ALL	P3,P4,...P8
<LAG1>	SELECT	P1,P2

When a second LAG, LAG2, is formed with ports P3 and P4,

GroupID	GroupType	EgressPorts
<B'castgID>	ALL	P5,P6,...P8
<LAG1>	SELECT	P1,P2
<LAG2>	SELECT	P3,P4

How applications can program OpenFlow flows using LACP-created LAG groups

OpenDaylight controller modules can get the information of LAG by listening/querying the LACP Aggregator datastore.

When any application receives packets, it can check, if the ingress port is part of a LAG by verifying the LAG Aggregator reference (lacp-agg-ref) for the source nodeConnector that OpenFlow plugin provides.

When applications want to add flows to egress out of the LAG, they must use the group entry corresponding to the LAG.

From the above example, for a flow to egress out of LAG1,

add-flow eth_type=<xxxx>,ip_dst=<x.x.x.x>,actions=output:<LAG1>

Similarly, when applications want traffic to be broadcasted, they should use the group table entries **<B'castgID>,<LAG1>,<LAG2>** in output action.

For all applications, the group table information is accessible from LACP Aggregator datastore.

NEtwork MOdeling (NEMO)

Overview

The NEMO engine provides REST APIs to express intent, and manage it. With this northbound API, user could query what intents have been handled successfully, and what types have been predefined.

NEMO Architecture

In NEMO project, it provides three features facing developer.

- `odl-nemo-engine`: it is a whole model to handle intent.
- `odl-nemo-openflow-renderer`: it is a southbound render to translate intent to flow table in devices supporting for OpenFlow protocol.
- `odl-nemo-cli-render`: it is also a southbound render to translate intent into forwarding table in devices supporting for traditional protocol.

Key APIs and Interfaces

NEMO projects provide four basic REST methods for user to use.

- PUT: store the information expressed in NEMO model directly without handled by NEMO engine.
- POST: the information expressed in NEMO model will be handled by NEMO engine, and will be translated into southbound configuration.
- GET: obtain the data stored in data store.
- DELETE: delete the data in data store.

NEMO Intent API

NEMO provides several RPCs to handle user's intent. All RPCs use POST method.

- `http://{controller-ip}:8181/restconf/operations/nemo-intent:register-user`: a REST API to register a new user. It is the first and necessary step to express intent.
- `http://{controller-ip}:8181/restconf/operations/nemo-intent:transaction-begin`: a REST type to start a transaction. The intent exist in the transaction will be handled together.
- `http://{controller-ip}:8181/restconf/operations/nemo-intent:transaction-end`: a REST API to end a transaction. The intent exist in the transaction will be handled together.
- `http://{controller-ip}:8181/restconf/operations/nemo-intent:structure-style-nemo-updat` a REST API to create, import or update intent in a structure style, that is, user could express the structure of intent in json body.
- `http://{controller-ip}:8181/restconf/operations/nemo-intent:structure-style-nemo-delet` a REST API to delete intent in a structure style.

- `http://{controller-ip}:8181/restconf/operations/nemo-intent:language-style-nemo-reques` a REST API to create, import, update and delete intent in a language style, that is, user could express intent with NEMO script. On the other hand, with this interface, user could query which intent have been handled successfully.

API Reference Documentation

Go to `http://${IPADDRESS}:8181/apidoc/explorer/index.html`. User could see many useful APIs to deploy or query intent.

NETCONF Developer Guide

Note: Reading the NETCONF section in the User Guide is likely useful as it contains an overview of NETCONF in OpenDaylight and a how-to for spawning and configuring NETCONF connectors.

This chapter is recommended for application developers who want to interact with mounted NETCONF devices from their application code. It tries to demonstrate all the use cases from user guide with RESTCONF but now from the code level. One important difference would be the demonstration of NETCONF notifications and notification listeners. The notifications were not shown using RESTCONF because **RESTCONF does not support notifications from mounted NETCONF devices.**

Note: It may also be useful to read the generic OpenDaylight MD-SAL app development tutorial before diving into this chapter. This guide assumes awareness of basic OpenDaylight application development.

Sample app overview

All the examples presented here are implemented by a sample OpenDaylight application called **ncmount** in the `coretutorials` OpenDaylight project. It can be found on the github mirror of OpenDaylight's repositories:

- https://github.com/opendaylight/coretutorials/tree/stable/boron/ncmount

or checked out from the official OpenDaylight repository:

- https://git.opendaylight.org/gerrit/#/admin/projects/coretutorials

The application was built using the project startup maven archetype **and demonstrates how to:**

- preconfigure connectors to NETCONF devices
- retrieve MountPointService (registry of available mount points)
- listen and react to changing connection state of netconf-connector
- add custom device YANG models to the app and work with them
- read data from device in binding aware format (generated java APIs from provided YANG models)
- write data into device in binding aware format
- trigger and listen to NETCONF notifications in binding aware format

Detailed information about the structure of the application can be found at: https://wiki.opendaylight.org/view/Controller_Core_Functionality_Tutorials:Tutorials:Netconf_Mount

Note: The code in ncmount is fully **binding aware** (works with generated java APIs from provided YANG models). However it is also possible to perform the same operations in **binding independent** manner.

NcmountProvider

The NcmountProvider class (found in NcmountProvider.java) is the central point of the ncmount application and all the application logic is contained there. The following sections will detail its most interesting pieces.

Retrieve MountPointService

The MountPointService is a central registry of all available mount points in OpenDaylight. It is just another MD-SAL service and is available from the `session` attribute passed by `onSessionInitiated` callback:

```
@Override
public void onSessionInitiated(ProviderContext session) {
    LOG.info("NcmountProvider Session Initiated");

    // Get references to the data broker and mount service
    this.mountService = session.getSALService(MountPointService.class);

    ...

    }
}
```

Listen for connection state changes

It is important to know when a mount point appears, when it is fully connected and when it is disconnected or removed. The exact states of a mount point are:

- Connected
- Connecting
- Unable to connect

To receive this kind of information, an application has to register itself as a notification listener for the preconfigured netconf-topology subtree in MD-SAL's datastore. This can be performed in the `onSessionInitiated` callback as well:

```
@Override
public void onSessionInitiated(ProviderContext session) {

    ...

    this.dataBroker = session.getSALService(DataBroker.class);

    // Register ourselves as the REST API RPC implementation
    this.rpcReg = session.addRpcImplementation(NcmountService.class, this);

    // Register ourselves as data change listener for changes on Netconf
    // nodes. Netconf nodes are accessed via "Netconf Topology" - a special
    // topology that is created by the system infrastructure. It contains
```

```
    // all Netconf nodes the Netconf connector knows about. NETCONF_TOPO_IID
    // is equivalent to the following URL:
    // .../restconf/operational/network-topology:network-topology/topology/topology-
↪netconf
    if (dataBroker != null) {
        this.dclReg = dataBroker.registerDataChangeListener(LogicalDatastoreType.
↪OPERATIONAL,
                NETCONF_TOPO_IID.child(Node.class),
                this,
                DataChangeScope.SUBTREE);
    }
}
```

The implementation of the callback from MD-SAL when the data change can be found in the `onDataChanged(AsyncDataChangeEvent<InstanceIdentifier<?>, DataObject> change)` callback of NcmountProvider class.

Reading data from the device

The first step when trying to interact with the device is to get the exact mount point instance (identified by an instance identifier) from the MountPointService:

```
@Override
public Future<RpcResult<ShowNodeOutput>> showNode(ShowNodeInput input) {
    LOG.info("showNode called, input {}", input);

    // Get the mount point for the specified node
    // Equivalent to '.../restconf/<config | operational>/opendaylight-
↪inventory:nodes/node/<node-name>/yang-ext:mount/'
    // Note that we can read both config and operational data from the same
    // mount point
    final Optional<MountPoint> xrNodeOptional = mountService.getMountPoint(NETCONF_
↪TOPO_IID
            .child(Node.class, new NodeKey(new NodeId(input.getNodeName()))));

    Preconditions.checkArgument(xrNodeOptional.isPresent(),
            "Unable to locate mountpoint: %s, not mounted yet or not configured",
            input.getNodeName());
    final MountPoint xrNode = xrNodeOptional.get();

    ....
}
```

Note: The triggering method in this case is called `showNode`. It is a YANG-defined RPC and NcmountProvider serves as an MD-SAL RPC implementation among other things. This means that `showNode` an be triggered using RESTCONF.

The next step is to retrieve an instance of the `DataBroker` API from the mount point and start a read transaction:

```
@Override
public Future<RpcResult<ShowNodeOutput>> showNode(ShowNodeInput input) {

    ...
```

```
    // Get the DataBroker for the mounted node
    final DataBroker xrNodeBroker = xrNode.getService(DataBroker.class).get();
    // Start a new read only transaction that we will use to read data
    // from the device
    final ReadOnlyTransaction xrNodeReadTx = xrNodeBroker.newReadOnlyTransaction();

    ...
}
```

Finally, it is possible to perform the read operation:

```
@Override
public Future<RpcResult<ShowNodeOutput>> showNode(ShowNodeInput input) {

    ...

    InstanceIdentifier<InterfaceConfigurations> iid =
            InstanceIdentifier.create(InterfaceConfigurations.class);

    Optional<InterfaceConfigurations> ifConfig;
    try {
        // Read from a transaction is asynchronous, but a simple
        // get/checkedGet makes the call synchronous
        ifConfig = xrNodeReadTx.read(LogicalDatastoreType.CONFIGURATION, iid).
↪checkedGet();
    } catch (ReadFailedException e) {
        throw new IllegalStateException("Unexpected error reading data from " + input.
↪getNodeName(), e);
    }

    ...
}
```

The instance identifier is used here again to specify a subtree to read from the device. At this point application can process the data as it sees fit. The ncmount app transforms the data into its own format and returns it from `showNode`.

Note: More information can be found in the source code of ncmount sample app + on wiki: https://wiki.opendaylight.org/view/Controller_Core_Functionality_Tutorials:Tutorials:Netconf_Mount

---

Network Intent Composition (NIC) Developer Guide

Overview

The Network Intent Composition (NIC) provides four features:

- odl-nic-core-hazelcast: Provides a distributed intent mapping service, implemented using hazelcast, that stores metadata needed by odl-nic-core feature.
- odl-nic-core-mdsal: Provides an intent rest API to external applications for CRUD operations on intents, conflict resolution and event handling. Uses MD-SAL as backend.
- odl-nic-console: Provides a karaf CLI extension for intent CRUD operations and mapping service operations.
- odl-nic-renderer-of - Generic OpenFlow Renderer.
- odl-nic-renderer-vtn - a feature that transforms an intent to a network modification using the VTN project

- odl-nic-renderer-gbp - a feature that transforms an intent to a network modification using the Group Policy project
- odl-nic-renderer-nemo - a feature that transforms an intent to a network modification using the NEMO project
- odl-nic-listeners - adds support for event listening. (depends on: odl-nic-renderer-of)
- odl-nic-neutron-integration - allow integration with openstack neutron to allow coexistence between existing neutron security rules and intents pushed by ODL applications.

Only a single renderer feature should be installed at a time for the Boron release.

odl-nic-core-mdsal XOR odl-nic-core-hazelcast

This feature supplies the base models for the Network Intent Composition (NIC) capability. This includes the definition of intent as well as the configuration and operational data trees.

This feature only provides an information model. The interface for NIC is to modify the information model via the configuraiton data tree, which will trigger the renderer to make the appropriate changes in the controlled network.

Installation

First you need to install one of the core installations:

```
feature:install odl-nic-core-service-mdsal odl-nic-console
```

OR

```
feature:install odl-nic-core-service-hazelcast odl-nic-console
```

Then pick a renderer:

```
feature:install odl-nic-listeners (will install odl-nic-renderer-of)
```

OR

```
feature:install odl-nic-renderer-vtn
```

OR

```
feature:install odl-nic-renderer-gbp
```

OR

```
feature:install odl-nic-renderer-nemo
```

REST Supported operations

POST / PUT (configuration)

This operations create instances of an intent in the configuration data tree and trigger the creation or modification of an intent.

GET (configuration / operational)

This operation lists all or fetches a single intent from the data tree.

DELETE (configuration)

This operation will cause an intent to be removed from the system and trigger any configuration changes on the network rendered from this intent to be removed.

odl-nic-cli user guide

This feature provides karaf console CLI command to manipulate the intent data model. The CLI essentailly invokes the equivalent data operations.

intent:add

Creates a new intent in the configuration data tree

```
DESCRIPTION
        intent:add

    Adds an intent to the controller.

Examples: --actions [ALLOW] --from <subject> --to <subject>
          --actions [BLOCK] --from <subject>

SYNTAX
        intent:add [options]

OPTIONS
        -a, --actions
                Action to be performed.
                -a / --actions BLOCK/ALLOW
                (defaults to [BLOCK])
        --help
                Display this help message
        -t, --to
                Second Subject.
                -t / --to <subject>
                (defaults to any)
        -f, --from
                First subject.
                -f / --from <subject>
                (defaults to any)
```

intent:delete

Removes an existing intent from the system

```
DESCRIPTION
        intent:remove
```

```
    Removes an intent from the controller.

SYNTAX
        intent:remove id

ARGUMENTS
        id  Intent Id
```

intent:list

Lists all the intents in the system

```
DESCRIPTION
        intent:list

    Lists all intents in the controller.

SYNTAX
        intent:list [options]

OPTIONS
        -c, --config
                List Configuration Data (optional).
                -c / --config <ENTER>
        --help
                Display this help message
```

intent:show

Displays the details of a single intent

```
DESCRIPTION
        intent:show

    Shows detailed information about an intent.

SYNTAX
        intent:show id

ARGUMENTS
        id  Intent Id
```

intent:map

List/Add/Delete current state from/to the mapping service.

```
DESCRIPTION
        intent:map

        List/Add/Delete current state from/to the mapping service.

SYNTAX
```

```
        intent:map [options]

         Examples: --list, -l [ENTER], to retrieve all keys.
                   --add-key <key> [ENTER], to add a new key with empty contents.
                   --del-key <key> [ENTER], to remove a key with it's values."
                   --add-key <key> --value [<value 1>, <value 2>, ...] [ENTER],
                     to add a new key with some values (json format).
OPTIONS
       --help
           Display this help message
       -l, --list
           List values associated with a particular key.
       -l / --filter <regular expression> [ENTER]
       --add-key
           Adds a new key to the mapping service.
       --add-key <key name> [ENTER]
       --value
           Specifies which value should be added/delete from the mapping service.
       --value "key=>value"... --value "key=>value" [ENTER]
           (defaults to [])
       --del-key
           Deletes a key from the mapping service.
       --del-key <key name> [ENTER]
```

Sample Use case: MPLS

Description

The scope of this use-case is to add MPLS intents between two MPLS endpoints. The use-case tries to address the real-world scenario illustrated in the diagram below:

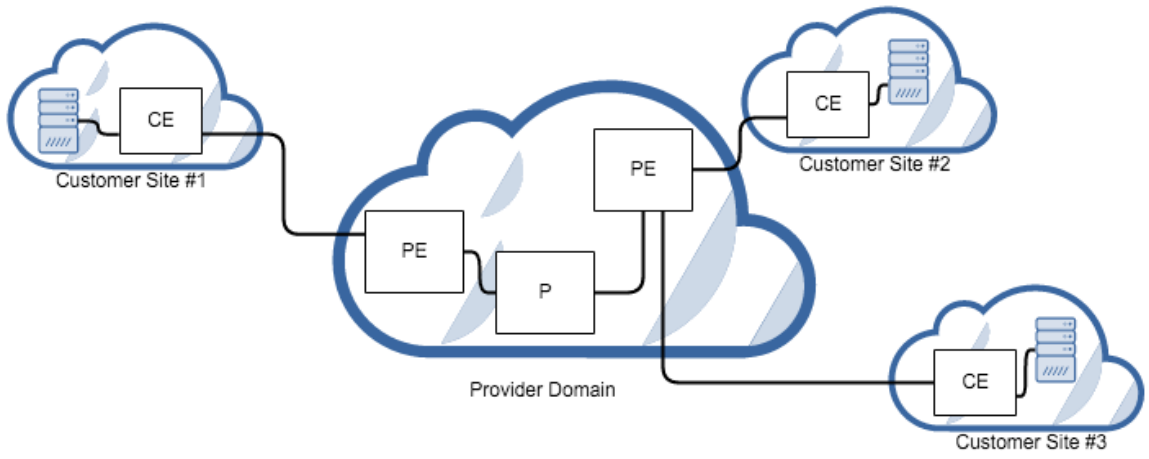


Fig. 2.27: MPLS VPN Service Diagram

where PE (Provider Edge) and P (Provider) switches are managed by OpenDaylight. In NIC's terminology the endpoints are the PE switches. There could be many P switches between the PEs.

In order for NIC to recognize endpoints as MPLS endpoints, the user is expected to add mapping information about the PE switches to NIC's mapping service to include the below properties:

1. MPLS Label to identify a PE
2. IPv4 Prefix for the customer site that are connected to a PE
3. Switch-Port: Ingress (or Egress) for source (or Destination) endpoint of the source (or Destination) PE

An intent:add between two MPLS endpoints renders OpenFlow rules for: 1. push/pop labels to the MPLS endpoint nodes after an IPv4 Prefix match. 2. forward to port rule after MPLS label match to all the switches that form the shortest path between the endpoints (calculated using Dijkstra algorithm).

Additionally, we have also added constraints to Intent model for protection and failover mechanism to ensure end-to-end connectivity between endpoints. By specifying these constraints to intent:add the use-case aims to reduces the risk of connectivity failure due to a single link or port down event on a forwarding device.

- Protection constraint: Constraint that requires an end-to-end connectivity to be protected by providing redundant paths.
- Failover constraint: Constraint that specifies the type of failover implementation. slow-reroute: Uses disjoint path calculation algorithms like Suurballe to provide alternate end-to-end routes. fast-reroute: Uses failure detection feature in hardware forwarding device through OF group table features (Future plans) When no constraint is requested by the user we default to offering a since end-to-end route using Dijkstra shortest path.

How to use it?

1. Start Karaf and install related features:

```
feature:install odl-nic-core-service-mdsal odl-nic-core odl-nic-console odl-nic-
↪listeners
feature:install  odl-dlux-core odl-dluxapps-applications
```

2. Start mininet topology and verify in DLUX Topology page for the nodes and link.

```
mn --controller=remote,ip=$CONTROLLER_IP --custom ~/shortest_path.py --topo␣
↪shortest_path --switch ovsk,protocols=OpenFlow13
```

```
cat shortest.py -->
from mininet.topo import Topo
from mininet.cli import CLI
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.util import irange,dumpNodeConnections
from mininet.log import setLogLevel
```

```
class Fast_Failover_Demo_Topo(Topo):
```

```
def __init__(self):
    # Initialize topology and default options
    Topo.__init__(self)
```

```
s1 = self.addSwitch('s1',dpid='0000000000000001')
s2a = self.addSwitch('s2a',dpid='000000000000002a')
```

```
s2b = self.addSwitch('s2b',dpid='000000000000002b')
s2c = self.addSwitch('s2c',dpid='000000000000002c')
s3 = self.addSwitch('s3',dpid='0000000000000003')
self.addLink(s1, s2a)
self.addLink(s1, s2b)
self.addLink(s2b, s2c)
self.addLink(s3, s2a)
self.addLink(s3, s2c)
host_1 = self.addHost('h1',ip='10.0.0.1',mac='10:00:00:00:00:01')
host_2 = self.addHost('h2',ip='10.0.0.2',mac='10:00:00:00:00:02')
self.addLink(host_1, s1)
self.addLink(host_2, s3)
```

```
topos = { 'shortest_path': ( lambda: Demo_Topo() ) }
```

3. Update the mapping service with required information

   Sample payload:

```
{
  "mappings": {
    "outer-map": [
      {
        "id": "uva",
        "inner-map": [
          {
            "inner-key": "ip_prefix",
            "value": "10.0.0.1/32"
          },
          {
            "inner-key": "mpls_label",
            "value": "15"
          },
          {
            "inner-key": "switch_port",
            "value": "openflow:1:1"
          }
        ]
      },
      {
        "id": "eur",
        "inner-map": [
          {
            "inner-key": "ip_prefix",
            "value": "10.0.0.2/32"
          },
          {
            "inner-key": "mpls_label",
            "value": "16"
          },
          {
            "inner-key": "switch_port",
            "value": "openflow:3:1"
          }
        ]
      }
    ]
  }
```

```
}
```

4. Create bidirectional Intents using Karaf command line or RestCONF:

   Example:

```
intent:add -f uva -t eur -a ALLOW
intent:add -f eur -t uva -a ALLOW
```

5. Verify by running ovs-ofctl command on mininet if the flows were pushed correctly to the nodes that form the shortest path.

   Example:

```
ovs-ofctl -O OpenFlow13 dump-flows s1
```

NetIDE Developer Guide

Overview

The NetIDE Network Engine enables portability and cooperation inside a single network by using a client/server multi-controller SDN architecture. Separate "Client SDN Controllers" host the various SDN Applications with their access to the actual physical network abstracted and coordinated through a single "Server SDN Controller", in this instance OpenDaylight. This allows applications written for Ryu/Floodlight/Pyretic to execute on OpenDaylight managed infrastructure.

The "Network Engine" is modular by design:

- An OpenDaylight plugin, "shim", sends/receives messages to/from subscribed SDN Client Controllers. This consumes the ODL OpenFlow Plugin
- An initial suite of SDN Client Controller "Backends": Floodlight, Ryu, Pyretic. Further controllers may be added over time as the engine is extensible.

The Network Engine provides a compatibility layer capable of translating calls of the network applications running on top of the client controllers, into calls for the server controller framework. The communication between the client and the server layers is achieved through the NetIDE intermediate protocol, which is an application-layer protocol on top of TCP that transmits the network control/management messages from the client to the server controller and vice-versa. Between client and server controller sits the Core Layer which also "speaks" the intermediate protocol. The core layer implements three main functions:

1. interfacing with the client backends and server shim, controlling the lifecycle of controllers as well as modules in them,
2. orchestrating the execution of individual modules (in one client controller) or complete applications (possibly spread across multiple client controllers),
3. interfacing with the tools.

NetIDE Intermediate Protocol

The Intermediate Protocol serves several needs, it has to:

1. carry control messages between core and shim/backend, e.g., to start up/take down a particular module, providing unique identifiers for modules,

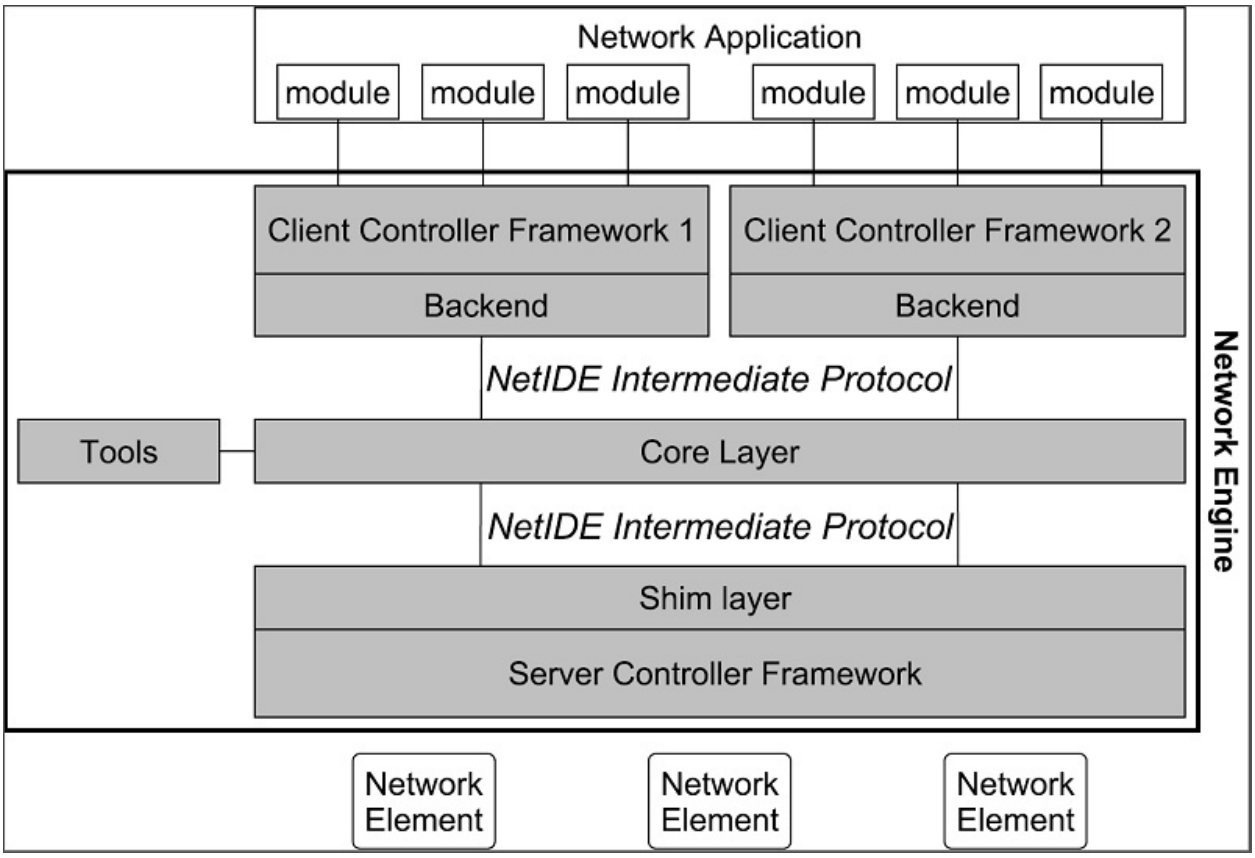


Fig. 2.28: NetIDE Network Engine Architecture

2. carry event and action messages between shim, core, and backend, properly demultiplexing such messages to the right module based on identifiers,
3. encapsulate messages specific to a particular SBI protocol version (e.g., OpenFlow 1.X, NETCONF, etc.) towards the client controllers with proper information to recognize these messages as such.

The NetIDE packages can be added as dependencies in Maven projects by putting the following code in the *pom.xml* file.

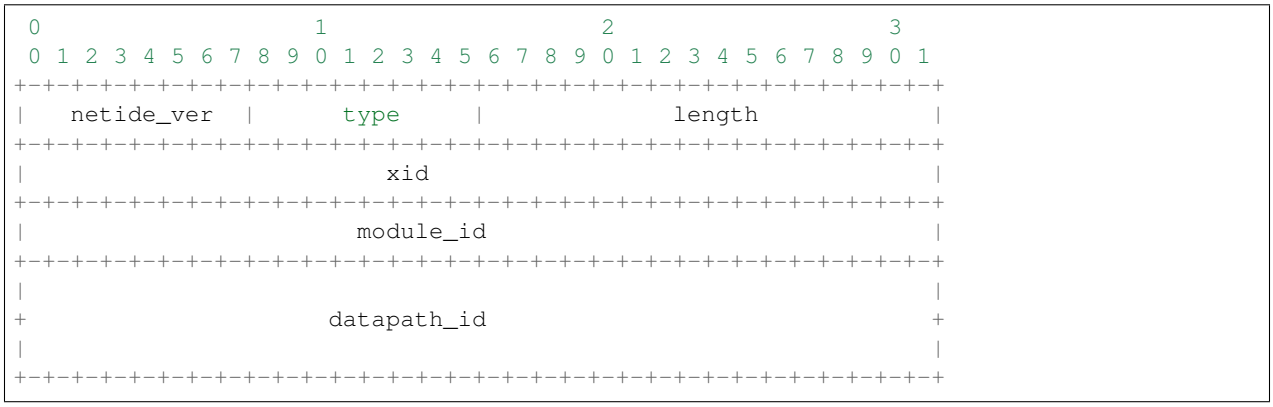
```
<dependency>
    <groupId>org.opendaylight.netide</groupId>
    <artifactId>api</artifactId>
    <version>${NETIDE_VERSION}</version>
</dependency>
```

The current stable version for NetIDE is `0.2.0-Boron`.

Protocol specification

Messages of the NetIDE protocol contain two basic elements: the NetIDE header and the data (or payload). The NetIDE header, described below, is placed before the payload and serves as the communication and control link between the different components of the Network Engine. The payload can contain management messages, used by the components of the Network Engine to exchange relevant information, or control/configuration messages (such as OpenFlow, NETCONF, etc.) crossing the Network Engine generated by either network application modules or by the network elements.

The NetIDE header is defined as follows:



where each tick mark represents one bit position. Alternatively, in a C-style coding format, the NetIDE header can be represented with the following structure:

```
struct netide_header {
    uint8_t netide_ver ;
    uint8_t type ;
    uint16_t length ;
    uint32_t xid
    uint32_t module_id
    uint64_t datapath_id
};
```

- `netide_ver` is the version of the NetIDE protocol (the current version is v1.2, which is identified with value 0x03).
- `length` is the total length of the payload in bytes.

- `type` contains a code that indicates the type of the message according with the following values:

```
enum type {
    NETIDE_HELLO = 0x01 ,
    NETIDE_ERROR = 0x02 ,
    NETIDE_MGMT = 0x03 ,
    MODULE_ANNOUNCEMENT = 0x04 ,
    MODULE_ACKNOWLEDGE = 0x05 ,
    NETIDE_HEARTBEAT = 0x06 ,
    NETIDE_OPENFLOW = 0x11 ,
    NETIDE_NETCONF = 0x12 ,
    NETIDE_OPFLEX = 0x13
};
```

- `datapath_id` is a 64-bit field that uniquely identifies the network elements.
- `module_id` is a 32-bits field that uniquely identifies Backends and application modules running on top of each client controller. The composition mechanism in the core layer leverages on this field to implement the correct execution flow of these modules.
- `xid` is the transaction identifier associated to the each message. Replies must use the same value to facilitate the pairing.

Module announcement

The first operation performed by a Backend is registering itself and the modules that it is running to the Core. This is done by using the `MODULE_ANNOUNCEMENT` and `MODULE_ACKNOWLEDGE` message types. As a result of this process, each Backend and application module can be recognized by the Core through an identifier (the `module_id`) placed in the NetIDE header. First, a Backend registers itself by using the following schema: backend-<platform name>-<pid>.

For example,odule a Ryu Backend will register by using the following name in the message backend-ryu-12345 where 12345 is the process ID of the registering instance of the Ryu platform. The format of the message is the following:

```
struct NetIDE_message {
    netide_ver = 0x03
    type = MODULE_ANNOUNCEMENT
    length = len(" backend -< platform_name >-<pid >")
    xid = 0
    module_id = 0
    datapath_id = 0
    data = " backend -< platform_name >-<pid >"
}
```

The answer generated by the Core will include a `module_id` number and the Backend name in the payload (the same indicated in the `MODULE_ANNOUNCEMENT` message):

```
struct NetIDE_message {
    netide_ver = 0x03
    type = MODULE_ACKNOWLEDGE
    length = len(" backend -< platform_name >-<pid >")
    xid = 0
    module_id = MODULE_ID
    datapath_id = 0
    data = " backend -< platform_name >-<pid >"
}
```

Once a Backend has successfully registered itself, it can start registering its modules with the same procedure described above by indicating the name of the module in the data (e.g. data="Firewall"). From this point on, the Backend will insert its own `module_id` in the header of the messages it generates (e.g. heartbeat, hello messages, OpenFlow echo messages from the client controllers, etc.). Otherwise, it will encapsulate the control/configuration messages (e.g. FlowMod, PacketOut, FeatureRequest, NETCONF request, etc.) generated by network application modules with the specific +module_id+s.

Heartbeat

The heartbeat mechanism has been introduced after the adoption of the ZeroMQ messaging queuing library to transmit the NetIDE messages. Unfortunately, the ZeroMQ library does not offer any mechanism to find out about disrupted connections (and also completely unresponsive peers). This limitation of the ZeroMQ library can be an issue for the Core's composition mechanism and for the tools connected to the Network Engine, as they cannot understand when an client controller disconnects or crashes. As a consequence, Backends must periodically send (let's say every 5 seconds) a "heartbeat" message to the Core. If the Core does not receive at least one "heartbeat" message from the Backend within a certain timeframe, the Core considers it disconnected, removes all the related data from its memory structures and informs the relevant tools. The format of the message is the following:

```
struct NetIDE_message {
    netide_ver = 0x03
    type = NETIDE_HEARTBEAT
    length = 0
    xid = 0
    module_id = backend -id
    datapath_id = 0
    data = 0
}
```

Handshake

Upon a successful connection with the Core, the client controller must immediately send a hello message with the list of the control and/or management protocols needed by the applications deployed on top of it.

```
struct NetIDE_message {
    struct netide_header header ;
    uint8 data [0]
};
```

The header contains the following values:

- `netide ver=0x03`
- `type=NETIDE_HELLO`
- `length=2*NR_PROTOCOLS`
- `data` contains one 2-byte word (in big endian order) for each protocol, with the first byte containing the code of the protocol according to the above enum, while the second byte in- dictates the version of the protocol (e.g. according to the ONF specification, 0x01 for OpenFlow v1.0, 0x02 for OpenFlow v1.1, etc.). NETCONF version is marked with 0x01 that refers to the specification in the RFC6241, while OpFlex version is marked with 0x00 since this protocol is still in work-in-progress stage.

The Core relays hello messages to the server controller which responds with another hello message containing the following:

- `netide ver=0x03`

- `type=NETIDE_HELLO`
- `length=2*NR_PROTOCOLS`

If at least one of the protocols requested by the client is supported. In particular, `data` contains the codes of the protocols that match the client's request (2-bytes words, big endian order). If the hand- shake fails because none of the requested protocols is supported by the server controller, the header of the answer is as follows:

- `netide ver=0x03`
- `type=NETIDE_ERROR`
- `length=2*NR_PROTOCOLS`
- `data` contains the codes of all the protocols supported by the server controller (2-bytes words, big endian order). In this case, the TCP session is terminated by the server controller just after the answer is received by the client. ‘

NetVirt Developer Guide

Neutron Service Developer Guide

Overview

This Karaf feature (`odl-neutron-service`) provides integration support for OpenStack Neutron via the OpenDaylight ML2 mechanism driver. The Neutron Service is only one of the components necessary for OpenStack integration. It defines YANG models for OpenStack Neutron data models and northbound API via REST API and YANG model RESTCONF.

Those developers who want to add new provider for new OpenStack Neutron extensions/services (Neutron constantly adds new extensions/services and OpenDaylight will keep up with those new things) need to communicate with this Neutron Service or add models to Neutron Service. If you want to add new extensions/services themselves to the Neutron Service, new YANG data models need to be added, but that is out of scope of this document because this guide is for a developer who will be *using* the feature to build something separate, but *not* somebody who will be developing code for this feature itself.

Neutron Service Architecture

The Neutron Service defines YANG models for OpenStack Neutron integration. When OpenStack admins/users request changes (creation/update/deletion) of Neutron resources, e.g., Neutron network, Neutron subnet, Neutron port, the corresponding YANG model within OpenDaylight will be modified. The OpenDaylight OpenStack will subscribe the changes on those models and will be notified those modification through MD-SAL when changes are made. Then the provider will do the necessary tasks to realize OpenStack integration. How to realize it (or even not realize it) is up to each provider. The Neutron Service itself does not take care of it.

How to Write a SB Neutron Consumer

In Boron, there is only one options for SB Neutron Consumers:

- Listening for changes via the Neutron YANG model

Until Beryllium there was another way with the legacy I*Aware interface. From Boron, the interface was eliminated. So all the SB Neutron Consumers have to use Neutron YANG model.

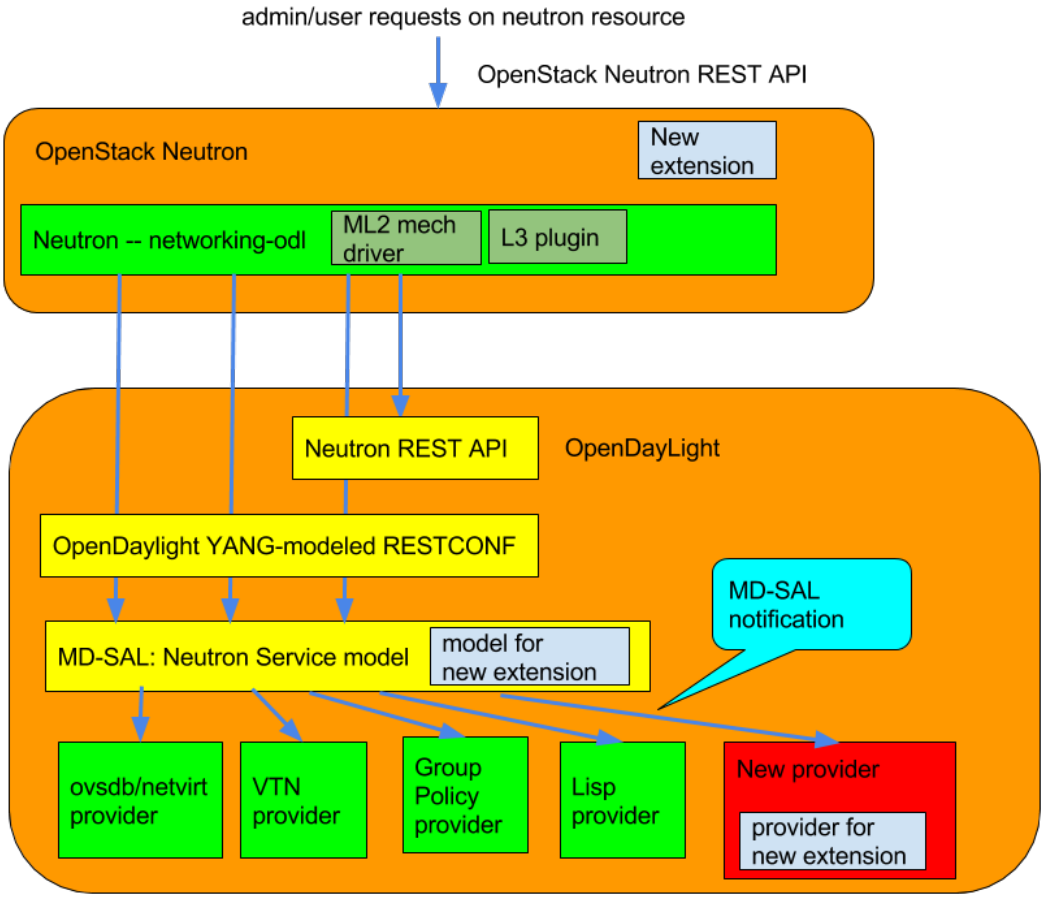


Fig. 2.29: Neutron Service Architecture

Neutron YANG models

Neutron service defines YANG models for Neutron. The details can be found at

- https://git.opendaylight.org/gerrit/gitweb?p=neutron.git;a=tree;f=model/src/main/yang;hb=refs/heads/stable/boron

Basically those models are based on OpenStack Neutron API definitions. For exact definitions, OpenStack Neutron source code needs to be referred as the above documentation doesn't always cover the necessary details. There is nothing special to utilize those Neutron YANG models. The basic procedure will be:

1. subscribe for changes made to the the model
2. respond on the data change notification for each models

Note: Currently there is no way to refuse the request configuration at this point. That is left to future work.

```
public class NeutronNetworkChangeListener implements DataChangeListener,␣
↪AutoCloseable {
    private ListenerRegistration<DataChangeListener> registration;
    private DataBroker db;

    public NeutronNetworkChangeListener(DataBroker db){
        this.db = db;
        // create identity path to register on service startup
        InstanceIdentifier<Network> path = InstanceIdentifier
                .create(Neutron.class)
                .child(Networks.class)
                .child(Network.class);
        LOG.debug("Register listener for Neutron Network model data changes");
        // register for Data Change Notification
        registration =
                this.db.registerDataChangeListener(LogicalDatastoreType.CONFIGURATION,
↪ path, this, DataChangeScope.ONE);

    }

    @Override
    public void onDataChanged(
            AsyncDataChangeEvent<InstanceIdentifier<?>, DataObject> changes) {
        LOG.trace("Data changes : {}",changes);

        // handle data change notification
        Object[] subscribers = NeutronIAwareUtil.getInstances(INeutronNetworkAware.
↪class, this);
        createNetwork(changes, subscribers);
        updateNetwork(changes, subscribers);
        deleteNetwork(changes, subscribers);
    }
}
```

Neutron configuration

From Boron, new models of configuration for OpenDaylight to tell OpenStack neutron/networking-odl its configuration/capability.

hostconfig

This is for OpenDaylight to tell per-node configuration to Neutron. Especially this is used by pseudo agent port binding heavily.

The model definition can be found at

- https://git.opendaylight.org/gerrit/gitweb?p=neutron.git;a=blob;f=model/src/main/yang/neutron-hostconfig.yang;hb=refs/heads/stable/boron

How to populate this for pseudo agent port binding is documented at

- http://git.openstack.org/cgit/openstack/networking-odl/tree/doc/source/devref/hostconfig.rst

Neutron extension config

In Boron this is experimental. The model definition can be found at

- https://git.opendaylight.org/gerrit/gitweb?p=neutron.git;a=blob;f=model/src/main/yang/neutron-extensions.yang;hb=refs/heads/stable/boron

Each Neutron Service provider has its own feature set. Some support the full features of OpenStack, but others support only a subset. With same supported Neutron API, some functionality may or may not be supported. So there is a need for a way that OpenDaylight can tell networking-odl its capability. Thus networking-odl can initialize Neutron properly based on reported capability.

Neutorn Logger

There is another small Karaf feature, `odl-neutron-logger`, which logs changes of Neutron YANG models. which can be used for debug/audit.

It would also help to understand how to listen the change.

- https://git.opendaylight.org/gerrit/gitweb?p=neutron.git;a=blob;f=neutron-logger/src/main/java/org/opendaylight/neutron/logger/NeutronLogger.java;hb=refs/heads/stable/boron

API Reference Documentation

The OpenStack Neutron API references

- http://developer.openstack.org/api-ref-networking-v2.html
- http://developer.openstack.org/api-ref-networking-v2-ext.html

Neutron Northbound

How to add new API support

OpenStack Neutron is a moving target. It is continuously adding new features as new rest APIs. Here is a basic step to add new API support:

In the Neutron Northbound project:

- Add new YANG model for it under `neutron/model/src/main/yang` and `update neutron.yang`
- Add northbound API for it, and neutron-spi

  - Implement `Neutron<New API>Request.java` and `Neutron<New API>Norhtbound.java` under `neutron/northbound-api/src/main/java/org/opendaylight/neutron/northbound/api/`
  - Implement `INeutron<New API>CRUD.java` and new data structure if any under `neutron/neutron-spi/src/main/java/org/opendaylight/neutron/spi/`
  - update `neutron/neutron-spi/src/main/java/org/opendaylight/neutron/spi/NeutronCRUDInterfaces.java` to wire new CRUD interface
  - Add unit tests, `Neutron<New structure>JAXBTest.java` under `neutron/neutron-spi/src/test/java/org/opendaylight/neutron/spi/`

- update `neutron/northbound-api/src/main/java/org/opendaylight/neutron/northbound/api/NeutronNorthboundRSApplication.java` to wire new northbound api to `RSApplication`
- Add transcriber, `Neutron<New API>Interface.java` under `transcriber/src/main/java/org/opendaylight/neutron/transcriber/`
- update `transcriber/src/main/java/org/opendaylight/neutron/transcriber/NeutronTranscriberProvider.java` to wire a new transcriber
  - Add integration tests `Neutron<New API>Tests.java` under `integration/test/src/test/java/org/opendaylight/neutron/e2etest/`
  - update `integration/test/src/test/java/org/opendaylight/neutron/e2etest/ITNeutronE2E.java` to run a newly added tests.

In OpenStack networking-odl

- Add new driver (or plugin) for new API with tests.

In a southbound Neutron Provider

- implement actual backend to realize those new API by listening related YANG models.

How to write transcriber

For each Neutron data object, there is an `Neutron*Interface` defined within the transcriber artifact that will write that object to the MD-SAL configuration datastore.

All `Neutron*Interface` extend `AbstractNeutronInterface`, in which two methods are defined:

- one takes the Neutron object as input, and will create a data object from it.
- one takes an uuid as input, and will create a data object containing the uuid.

```
protected abstract T toMd(S neutronObject);
protected abstract T toMd(String uuid);
```

In addition the `AbstractNeutronInterface` class provides several other helper methods (`addMd`, `updateMd`, `removeMd`), which handle the actual writing to the configuration datastore.

The semantics of the `toMD()` methods

Each of the Neutron YANG models defines structures containing data. Further each YANG-modeled structure has it own builder. A particular `toMD()` method instantiates an instance of the correct builder, fills in the properties of the builder from the corresponding values of the Neutron object and then creates the YANG-modeled structures via the `build()` method.

As an example, one of the `toMD` code for Neutron Networks is presented below:

```
protected Network toMd(NeutronNetwork network) {
    NetworkBuilder networkBuilder = new NetworkBuilder();
    networkBuilder.setAdminStateUp(network.getAdminStateUp());
    if (network.getNetworkName() != null) {
        networkBuilder.setName(network.getNetworkName());
    }
    if (network.getShared() != null) {
        networkBuilder.setShared(network.getShared());
    }
    if (network.getStatus() != null) {
        networkBuilder.setStatus(network.getStatus());
    }
    if (network.getSubnets() != null) {
        List<Uuid> subnets = new ArrayList<Uuid>();
        for( String subnet : network.getSubnets()) {
            subnets.add(toUuid(subnet));
        }
        networkBuilder.setSubnets(subnets);
    }
    if (network.getTenantID() != null) {
        networkBuilder.setTenantId(toUuid(network.getTenantID()));
    }
    if (network.getNetworkUUID() != null) {
        networkBuilder.setUuid(toUuid(network.getNetworkUUID()));
    } else {
        logger.warn("Attempting to write neutron network without UUID");
    }
    return networkBuilder.build();
}
```

ODL Parent Developer Guide

Parent POMs

Overview

The ODL Parent component for OpenDaylight provides a number of Maven parent POMs which allow Maven projects to be easily integrated in the OpenDaylight ecosystem. Technically, the aim of projects in OpenDaylight is to produce Karaf features, and these parent projects provide common support for the different types of projects involved.

These parent projects are:

- `odlparent-lite` — the basic parent POM for Maven modules which don't produce artifacts (*e.g.* aggregator POMs)
- `odlparent` — the common parent POM for Maven modules containing Java code
- `bundle-parent` — the parent POM for Maven modules producing OSGi bundles

The following parent projects are deprecated, but still used in Carbon:

- `feature-parent` — the parent POM for Maven modules producing Karaf 3 feature repositories
- `karaf-parent` — the parent POM for Maven modules producing Karaf 3 distributions

The following parent projects are new in Carbon, for Karaf 4 support (which won't be complete until Nitrogen):

- `single-feature-parent` — the parent POM for Maven modules producing a single Karaf 4 feature
- `feature-repo-parent` — the parent POM for Maven modules producing Karaf 4 feature repositories
- `karaf4-parent` — the parent POM for Maven modules producing Karaf 4 distributions

odlparent-lite

This is the base parent for all OpenDaylight Maven projects and modules. It provides the following, notably to allow publishing artifacts to Maven Central:

- license information;
- organization information;
- issue management information (a link to our Bugzilla);
- continuous integration information (a link to our Jenkins setup);
- default Maven plugins (`maven-clean-plugin`, `maven-deploy-plugin`, `maven-install-plugin`, `maven-javadoc-plugin` with HelpMojo support, `maven-project-info-reports-plugin`, `maven-site-plugin` with Asciidoc support, `jdepend-maven-plugin`);
- distribution management information.

It also defines two profiles which help during development:

- `q` (`-Pq`), the quick profile, which disables tests, code coverage, Javadoc generation, code analysis, etc. — anything which isn't necessary to build the bundles and features (see this blog post for details);
- `addInstallRepositoryPath` (`-DaddInstallRepositoryPath=.../karaf/system`) which can be used to drop a bundle in the appropriate Karaf location, to enable hot-reloading of bundles during development (see this blog post for details).

For modules which don't produce any useful artifacts (*e.g.* aggregator POMs), you should add the following to avoid processing artifacts:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-deploy-plugin</artifactId>
            <configuration>
                <skip>true</skip>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-install-plugin</artifactId>
            <configuration>
                <skip>true</skip>
            </configuration>
        </plugin>
    </plugins>
</build>
```

odlparent

This inherits from `odlparent-lite` and mainly provides dependency and plugin management for OpenDaylight projects.

If you use any of the following libraries, you should rely on `odlparent` to provide the appropriate versions:

- Akka (and Scala)
- Apache Commons:
  - `commons-codec`
  - `commons-fileupload`
  - `commons-io`
  - `commons-lang`
  - `commons-lang3`
  - `commons-net`
- Apache Shiro
- Guava
- JAX-RS with Jersey
- JSON processing:
  - GSON
  - Jackson
- Logging:
  - Logback
  - SLF4J
- Netty
- OSGi:
  - Apache Felix
  - core OSGi dependencies (`core`, `compendium`...)
- Testing:
  - Hamcrest
  - JSON assert
  - JUnit
  - Mockito
  - Pax Exam
  - PowerMock
- XML/XSL:
  - Xerces
  - XML APIs

Note: This list isn't exhaustive. It's also not cast in stone; if you'd like to add a new dependency (or migrate a dependency), please contact the mailing list.

`odlparent` also enforces some Checkstyle verification rules. In particular, it enforces the common license header used in all OpenDaylight code:

```
/*
 * Copyright © ${year} ${holder} and others.  All rights reserved.
 *
 * This program and the accompanying materials are made available under the
 * terms of the Eclipse Public License v1.0 which accompanies this distribution,
 * and is available at http://www.eclipse.org/legal/epl-v10.html
 */
```

where "`${year}`" is initially the first year of publication, then (after a year has passed) the first and latest years of publication, separated by commas (*e.g.* "2014, 2016"), and "`${holder}`" is the initial copyright holder (typically, the first author's employer). "All rights reserved" is optional.

If you need to disable this license check, *e.g.* for files imported under another license (EPL-compatible of course), you can override the `maven-checkstyle-plugin` configuration. `features-test` does this for its `CustomBundleUrlStreamHandlerFactory` class, which is ASL-licensed:

```
<plugin>
    <artifactId>maven-checkstyle-plugin</artifactId>
    <executions>
        <execution>
            <id>check-license</id>
            <goals>
                <goal>check</goal>
            </goals>
            <phase>process-sources</phase>
            <configuration>
                <configLocation>check-license.xml</configLocation>
                <headerLocation>EPL-LICENSE.regexp.txt</headerLocation>
                <includeResources>false</includeResources>
                <includeTestResources>false</includeTestResources>
                <sourceDirectory>${project.build.sourceDirectory}</sourceDirectory>
                <excludes>
                    <!-- Skip Apache Licensed files -->
                    org/opendaylight/odlparent/featuretest/
↪CustomBundleUrlStreamHandlerFactory.java
                </excludes>
                <failsOnError>false</failsOnError>
                <consoleOutput>true</consoleOutput>
            </configuration>
        </execution>
    </executions>
</plugin>
```

bundle-parent

This inherits from `odlparent` and enables functionality useful for OSGi bundles:

- `maven-javadoc-plugin` is activated, to build the Javadoc JAR;
- `maven-source-plugin` is activated, to build the source JAR;

- `maven-bundle-plugin` is activated (including extensions), to build OSGi bundles (using the "bundle" packaging).

In addition to this, JUnit is included as a default dependency in "test" scope.

features-parent

This inherits from `odlparent` and enables functionality useful for Karaf features:

- `karaf-maven-plugin` is activated, to build Karaf features — but for OpenDaylight, projects need to use "jar" packaging (**not** "feature" or "kar");
- `features.xml` files are processed from templates stored in `src/main/features/features.xml`;
- Karaf features are tested after build to ensure they can be activated in a Karaf container.

The `features.xml` processing allows versions to be ommitted from certain feature dependencies, and replaced with "`{{version}}`". For example:

```
<features name="odl-mdsal-${project.version}" xmlns="http://karaf.apache.org/xmlns/
↪features/v1.2.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://karaf.apache.org/xmlns/features/v1.2.0 http://karaf.
↪apache.org/xmlns/features/v1.2.0">

    <repository>mvn:org.opendaylight.odlparent/features-odlparent/{{VERSION}}/xml/
↪features</repository>

    [...]
    <feature name='odl-mdsal-broker-local' version='${project.version}' description=
↪"OpenDaylight :: MDSAL :: Broker">
        <feature version='${yangtools.version}'>odl-yangtools-common</feature>
        <feature version='${mdsal.version}'>odl-mdsal-binding-dom-adapter</feature>
        <feature version='${mdsal.model.version}'>odl-mdsal-models</feature>
        <feature version='${project.version}'>odl-mdsal-common</feature>
        <feature version='${config.version}'>odl-config-startup</feature>
        <feature version='${config.version}'>odl-config-netty</feature>
        <feature version='[3.3.0,4.0.0)'>odl-lmax</feature>
        [...]
        <bundle>mvn:org.opendaylight.controller/sal-dom-broker-config/{{VERSION}}</
↪bundle>
        <bundle start-level="40">mvn:org.opendaylight.controller/blueprint/{{VERSION}}
↪</bundle>
        <configfile finalname="${config.configfile.directory}/${config.mdsal.
↪configfile}">mvn:org.opendaylight.controller/md-sal-config/{{VERSION}}/xml/config</
↪configfile>
    </feature>
```

As illustrated, versions can be ommitted in this way for repository dependencies, bundle dependencies and configuration files. They must be specified traditionally (either hard-coded, or using Maven properties) for feature dependencies.

karaf-parent

This allows building a Karaf 3 distribution, typically for local testing purposes. Any runtime-scoped feature dependencies will be included in the distribution, and the `karaf.localFeature` property can be used to specify the boot feature (in addition to `standard`).

single-feature-parent

This inherits from `odlparent` and enables functionality useful for Karaf 4 features:

- `karaf-maven-plugin` is activated, to build Karaf features, typically with "feature" packaging ("kar" is also supported);
- `feature.xml` files are generated based on the compile-scope dependencies defined in the POM, optionally initialised from a stub in `src/main/feature/feature.xml`.
- Karaf features are tested after build to ensure they can be activated in a Karaf container.

The `feature.xml` processing adds transitive dependencies by default, which allows features to be defined using only the most significant dependencies (those that define the feature); other requirements are determined automatically as long as they exist as Maven dependencies.

"configfiles" need to be defined both as Maven dependencies (with the appropriate type and classifier) and as `<configfile>` elements in the `feature.xml` stub.

Other features which a feature depends on need to be defined as Maven dependencies with type "xml" and classifier "features" (note the plural here).

feature-repo-parent

This inherits from `odlparent` and enables functionality useful for Karaf 4 feature repositories. It follows the same principles as `single-feature-parent`, but is designed specifically for repositories and should be used only for this type of artifacts.

It builds a feature repository referencing all the (feature) dependencies listed in the POM.

karaf4-parent

This allows building a Karaf 4 distribution, typically for local testing purposes. Any runtime-scoped feature dependencies will be included in the distribution, and the `karaf.localFeature` property can be used to specify the boot feature (in addition to `standard`).

Features (for Karaf 3)

The ODL Parent component for OpenDaylight provides a number of Karaf 3 features which can be used by other Karaf 3 features to use certain third-party upstream dependencies.

These features are:

- Akka features (in the `features-akka` repository):
  - `odl-akka-all` — all Akka bundles;
  - `odl-akka-scala-2.11` — Scala runtime for OpenDaylight;
  - `odl-akka-system-2.4` — Akka actor framework bundles;
  - `odl-akka-clustering-2.4` — Akka clustering bundles and dependencies;
  - `odl-akka-leveldb-0.7` — LevelDB;
  - `odl-akka-persistence-2.4` — Akka persistence;
- general third-party features (in the `features-odlparent` repository):

- odl-netty-4 — all Netty bundles;
- odl-guava-18 — Guava 18;
- odl-guava-21 — Guava 21 (not indended for use in Carbon);
- odl-lmax-3 — LMAX Disruptor;
- odl-triemap-0.2 — Concurrent Trie HashMap.

To use these, you need to declare a dependency on the appropriate repository in your `features.xml` file:

```
<repository>mvn:org.opendaylight.odlparent/features-odlparent/{{VERSION}}/xml/features
↪</repository>
```

and then include the feature, *e.g.*:

```
<feature name='odl-mdsal-broker-local' version='${project.version}' description=
↪"OpenDaylight :: MDSAL :: Broker">
    [...]
    <feature version='[3.3.0,4.0.0)'>odl-lmax</feature>
    [...]
</feature>
```

You also need to depend on the features repository in your POM:

```
<dependency>
    <groupId>org.opendaylight.odlparent</groupId>
    <artifactId>features-odlparent</artifactId>
    <classifier>features</classifier>
    <type>xml</type>
</dependency>
```

assuming the appropriate dependency management:

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.opendaylight.odlparent</groupId>
            <artifactId>odlparent-artifacts</artifactId>
            <version>1.8.0-SNAPSHOT</version>
            <scope>import</scope>
            <type>pom</type>
        </dependency>
    </dependencies>
</dependencyManagement>
```

(the version number there is appropriate for Carbon). For the time being you also need to depend separately on the individual JARs as compile-time dependencies to build your dependent code; the relevant dependencies are managed in `odlparent`'s dependency management.

The suggested version ranges are as follows:

- `odl-netty`: `[4.0.37,4.1.0)` or `[4.0.37,5.0.0)`;
- `odl-guava`: `[18,19)` (if your code is ready for it, `[19,20)` is also available, but the current default version of Guava in OpenDaylight is 18);
- `odl-lmax`: `[3.3.4,4.0.0)`

Features (for Karaf 4)

There are equivalent features to all the Karaf 3 features, for Karaf 4. The repositories use "features4" instead of "features", and the features use "odl4" instead of "odl".

The following new features are specific to Karaf 4:

- Karaf wrapper features (also in the `features4-odlparent` repository) — these can be used to pull in a Karaf feature using a Maven dependency in a POM:
  - `odl-karaf-feat-feature` — the Karaf `feature` feature;
  - `odl-karaf-feat-jdbc` — the Karaf `jdbc` feature;
  - `odl-karaf-feat-jetty` — the Karaf `jetty` feature;
  - `odl-karaf-feat-war` — the Karaf `war` feature.

To use these, all you need to do now is add the appropriate dependency in your feature POM; for example:

```
<dependency>
    <groupId>org.opendaylight.odlparent</groupId>
    <artifactId>odl4-guava-18</artifactId>
    <classifier>features</classifier>
    <type>xml</type>
</dependency>
```

assuming the appropriate dependency management:

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.opendaylight.odlparent</groupId>
            <artifactId>odlparent-artifacts</artifactId>
            <version>1.8.0-SNAPSHOT</version>
            <scope>import</scope>
            <type>pom</type>
        </dependency>
    </dependencies>
</dependencyManagement>
```

(the version number there is appropriate for Carbon). We no longer use version ranges, the feature dependencies all use the `odlparent` version (but you should rely on the artifacts POM).

OCP Plugin Developer Guide

This document is intended for both OCP (ORI [Open Radio Interface] C&M [Control and Management] Protocol) agent developers and OpenDaylight service/application developers. It describes essential information needed to implement an OCP agent that is capable of interoperating with the OCP plugin running in OpenDaylight, including the OCP connection establishment and state machines used on both ends of the connection. It also provides a detailed description of the northbound/southbound APIs that the OCP plugin exposes to allow automation and programmability.

Overview

OCP is an ETSI standard protocol for control and management of Remote Radio Head (RRH) equipment. The OCP Project addresses the need for a southbound plugin that allows applications and controller services to interact with

RRHs using OCP. The OCP southbound plugin will allow applications acting as a Radio Equipment Control (REC) to interact with RRHs that support an OCP agent.

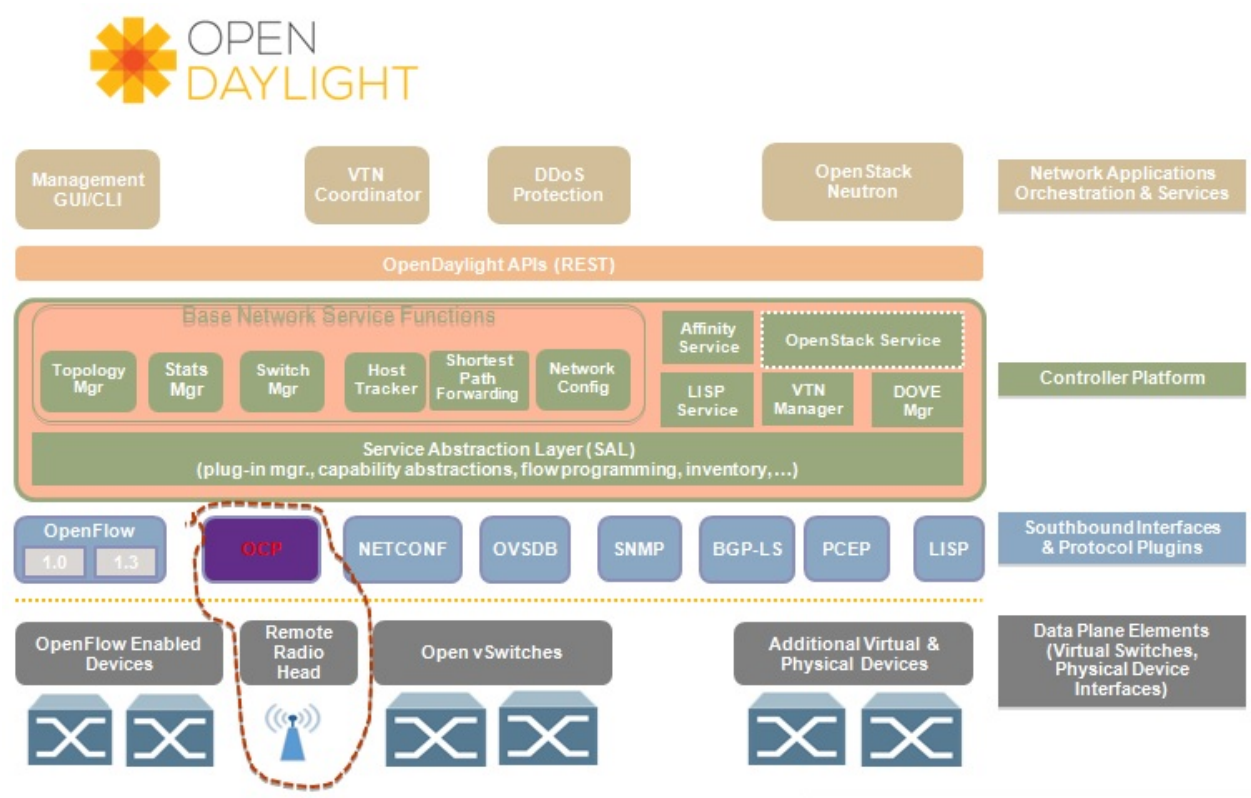


Fig. 2.30: OCP southbound plugin

Architecture

OCP is a vendor-neutral standard communications interface defined to enable control and management between RE and REC of an ORI architecture. The OCP Plugin supports the implementation of the OCP specification; it is based on the Model Driven Service Abstraction Layer (MD-SAL) architecture.

The OCP Plugin project consists of three main components: OCP southbound plugin, OCP protocol library and OCP service. For details on each of them, refer to the OCP Plugin User Guide.

Connection Establishment

The OCP layer is transported over a TCP/IP connection established between the RE and the REC. OCP provides the following functions:

- Control & Management of the RE by the REC
- Transport of AISG/3GPP Iuant Layer 7 messages and alarms between REC and RE

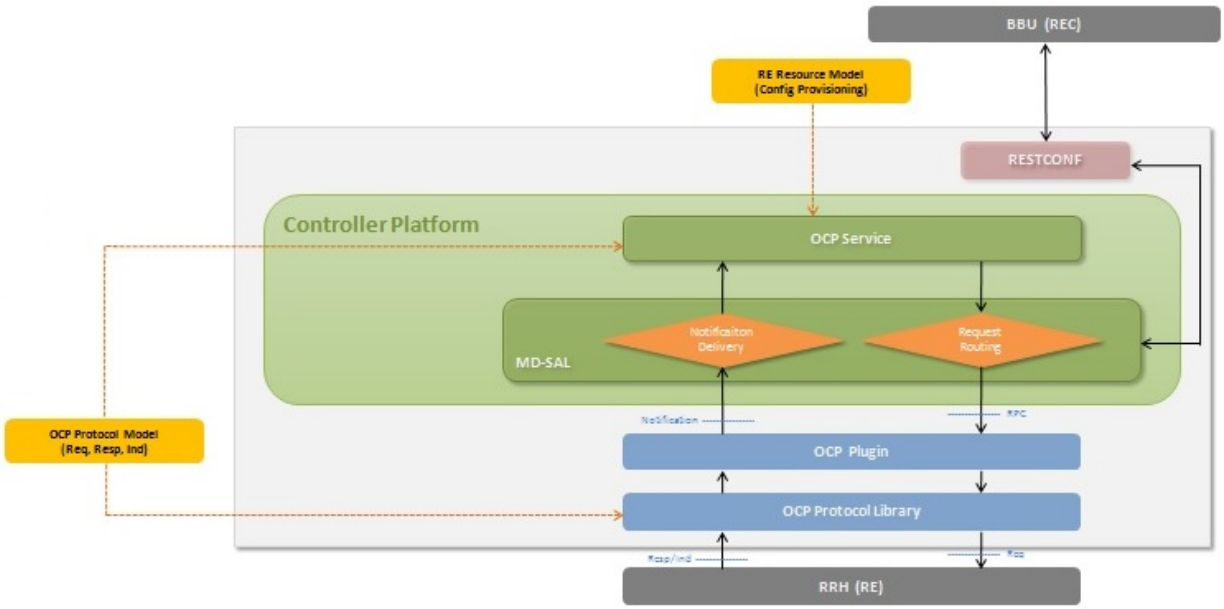


Fig. 2.31: Overall architecture

Hello Message

Hello message is used by the OCP agent during connection setup. It is used for version negotiation. When the connection is established, the OCP agent immediately sends a Hello message with the version field set to highest OCP version supported by itself, along with the verdor ID and serial number of the radio head it is running on.

The combinaiton of the verdor ID and serial number will be used by the OCP plugin to uniquely identify a managed radio head. When not receiving reply from the OCP plugin, the OCP agent can resend Hello message with pre-defined Hello timeout (THLO) and Hello resend times (NHLO).

According to ORI spec, the default value of TCP Link Monitoring Timer (TTLM) is 50 seconds. The RE shall trigger an OCP layer restart while TTLM expires in RE or the RE detects a TCP link failure. So we may define NHLO * THLO = 50 seconds (e.g. NHLO = 10, THLO = 5 seconds).

By nature the Hello message is a new type of indication, and it contains supported OCP version, vendor ID and serial number as shown below.

Hello message.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns="http://uri.etsi.org/ori/002-2/v4.1.1">
  <header>
    <msgType>IND</msgType>
    <msgUID>0</msgUID>
  </header>
  <body>
    <helloInd>
      <version>4.1.1</version>
      <vendorId>XYZ</vendorId>
      <serialNumber>ABC123</serialNumber>
    </helloInd>
  </body>
</msg>
```

Ack Message

Hello from the OCP agent will always make the OCP plugin respond with ACK. In case everything is OK, it will be ACK(OK). In case something is wrong, it will be ACK(FAIL).

If the OCP agent receives ACK(OK), it goes to Established state. If the OCP agent receives ACK(FAIL), it goes to Maintenance state. The failure code and reason of ACK(FAIL) are defined as below:

- FAIL_OCP_VERSION (OCP version not supported)
- FAIL_NO_MORE_CAPACITY (OCP plugin cannot control any more radio heads)

The result inside Ack message indicates OK or FAIL with different reasons.

Ack message.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns="http://uri.etsi.org/ori/002-2/v4.1.1">
  <header>
    <msgType>ACK</msgType>
    <msgUID>0</msgUID>
  </header>
  <body>
    <helloAck>
      <result>FAIL_OCP_VERSION</result>
    </helloAck>
  </body>
</msg>
```

State Machines

The following figures illustrate the Finite State Machine (FSM) of the OCP agent and OCP plugin for new connection procedure.

Northbound APIs

There are ten exposed northbound APIs: health-check, set-time, re-reset, get-param, modify-param, create-obj, delete-obj, get-state, modify-state and get-fault

health-check

The Health Check procedure allows the application to verify that the OCP layer is functioning correctly at the RE.

Default URL: http://localhost:8181/restconf/operations/ocp-service:health-check-nb

POST Input

Field Name	Type	Description	Example	Required ?
nodeId	String	Inventory node reference for OCP radio head	ocp:MTI-101-200	Yes
tcpLinkMonTimeout	unsigned Short	TCP Link Monitoring Timeout (unit: seconds)	50	Yes

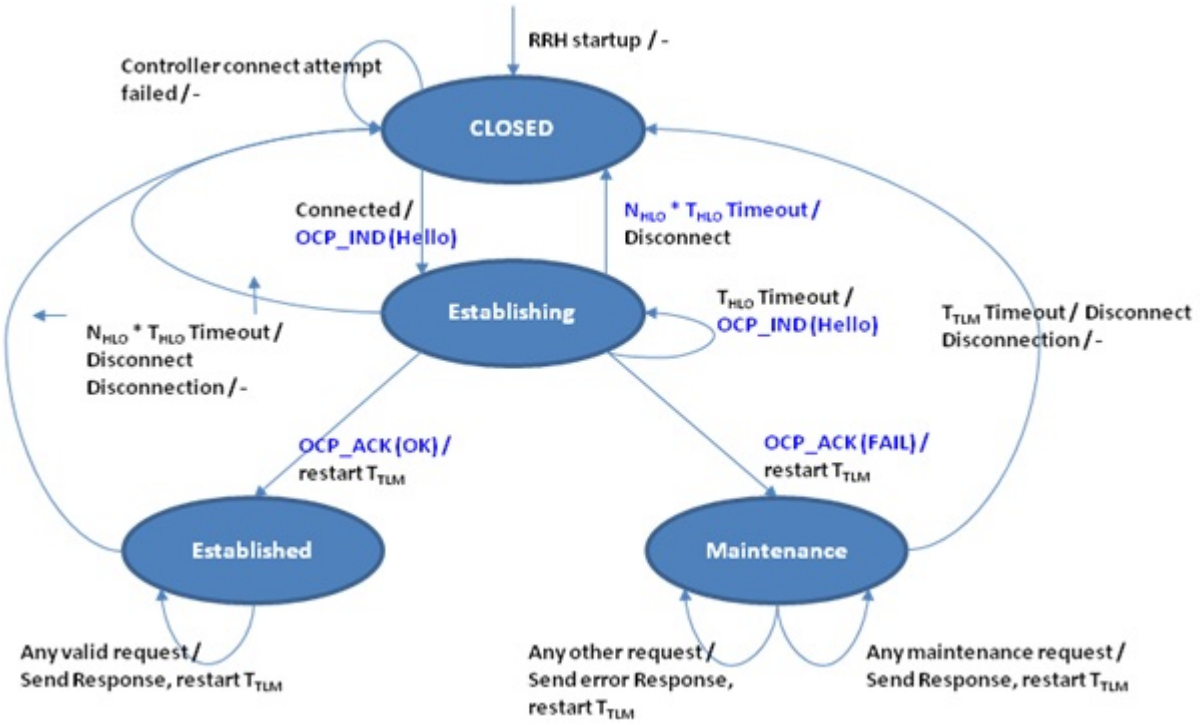


Fig. 2.32: OCP agent state machine

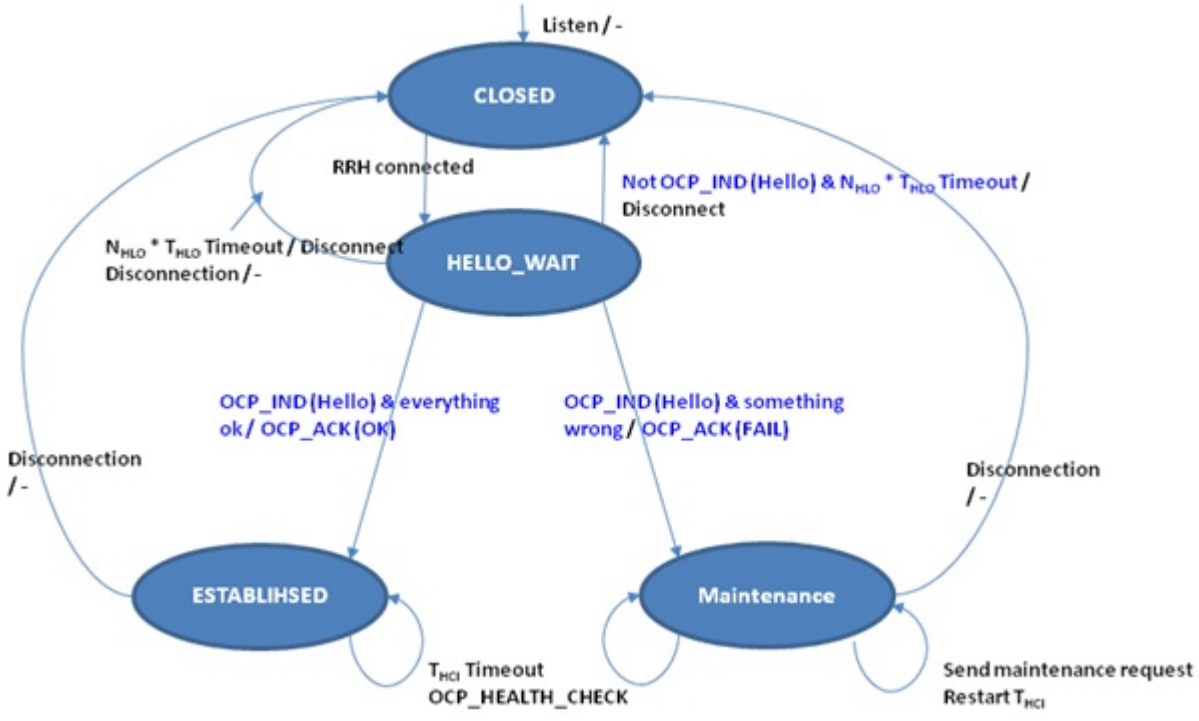


Fig. 2.33: OCP plugin state machine

Example.

```
{
    "health-check-nb": {
        "input": {
            "nodeId": "ocp:MTI-101-200",
            "tcpLinkMonTimeout": "50"
        }
    }
}
```

POST Output

Field Name	Type	Description
result	String, enumerated	Common default result codes

Example.

```
{
    "output": {
        "result": "SUCCESS"
    }
}
```

set-time

The Set Time procedure allows the application to set/update the absolute time reference that shall be used by the RE.

Default URL: http://localhost:8181/restconf/operations/ocp-service:set-time-nb

POST Input

Field Name	Type	Description	Example	Required?
nodeId	String	Inventory node reference for OCP radio head	ocp:MTI-101-200	Yes
newTime	dateTime	New datetime setting for radio head	2016-04-26T10:23:00-05:00	Yes

Example.

```
{
    "set-time-nb": {
        "input": {
            "nodeId": "ocp:MTI-101-200",
            "newTime": "2016-04-26T10:23:00-05:00"
        }
    }
}
```

POST Output

Field Name	Type	Description
result	String, enumerated	Common default result codes + FAIL_INVALID_TIMEDATA

Example.

```
{
    "output": {
        "result": "SUCCESS"
    }
}
```

re-reset

The RE Reset procedure allows the application to reset a specific RE.

Default URL: http://localhost:8181/restconf/operations/ocp-service:re-reset-nb

POST Input

Field Name	Type	Description	Example	Required?
nodeId	String	Inventory node reference for OCP radio head	ocp:MTI-101-200	Yes

Example.

```
{
    "re-reset-nb": {
        "input": {
            "nodeId": "ocp:MTI-101-200"
        }
    }
}
```

POST Output

Field Name	Type	Description
result	String, enumerated	Common default result codes

Example.

```
{
    "output": {
        "result": "SUCCESS"
    }
}
```

get-param

The Object Parameter Reporting procedure allows the application to retrieve the following information:

1. the defined object types and instances within the Resource Model of the RE

2. the values of the parameters of the objects

Default URL: http://localhost:8181/restconf/operations/ocp-service:get-param-nb

POST Input

Field Name	Type	Description	Example	Required?
nodeId	String	Inventory node reference for OCP radio head	ocp:MTI-101-200	Yes
objId	String	Object ID	RxSigPath_5G:1	Yes
paramName	String	Parameter name	dataLink	Yes

Example.

```
{
    "get-param-nb": {
        "input": {
            "nodeId": "ocp:MTI-101-200",
            "objId": "RxSigPath_5G:1",
            "paramName": "dataLink"
        }
    }
}
```

POST Output

Field Name	Type	Description
id	String	Object ID
name	String	Object parameter name
value	String	Object parameter value
result	String, enumerated	Common default result codes + “FAIL_UNKNOWN_OBJECT”, “FAIL_UNKNOWN_PARAM”

Example.

```
{
    "output": {
        "obj": [
            {
                "id": "RxSigPath_5G:1",
                "param": [
                    {
                        "name": "dataLink",
                        "value": "dataLink:1"
                    }
                ]
            }
        ],
        "result": "SUCCESS"
    }
}
```

modify-param

The Object Parameter Modification procedure allows the application to configure the values of the parameters of the objects identified by the Resource Model.

Default URL: http://localhost:8181/restconf/operations/ocp-service:modify-param-nb

POST Input

Field Name	Type	Description	Example	Required?
nodeId	String	Inventory node reference for OCP radio head	ocp:MTI-101-200	Yes
objId	String	Object ID	RxSigPath_5G:1	Yes
name	String	Object parameter name	dataLink	Yes
value	String	Object parameter value	dataLink:1	Yes

Example.

```
{
    "modify-param-nb": {
        "input": {
            "nodeId": "ocp:MTI-101-200",
            "objId": "RxSigPath_5G:1",
            "param": [
                {
                    "name": "dataLink",
                    "value": "dataLink:1"
                }
            ]
        }
    }
}
```

POST Output

Field Name	Type	Description
objId	String	Object ID
globResult	String, enumerated	Common default result codes + “FAIL_UNKNOWN_OBJECT”, “FAIL_PARAMETER_FAIL”, “FAIL_NOSUCH_RESOURCE”
name	String	Object parameter name
result	String, enumerated	“SUCCESS”, “FAIL_UNKNOWN_PARAM”, “FAIL_PARAM_READONLY”, “FAIL_PARAM_LOCKREQUIRED”, “FAIL_VALUE_OUTOF_RANGE”, “FAIL_VALUE_TYPE_ERROR”

Example.

```
{
    "output": {
        "objId": "RxSigPath_5G:1",
        "globResult": "SUCCESS",
        "param": [
            {
                "name": "dataLink",
```

```
                "result": "SUCCESS"
            }
        ]
    }
}
```

create-obj

The Object Creation procedure allows the application to create and initialize a new instance of the given object type on the RE.

Default URL: http://localhost:8181/restconf/operations/ocp-service:create-obj-nb

POST Input

Field Name	Type	Description	Example	Required?
nodeId	String	Inventory node reference for OCP radio head	ocp:MTI-101-200	Yes
objType	String	Object type	RxSigPath_5G	Yes
name	String	Object parameter name	dataLink	No
value	String	Object parameter value	dataLink:1	No

Example.

```
{
    "create-obj-nb": {
        "input": {
            "nodeId": "ocp:MTI-101-200",
            "objType": "RxSigPath_5G",
            "param": [
                {
                    "name": "dataLink",
                    "value": "dataLink:1"
                }
            ]
        }
    }
}
```

POST Output

Field Name	Type	Description
objId	String	Object ID
gloBResult	String, enumerated	Common default result codes + “FAIL_UNKNOWN_OBJTYPE”, “FAIL_STATIC_OBJTYPE”, “FAIL_UNKNOWN_OBJECT”, “FAIL_CHILD_NOTALLOWED”, “FAIL_OUTOF_RESOURCES” “FAIL_PARAMETER_FAIL”, “FAIL_NOSUCH_RESOURCE”
name	String	Object parameter name
result	String, enumerated	“SUCCESS”, “FAIL_UNKNOWN_PARAM”, “FAIL_PARAM_READONLY”, “FAIL_PARAM_LOCKREQUIRED”, “FAIL_VALUE_OUTOF_RANGE”, “FAIL_VALUE_TYPE_ERROR”

Example.

```
{
    "output": {
        "objId": "RxSigPath_5G:0",
        "globResult": "SUCCESS",
        "param": [
            {
                "name": "dataLink",
                "result": "SUCCESS"
            }
        ]
    }
}
```

delete-obj

The Object Deletion procedure allows the application to delete a given object instance and recursively its entire child objects on the RE.

Default URL: http://localhost:8181/restconf/operations/ocp-service:delete-obj-nb

POST Input

Field Name	Type	Description	Example	Required?
nodeId	String	Inventory node reference for OCP radio head	ocp:MTI-101-200	Yes
objId	String	Object ID	RxSigPath_5G:1	Yes

Example.

```
{
    "delete-obj-nb": {
        "input": {
            "nodeId": "ocp:MTI-101-200",
            "obj-id": "RxSigPath_5G:0"
        }
    }
}
```

POST Output

Field Name	Type	Description
result	String, enumerated	Common default result codes + “FAIL_UNKNOWN_OBJECT”, “FAIL_STATIC_OBJTYPE”, “FAIL_LOCKREQUIRED”

Example.

```
{
    "output": {
        "result": "SUCCESS"
    }
}
```

get-state

The Object State Reporting procedure allows the application to acquire the current state (for the requested state type) of one or more objects of the RE resource model, and additionally configure event-triggered reporting of the detected state changes for all state types of the indicated objects.

Default URL: http://localhost:8181/restconf/operations/ocp-service:get-state-nb

POST Input

Field Name	Type	Description	Example	Required ?
nodeId	String	Inventory node reference for OCP radio head	ocp:MTI-101-200	Yes
objId	String	Object ID	RxSig-Path_5G:1	Yes
stateType	String, enumerated	Valid values: “AST”, “FST”, “ALL”	ALL	Yes
eventDrivenReporting	Boolean	Event-triggered reporting of state change	true	Yes

Example.

```
{
    "get-state-nb": {
        "input": {
            "nodeId": "ocp:MTI-101-200",
            "objId": "antPort:0",
            "stateType": "ALL",
            "eventDrivenReporting": "true"
        }
    }
}
```

POST Output

Field Name	Type	Description
id	String	Object ID
type	String, enumerated	State type. Valid values: “AST”, “FST
value	String, enumerated	State value. Valid values: For state type = “AST”: “LOCKED”, “UNLOCKED”. For state type = “FST”: “PRE_OPERATIONAL”, “OPERATIONAL”, “DEGRADED”, “FAILED”, “NOT_OPERATIONAL”, “DISABLED”
result	String, enumerated	Common default result codes + “FAIL_UNKNOWN_OBJECT”, “FAIL_UNKNOWN_STATETYPE”, “FAIL_VALUE_OUTOF_RANGE”

Example.

```
{
    "output": {
```

```
        "obj": [
            {
                "id": "antPort:0",
                "state": [
                    {
                        "type": "FST",
                        "value": "DISABLED"
                    },
                    {
                        "type": "AST",
                        "value": "LOCKED"
                    }
                ]
            }
        ],
        "result": "SUCCESS"
    }
}
```

modify-state

The Object State Modification procedure allows the application to trigger a change in the state of an object of the RE Resource Model.

Default URL: http://localhost:8181/restconf/operations/ocp-service:modify-state-nb

POST Input

Field Name	Type	Description	Example	Required?
nodeId	String	Inventory node reference for OCP radio head	ocp:MTI-101-200	Yes
objId	String	Object ID	RxSigPath_5G:1	Yes
stateType	String, enumerated	Valid values: “AST”, “FST”, “ALL”	AST	Yes
stateValue	String, enumerated	Valid values: For state type = “AST”: “LOCKED”, “UNLOCKED”. For state type = “FST”: “PRE_OPERATIONAL”, “OPERATIONAL”, “DEGRADED”, “FAILED”, “NOT_OPERATIONAL”, “DISABLED”	LOCKED	Yes

Example.

```
{
    "modify-state-nb": {
        "input": {
            "nodeId": "ocp:MTI-101-200",
            "objId": "RxSigPath_5G:1",
            "stateType": "AST",
            "stateValue": "LOCKED"
        }
    }
}
```

POST Output

Field Name	Type	Description
objId	String	Object ID
stateType	String, enumerated	State type. Valid values: “AST”, “FST
stateValue	String, enumerated	State value. Valid values: For state type = “AST”: “LOCKED”, “UNLOCKED”. For state type = “FST”: “PRE_OPERATIONAL”, “OPERATIONAL”, “DEGRADED”, “FAILED”, “NOT_OPERATIONAL”, “DISABLED”
result	String, enumerated	Common default result codes + “FAIL_UNKNOWN_OBJECT”, “FAIL_UNKNOWN_STATETYPE”, “FAIL_UNKNOWN_STATEVALUE”, “FAIL_STATE_READONLY”, “FAIL_RESOURCE_UNAVAILABLE”, “FAIL_RESOURCE_INUSE”, “FAIL_PARENT_CHILD_CONFLICT”, “FAIL_PRECONDITION_NOTMET

Example.

```
{
    "output": {
        "objId": "RxSigPath_5G:1",
        "stateType": "AST",
        "stateValue": "LOCKED",
        "result": "SUCCESS",
    }
}
```

get-fault

The Fault Reporting procedure allows the application to acquire information about all current active faults associated with a primary object, as well as configure the RE to report when the fault status changes for any of faults associated with the indicated primary object.

Default URL: http://localhost:8181/restconf/operations/ocp-service:get-fault-nb

POST Input

Field Name	Type	Description	Example	Required?
nodeId	String	Inventory node reference for OCP radio head	ocp:MTI-101-200	Yes
objId	String	Object ID	RE:0	Yes
eventDrivenReporting	Boolean	Event-triggered reporting of fault	true	Yes

Example.

```
{
    "get-fault-nb": {
        "input": {
            "nodeId": "ocp:MTI-101-200",
            "objId": "RE:0",
```

```
            "eventDrivenReporting": "true"
        }
    }
}
```

POST Output

Field Name	Type	Description
result	String, enumerated	Common default result codes + "FAIL_UNKNOWN_OBJECT", "FAIL_VALUE_OUTOF_RANGE"
id (obj)	String	Object ID
id (fault)	String	Fault ID
severity	String	Fault severity
timestamp	dateTime	Time stamp
descr	String	Text description
affectedObj	String	Affected object

Example.

```
{
    "output": {
        "result": "SUCCESS",
        "obj": [
            {
                "id": "RE:0",
                "fault": [
                    {
                        "id": "FAULT_OVERTEMP",
                        "severity": "DEGRADED",
                        "timestamp": "2012-02-12T16:35:00",
                        "descr": "PA temp too high; Pout reduced",
                        "affectedObj": [
                            "TxSigPath_EUTRA:0",
                            "TxSigPath_EUTRA:1"
                        ]
                    },
                    {
                        "id": "FAULT_VSWR_OUTOF_RANGE",
                        "severity": "WARNING",
                        "timestamp": "2012-02-12T16:01:05",
                    }
                ]
            }
        ],
    }
}
```

Note: The northbound APIs described above wrap the southbound APIs to make them accessible to external applications via RESTCONF, as well as take care of synchronizing the RE resource model between radio heads and the controller's datastore. See applications/ocp-service/src/main/yang/ocp-resourcemodel.yang for the yang representa-

tion of the RE resource model.

Java Interfaces (Southbound APIs)

The southbound APIs provide concrete implementation of the following OCP elementary functions: health-check, set-time, re-reset, get-param, modify-param, create-obj, delete-obj, get-state, modify-state and get-fault. Any OpenDaylight services/applications (of course, including OCP service) wanting to speak OCP to radio heads will need to use them.

SalDeviceMgmtService

Interface SalDeviceMgmtService defines three methods corresponding to health-check, set-time and re-reset.

SalDeviceMgmtService.java.

```
package org.opendaylight.yang.gen.v1.urn.opendaylight.ocp.device.mgmt.rev150811;

public interface SalDeviceMgmtService
    extends
    RpcService
{

    Future<RpcResult<HealthCheckOutput>> healthCheck(HealthCheckInput input);

    Future<RpcResult<SetTimeOutput>> setTime(SetTimeInput input);

    Future<RpcResult<ReResetOutput>> reReset(ReResetInput input);

}
```

SalConfigMgmtService

Interface SalConfigMgmtService defines two methods corresponding to get-param and modify-param.

SalConfigMgmtService.java.

```
package org.opendaylight.yang.gen.v1.urn.opendaylight.ocp.config.mgmt.rev150811;

public interface SalConfigMgmtService
    extends
    RpcService
{

    Future<RpcResult<GetParamOutput>> getParam(GetParamInput input);

    Future<RpcResult<ModifyParamOutput>> modifyParam(ModifyParamInput input);

}
```

SalObjectLifecycleService

Interface SalObjectLifecycleService defines two methods corresponding to create-obj and delete-obj.

SalObjectLifecycleService.java.

```
package org.opendaylight.yang.gen.v1.urn.opendaylight.ocp.object.lifecycle.rev150811;

public interface SalObjectLifecycleService
    extends
    RpcService
{

    Future<RpcResult<CreateObjOutput>> createObj(CreateObjInput input);

    Future<RpcResult<DeleteObjOutput>> deleteObj(DeleteObjInput input);

}
```

SalObjectStateMgmtService

Interface SalObjectStateMgmtService defines two methods corresponding to get-state and modify-state.

SalObjectStateMgmtService.java.

```
package org.opendaylight.yang.gen.v1.urn.opendaylight.ocp.object.state.mgmt.rev150811;

public interface SalObjectStateMgmtService
    extends
    RpcService
{

    Future<RpcResult<GetStateOutput>> getState(GetStateInput input);

    Future<RpcResult<ModifyStateOutput>> modifyState(ModifyStateInput input);

}
```

SalFaultMgmtService

Interface SalFaultMgmtService defines only one method corresponding to get-fault.

SalFaultMgmtService.java.

```
package org.opendaylight.yang.gen.v1.urn.opendaylight.ocp.fault.mgmt.rev150811;

public interface SalFaultMgmtService
    extends
    RpcService
{

    Future<RpcResult<GetFaultOutput>> getFault(GetFaultInput input);

}
```

Notifications

In addition to indication messages, the OCP southbound plugin will translate specific events (e.g., connect, disconnect) coming up from the OCP protocol library into MD-SAL Notification objects and then publish them to the MD-SAL. Also, the OCP service will notify the completion of certain operation via Notification as well.

SalDeviceMgmtListener

An onDeviceConnected Notification will be published to the MD-SAL as soon as a radio head is connected to the controller, and when that radio head is disconnected the OCP southbound plugin will publish an onDeviceDisconnected Notification in response to the disconnect event propagated from the OCP protocol library.

SalDeviceMgmtListener.java.

```
package org.opendaylight.yang.gen.v1.urn.opendaylight.ocp.device.mgmt.rev150811;

public interface SalDeviceMgmtListener
    extends
    NotificationListener
{

    void onDeviceConnected(DeviceConnected notification);

    void onDeviceDisconnected(DeviceDisconnected notification);

}
```

OcpServiceListener

The OCP service will publish an onAlignmentCompleted Notification to the MD-SAL once it has completed the OCP alignment procedure with the radio head.

OcpServiceListener.java.

```
package org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.ocp.
↪applications.ocp.service.rev150811;

public interface OcpServiceListener
    extends
    NotificationListener
{

    void onAlignmentCompleted(AlignmentCompleted notification);

}
```

SalObjectStateMgmtListener

When receiving a state change indication message, the OCP southbound plugin will propagate the indication message to upper layer services/applications by publishing a corresponding onStateChangeInd Notification to the MD-SAL.

SalObjectStateMgmtListener.java.

```
package org.opendaylight.yang.gen.v1.urn.opendaylight.ocp.object.state.mgmt.rev150811;

public interface SalObjectStateMgmtListener
    extends
    NotificationListener
{

    void onStateChangeInd(StateChangeInd notification);

}
```

SalFaultMgmtListener

When receiving a fault indication message, the OCP southbound plugin will propagate the indication message to upper layer services/applications by publishing a corresponding onFaultInd Notification to the MD-SAL.

SalFaultMgmtListener.java.

```
package org.opendaylight.yang.gen.v1.urn.opendaylight.ocp.fault.mgmt.rev150811;

public interface SalFaultMgmtListener
    extends
    NotificationListener
{

    void onFaultInd(FaultInd notification);

}
```

ODL-SDNi Developer Guide

Overview

This project aims at enabling inter-SDN controller communication by developing SDNi (Software Defined Networking interface) as an application (ODL-SDNi App).

ODL-SDNi Architecture

- SDNi Aggregator: Northbound SDNi plugin acts as an aggregator for collecting network information such as topology, stats, host etc. This plugin can be evolving as per needs of network data requested to be shared across federated SDN controllers.
- SDNi API: API view autogenerated and accessible through RESTCONF to fetch the aggregated information from the northbound plugin – SDNi aggregator.The RESTCONF protocol operates on a conceptual datastore defined with the YANG data modeling language.
- SDNi Wrapper: SDNi BGP Wrapper will be responsible for the sharing and collecting information to/from federated controllers.
- SDNi UI:This component displays the SDN controllers connected to each other.

SDNi Aggregator

- SDNiAggregator connects with the Base Network Service Functions of the controller. Currently it is querying network topology through MD-SAL for creating SDNi network capability.
- SDNiAggregator is customized to retrieve the host controller's details, while running the controller in cluster mode. Rest of the northbound APIs of controller will retrieve the entire topology information of all the connected controllers.
- The SDNiAggregator creates a topology structure.This structure is populated by the various network funtions.

SDNi API

Topology and QoS data is fetched from SDNiAggregator through RESTCONF.

http://\protect\T1\textdollar\protect\T1\textbraceleftcontrolleripaddress\protect\T1\textbraceright:8181/apidoc/explorer/index.html

http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/opendaylight-sdni-topology-msg:getAllPeerTopology

Peer Topology Data: Controller IP Address, Links, Nodes, Link Bandwidths, MAC Address of switches, Latency, Host IP address.

http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/opendaylight-sdni-qos-msg:get-all-node-connectors-statistics

QOS Data: Node, Port, Transmit Packets, Receive Packets, Collision Count, Receive Frame Error, Receive Over Run Error, Receive Crc Error

http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/opendaylight-sdni-qos-msg:get-all-peer-node-connectors-statistics

Peer QOS Data: Node, Port, Transmit Packets, Receive Packets, Collision Count, Receive Frame Error, Receive Over Run Error, Receive Crc Error

SDNi Wrapper

- SDNiWrapper is an extension of ODL-BGPCEP where SDNi topology data is exchange along with the Update NLRI message. Refer http://tools.ietf.org/html/draft-ietf-idr-ls-distribution-04 for more information on NLRI.
- SDNiWrapper gets the controller's network capabilities through SDNi Aggregator and serialize it in Update NLRI message. This NLRI message will get exchange between the clustered controllers through BGP-UPDATE message. Similarly peer controller's UPDATE message is received and unpacked then format to SDNi Network capability data, which will be stored for further purpose.

SDNi UI

This component displays the SDN controllers connected to each other.

http://localhost:8181/index.html#/sdniUI/sdnController

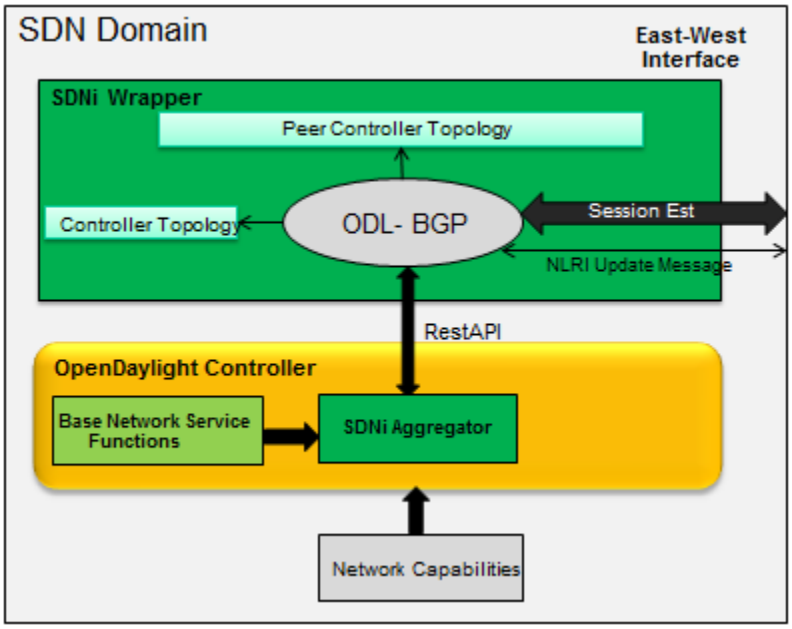


Fig. 2.34: SDNiWrapper

API Reference Documentation

Go to http://\protect\T1\textdollar\protect\T1\textbraceleftcontrolleripaddress\protect\T1\textbraceright:8181/apidoc/explorer/index.html, sign in, and expand the opendaylight-sdni panel. From there, users can execute various API calls to test their SDNi deployment.

OF-CONFIG Developer Guide

Overview

OF-CONFIG defines an OpenFlow switch as an abstraction called an OpenFlow Logical Switch. The OF-CONFIG protocol enables configuration of essential artifacts of an OpenFlow Logical Switch so that an OpenFlow controller can communicate and control the OpenFlow Logical switch via the OpenFlow protocol. OF-CONFIG introduces an operating context for one or more OpenFlow data paths called an OpenFlow Capable Switch for one or more switches. An OpenFlow Capable Switch is intended to be equivalent to an actual physical or virtual network element (e.g. an Ethernet switch) which is hosting one or more OpenFlow data paths by partitioning a set of OpenFlow related resources such as ports and queues among the hosted OpenFlow data paths. The OF-CONFIG protocol enables dynamic association of the OpenFlow related resources of an OpenFlow Capable Switch with specific OpenFlow Logical Switches which are being hosted on the OpenFlow Capable Switch. OF-CONFIG does not specify or report how the partitioning of resources on an OpenFlow Capable Switch is achieved. OF-CONFIG assumes that resources such as ports and queues are partitioned amongst multiple OpenFlow Logical Switches such that each OpenFlow Logical Switch can assume full control over the resources that is assigned to it.

How to start

- start OF-CONFIG feature as below:

```
feature:install odl-of-config-all
```

Compatible with NETCONF

- Config OpenFlow Capable Switch via OpenFlow Configuration Points

  Method: POST

  URI: http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules

  Headers: Content-Type" and "Accept" header attributes set to application/xml

  Payload:

```
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
  <type xmlns:prefix=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
↪prefix:sal-netconf-connector</type>
  <name>testtool</name>
  <address xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">10.74.
↪151.67</address>
  <port xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">830</
↪port>
  <username xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
↪mininet</username>
  <password xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
↪mininet</password>
  <tcp-only xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">false
↪</tcp-only>
  <event-executor xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:netty">
↪prefix:netty-event-executor</type>
    <name>global-event-executor</name>
  </event-executor>
  <binding-registry xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding">prefix:binding-
↪broker-osgi-registry</type>
    <name>binding-osgi-broker</name>
  </binding-registry>
  <dom-registry xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:dom
↪">prefix:dom-broker-osgi-registry</type>
    <name>dom-broker</name>
  </dom-registry>
  <client-dispatcher xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix=
↪"urn:opendaylight:params:xml:ns:yang:controller:config:netconf">prefix:netconf-
↪client-dispatcher</type>
```

```
    <name>global-netconf-dispatcher</name>
  </client-dispatcher>
  <processing-executor xmlns=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:threadpool
↪">prefix:threadpool</type>
    <name>global-netconf-processing-executor</name>
  </processing-executor>
</module>
```

- NETCONF establishes the connections with OpenFlow Capable Switches using the parameters in the previous step. NETCONF also gets the information of whether the OpenFlow Switch supports NETCONF during the signal handshaking. The information will be stored in the NETCONF topology as prosperity of a node.
- OF-CONFIG can be aware of the switches accessing and leaving by monitoring the data changes in the NETCONF topology. For the detailed information it can be refered to the implementation.

The establishment of OF-CONFIG topology

Firstly, OF-CONFIG will check whether the newly accessed switch supports OF-CONFIG by querying the NETCONF interface.

1. During the NETCONF connection's establishment, the NETCONF and the switches will exchange the their capabilities via the "hello" message.
2. OF-CONFIG gets the connection information between the NETCONF and switches by monitoring the data changes via the interface of DataChangeListener.
3. After the NETCONF connection established, the OF-CONFIG module will check whether OF-CONFIG capability is in the switch's capabilities list which is got in step 1.
4. If the result of step 3 is yes, the OF-CONFIG will call the following processing steps to create the topology database.

For the detailed information it can be referred to the implementation.

Secondly, the capable switch node and logical switch node are added in the OF-CONFIG topology if the switch supports OF-CONFIG.

OF-CONFIG's topology compromise: Capable Switch topology (underlay) and logical Switch topology (overlay). Both of them are enhanced (augment) on

/topo:network-topology/topo:topology/topo:node

The NETCONF will add the nodes in the Topology via the path of "/topo:network-topology/topo:topology/topo:node" if it gets the configuration information of the switches.

For the detailed information it can be referred to the implementation.

OpenFlow Protocol Library Developer Guide

Introduction

OpenFlow Protocol Library is component in OpenDaylight, that mediates communication between OpenDaylight controller and hardware devices supporting OpenFlow protocol. Primary goal is to provide user (or upper layers of OpenDaylight) communication channel, that can be used for managing network hardware devices.

Features Overview

There are three features inside openflowjava:

- **odl-openflowjava-protocol** provides all openflowjava bundles, that are needed for communication with openflow devices. It ensures message translation and handles network connections. It also provides openflow protocol specific model.
- **odl-openflowjava-all** currently contains only odl-openflowjava-protocol feature.
- **odl-openflowjava-stats** provides mechanism for message counting and reporting. Can be used for performance analysis.

odl-openflowjava-protocol Architecture

Basic bundles contained in this feature are openflow-protocol-api, openflow-protocol-impl, openflow-protocol-spi and util.

- **openflow-protocol-api** - contains openflow model, constants and keys used for (de)serializer registration.
- **openflow-protocol-impl** - contains message factories, that translate binary messages into DataObjects and vice versa. Bundle also contains network connection handlers - servers, netty pipeline handlers, ...
- **openflow-protocol-spi** - entry point for openflowjava configuration, startup and close. Basically starts implementation.
- **util** - utility classes for binary-Java conversions and to ease experimenter key creation

odl-openflowjava-stats Feature

Runs over odl-openflowjava-protocol. It counts various message types / events and reports counts in specified time periods. Statistics collection can be configured in openflowjava-config/src/main/resources/45-openflowjava-stats.xml

Key APIs and Interfaces

Basic API / SPI classes are ConnectionAdapter (Rpc/notifications) and SwitchConnectionProcider (configure, start, shutdown)

Installation

Pull the code and import project into your IDE.

```
git clone ssh://<username>@git.opendaylight.org:29418/openflowjava.git
```

Configuration

Current implementation allows to configure:

- listening port (mandatory)
- transfer protocol (mandatory)
- switch idle timeout (mandatory)

- TLS configuration (optional)
- thread count (optional)

You can find exemplary Openflow Protocol Library instance configuration below:

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <modules xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
    <!-- default OF-switch-connection-provider (port 6633) -->
    <module>
      <type xmlns:prefix=
↪"urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:provider:impl">
↪prefix:openflow-switch-connection-provider-impl</type>
      <name>openflow-switch-connection-provider-default-impl</name>
      <port>6633</port>
<!--  Possible transport-protocol options: TCP, TLS, UDP -->
      <transport-protocol>TCP</transport-protocol>
      <switch-idle-timeout>15000</switch-idle-timeout>
<!--        Exemplary TLS configuration:
            - uncomment the <tls> tag
            - copy exemplary-switch-privkey.pem, exemplary-switch-cert.pem and␣
↪exemplary-cacert.pem
              files into your virtual machine
            - set VM encryption options to use copied keys
            - start communication
           Please visit OpenflowPlugin or Openflow Protocol Library#Documentation␣
↪wiki pages
           for detailed information regarding TLS -->
<!--       <tls>
             <keystore>/exemplary-ctlKeystore</keystore>
             <keystore-type>JKS</keystore-type>
             <keystore-path-type>CLASSPATH</keystore-path-type>
             <keystore-password>opendaylight</keystore-password>
             <truststore>/exemplary-ctlTrustStore</truststore>
             <truststore-type>JKS</truststore-type>
             <truststore-path-type>CLASSPATH</truststore-path-type>
             <truststore-password>opendaylight</truststore-password>
             <certificate-password>opendaylight</certificate-password>
           </tls> -->
<!--       Exemplary thread model configuration. Uncomment <threads> tag below to␣
↪adjust default thread model -->
<!--       <threads>
             <boss-threads>2</boss-threads>
             <worker-threads>8</worker-threads>
           </threads> -->
    </module>
```

```
    <!-- default OF-switch-connection-provider (port 6653) -->
    <module>
      <type xmlns:prefix=
↪"urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:provider:impl">
↪prefix:openflow-switch-connection-provider-impl</type>
      <name>openflow-switch-connection-provider-legacy-impl</name>
      <port>6653</port>
<!--  Possible transport-protocol options: TCP, TLS, UDP -->
      <transport-protocol>TCP</transport-protocol>
      <switch-idle-timeout>15000</switch-idle-timeout>
<!--        Exemplary TLS configuration:
            - uncomment the <tls> tag
```

```
            - copy exemplary-switch-privkey.pem, exemplary-switch-cert.pem and␣
↪exemplary-cacert.pem
              files into your virtual machine
            - set VM encryption options to use copied keys
            - start communication
          Please visit OpenflowPlugin or Openflow Protocol Library#Documentation␣
↪wiki pages
          for detailed information regarding TLS -->
<!--      <tls>
            <keystore>/exemplary-ctlKeystore</keystore>
            <keystore-type>JKS</keystore-type>
            <keystore-path-type>CLASSPATH</keystore-path-type>
            <keystore-password>opendaylight</keystore-password>
            <truststore>/exemplary-ctlTrustStore</truststore>
            <truststore-type>JKS</truststore-type>
            <truststore-path-type>CLASSPATH</truststore-path-type>
            <truststore-password>opendaylight</truststore-password>
            <certificate-password>opendaylight</certificate-password>
          </tls> -->
<!--      Exemplary thread model configuration. Uncomment <threads> tag below to␣
↪adjust default thread model -->
<!--      <threads>
            <boss-threads>2</boss-threads>
            <worker-threads>8</worker-threads>
          </threads> -->
    </module>
```

```
  <module>
    <type xmlns:prefix=
↪"urn:opendaylight:params:xml:ns:yang:openflow:common:config:impl">prefix:openflow-
↪provider-impl</type>
    <name>openflow-provider-impl</name>
    <openflow-switch-connection-provider>
      <type xmlns:ofSwitch=
↪"urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:provider">
↪ofSwitch:openflow-switch-connection-provider</type>
      <name>openflow-switch-connection-provider-default</name>
    </openflow-switch-connection-provider>
    <openflow-switch-connection-provider>
      <type xmlns:ofSwitch=
↪"urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:provider">
↪ofSwitch:openflow-switch-connection-provider</type>
      <name>openflow-switch-connection-provider-legacy</name>
    </openflow-switch-connection-provider>
    <binding-aware-broker>
      <type xmlns:binding=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding">binding:binding-
↪broker-osgi-registry</type>
      <name>binding-osgi-broker</name>
    </binding-aware-broker>
  </module>
</modules>
```

Possible transport-protocol options:

- TCP
- TLS

- UDP

Switch-idle timeout specifies time needed to detect idle state of switch. When no message is received from switch within this time, upper layers are notified on switch idleness. To be able to use this exemplary TLS configuration:

- uncomment the `<tls>` tag
- copy *exemplary-switch-privkey.pem*, *exemplary-switch-cert.pem* and *exemplary-cacert.pem* files into your virtual machine
- set VM encryption options to use copied keys (please visit TLS support wiki page for detailed information regarding TLS)
- start communication

Thread model configuration specifies how many threads are desired to perform Netty's I/O operations.

- boss-threads specifies the number of threads that register incoming connections
- worker-threads specifies the number of threads performing read / write (+ serialization / deserialization) operations.

Architecture

Public API `(openflow-protocol-api)`

Set of interfaces and builders for immutable data transfer objects representing Openflow Protocol structures.

Transfer objects and service APIs are infered from several YANG models using code generator to reduce verbosity of definition and repeatibility of code.

The following YANG modules are defined:

- openflow-types - defines common Openflow specific types
- openflow-instruction - defines base Openflow instructions
- openflow-action - defines base Openflow actions
- openflow-augments - defines object augmentations
- openflow-extensible-match - defines Openflow OXM match
- openflow-protocol - defines Openflow Protocol messages
- system-notifications - defines system notification objects
- openflow-configuration - defines structures used in ConfigSubsystem

This modules also reuse types from following YANG modules:

- ietf-inet-types - IP adresses, IP prefixes, IP-protocol related types
- ietf-yang-types - Mac Address, etc.

The use of predefined types is to make APIs contracts more safe, better readable and documented (e.g using MacAddress instead of byte array...)

TCP Channel pipeline `(openflow-protocol-impl)`

Creates channel processing pipeline based on configuration and support.

TCP Channel pipeline.

imageopenflowjava/500px-TCPChannelPipeline.png[width=500]

Switch Connection Provider.

Implementation of connection point for other projects. Library exposes its functionality through this class. Library can be configured, started and shutdowned here. There are also methods for custom (de)serializer registration.

Tcp Connection Initializer.

In order to initialize TCP connection to a device (switch), OF Plugin calls method `initiateConnection()` in `SwitchConnectionProvider`. This method in turn initializes (Bootstrap) server side channel towards the device.

TCP Handler.

Represents single server that is handling incoming connections over TCP / TLS protocol. TCP Handler creates a single instance of TCP Channel Initializer that will initialize channels. After that it binds to configured InetAddress and port. When a new device connects, TCP Handler registers its channel and passes control to TCP Channel Initializer.

TCP Channel Initializer.

This class is used for channel initialization / rejection and passing arguments. After a new channel has been registered it calls Switch Connection Handler's (OF Plugin) accept method to decide if the library should keep the newly registered channel or if the channel should be closed. If the channel has been accepted, TCP Channel Initializer creates the whole pipeline with needed handlers and also with ConnectionAdapter instance. After the channel pipeline is ready, Switch Connection Handler is notified with `onConnectionReady` notification. OpenFlow Plugin can now start sending messages downstream.

Idle Handler.

If there are no messages received for more than time specified, this handler triggers idle state notification. The switch idle timeout is received as a parameter from ConnectionConfiguration settings. Idle State Handler is inactive while there are messages received within the switch idle timeout. If there are no messages received for more than timeout specified, handler creates SwitchIdleEvent message and sends it upstream.

TLS Handler.

It encrypts and decrypts messages over TLS protocol. Engaging TLS Handler into pipeline is matter of configuration (`<tls>` tag). TLS communication is either unsupported or required. TLS Handler is represented as a Netty's SslHandler.

OF Frame Decoder.

Parses input stream into correct length message frames for further processing. Framing is based on Openflow header length. If received message is shorter than minimal length of OpenFlow message (8 bytes), OF Frame Decoder waits for more data. After receiving at least 8 bytes the decoder checks length in OpenFlow header. If there are still some bytes missing, the decoder waits for them. Else the OF Frame Decoder sends correct length message to next handler in the channel pipeline.

OF Version Detector.

Detects version of used OpenFlow Protocol and discards unsupported version messages. If the detected version is supported, OF Version Detector creates `VersionMessageWrapper` object containing the detected version and byte message and sends this object upstream.

OF Decoder.

Chooses correct deserilization factory (based on message type) and deserializes messages into generated DTOs (Data Transfer Object). OF Decoder receives `VersionMessageWrapper` object and passes it to `DeserializationFactory` which will return translated DTO. `DeserializationFactory` creates `MessageCodeKey` object with version and type of received message and Class of object that will be the received message deserialized into. This object is used as key when searching for appropriate decoder in `DecoderTable`. `DecoderTable` is basically a map storing decoders. Found decoder translates received message into DTO. If there

was no decoder found, null is returned. After returning translated DTO back to OF Decoder, the decoder checks if it is null or not. When the DTO is null, the decoder logs this state and throws an Exception. Else it passes the DTO further upstream. Finally, the OF Decoder releases ByteBuf containing received and decoded byte message.

OF Encoder.

Chooses correct serialization factory (based on type of DTO) and serializes DTOs into byte messages. OF Encoder does the opposite than the OF Decoder using the same principle. OF Encoder receives DTO, passes it for translation and if the result is not null, it sends translated DTO downstream as a ByteBuf. Searching for appropriate encoder is done via MessageTypeKey, based on version and class of received DTO.

Delegating Inbound Handler.

Delegates received DTOs to Connection Adapter. It also reacts on channelInactive and channelUnregistered events. Upon one of these events is triggered, DelegatingInboundHandler creates DisconnectEvent message and sends it upstream, notifying upper layers about switch disconnection.

Channel Outbound Queue.

Message flushing handler. Stores outgoing messages (DTOs) and flushes them. Flush is performed based on time expired and on the number of messages enqueued.

Connection Adapter.

Provides a facade on top of pipeline, which hides netty.io specifics. Provides a set of methods to register for incoming messages and to send messages to particular channel / session. ConnectionAdapterImpl basically implements three interfaces (unified in one superinterface ConnectionFacade):

- ConnectionAdapter
- MessageConsumer
- OpenflowProtocolService

ConnectionAdapter interface has methods for setting up listeners (message, system and connection ready listener), method to check if all listeners are set, checking if the channel is alive and disconnect method. Disconnect method clears responseCache and disables consuming of new messages.

MessageConsumer interface holds only one method: `consume()`. `Consume()` method is called from DelegatingInboundHandler. This method processes received DTO's based on their type. There are three types of received objects:

- System notifications - invoke system notifications in OpenFlow Plugin (systemListener set). In case of `DisconnectEvent` message, the Connection Adapter clears response cache and disables consume() method processing,
- OpenFlow asynchronous messages (from switch) - invoke corresponding notifications in OpenFlow Plugin,
- OpenFlow symmetric messages (replies to requests) - create `RpcResponseKey` with XID and DTO's class set. This `RpcResponseKey` is then used to find corresponding future object in responseCache. Future object is set with success flag, received message and errors (if any occurred). In case no corresponding future was found in responseCache, Connection Adapter logs warning and discards the message. Connection Adapter also logs warning when an unknown DTO is received.

OpenflowProtocolService interface contains all rpc-methods for sending messages from upper layers (OpenFlow Plugin) downstream and responding. Request messages return Future filled with expected reply message, otherwise the expected Future is of type Void.

NOTE: MultipartRequest message is the only exception. Basically it is request - reply Message type, but it wouldn't be able to process more following MultipartReply messages if this was implemented as rpc (only one Future). This is why MultipartReply is implemented as notification. OpenFlow Plugin takes care of correct message processing.

UDP Channel pipeline (openflow-protocol-impl)

Creates UDP channel processing pipeline based on configuration and support. **Switch Connection Provider**, **Channel Outbound Queue** and **Connection Adapter** fulfill the same role as in case of TCP connection / channel pipeline (please see above).

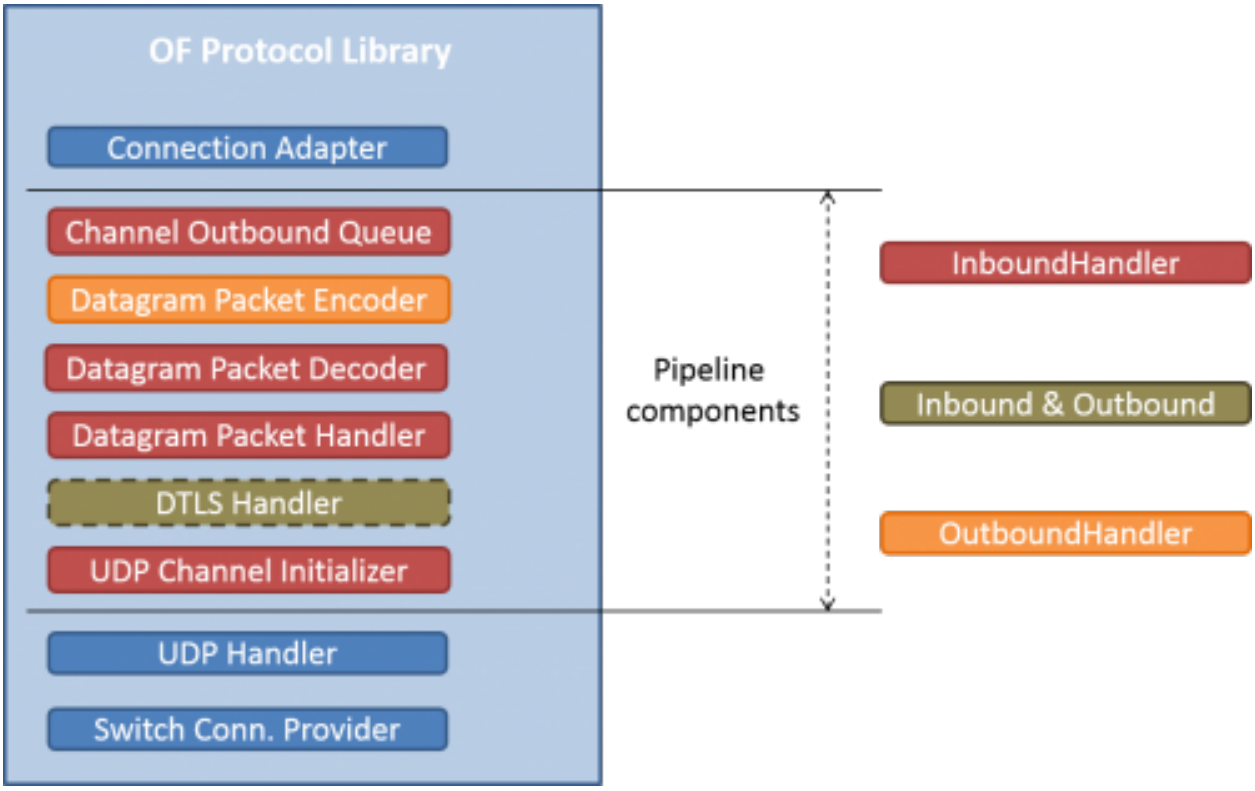


Fig. 2.35: UDP Channel pipeline

UDP Handler.

Represents single server that is handling incoming connections over UDP (DTLS) protocol. UDP Handler creates a single instance of UDP Channel Initializer that will initialize channels. After that it binds to configured InetAddress and port. When a new device connects, UDP Handler registers its channel and passes control to UDP Channel Initializer.

UDP Channel Initializer.

This class is used for channel initialization and passing arguments. After a new channel has been registered (for UDP there is always only one channel) UDP Channel Initializer creates whole pipeline with needed handlers.

DTLS Handler.

Haven't been implemented yet. Will take care of secure DTLS connections.

OF Datagram Packet Handler.

Combines functionality of OF Frame Decoder and OF Version Detector. Extracts messages from received datagram packets and checks if message version is supported. If there is a message received from yet unknown sender, OF Datagram Packet Handler creates Connection Adapter for this sender and stores it under sender's address in `UdpConnectionMap`. This map is also used for sending the messages and for correct Connection Adapter lookup - to delegate messages from one channel to multiple sessions.

OF Datagram Packet Decoder.

Chooses correct deserilization factory (based on message type) and deserializes messages into generated DTOs. OF Decoder receives `VersionMessageUdpWrapper` object and passes it to `DeserializationFactory` which will return translated DTO. `DeserializationFactory` creates `MessageCodeKey` object with version and type of received message and Class of object that will be the received message deserialized into. This object is used as key when searching for appropriate decoder in `DecoderTable`. `DecoderTable` is basically a map storing decoders. Found decoder translates received message into DTO (DataTransferObject). If there was no decoder found, null is returned. After returning translated DTO back to OF Datagram Packet Decoder, the decoder checks if it is null or not. When the DTO is null, the decoder logs this state. Else it looks up appropriate Connection Adapter in `UdpConnectionMap` and passes the DTO to found Connection Adapter. Finally, the OF Decoder releases `ByteBuf` containing received and decoded byte message.

OF Datagram Packet Encoder.

Chooses correct serialization factory (based on type of DTO) and serializes DTOs into byte messages. OF Datagram Packet Encoder does the opposite than the OF Datagram Packet Decoder using the same principle. OF Encoder receives DTO, passes it for translation and if the result is not null, it sends translated DTO downstream as a datagram packet. Searching for appropriate encoder is done via MessageTypeKey, based on version and class of received DTO.

SPI (openflow-protocol-spi)

Defines interface for library's connection point for other projects. Library exposes its functionality through this interface.

Integration test (openflow-protocol-it)

Testing communication with simple client.

Simple client(simple-client)

Lightweight switch simulator - programmable with desired scenarios.

Utility (util)

Contains utility classes, mainly for work with ByteBuf.

Library's lifecycle

Steps (after the library's bundle is started):

- [1] Library is configured by ConfigSubsystem (adress, ports, encryption, . . .)
- [2] Plugin injects its SwitchConnectionHandler into the Library
- [3] Plugin starts the Library
- [4] Library creates configured protocol handler (e.g. TCP Handler)
- [5] Protocol Handler creates Channel Initializer
- [6] Channel Initializer asks plugin whether to accept incoming connection on each new switch connection
- [7] Plugin responds:

  - true - continue building pipeline
  - false - reject connection / disconnect channel
- [8] Library notifies Plugin with onSwitchConnected(ConnectionAdapter) notification, passing reference to ConnectionAdapter, that will handle the connection
- [9] Plugin registers its system and message listeners
- [10] FireConnectionReadyNotification() is triggered, announcing that pipeline handlers needed for communication have been created and Plugin can start communication
- [11] Plugin shutdowns the Library when desired

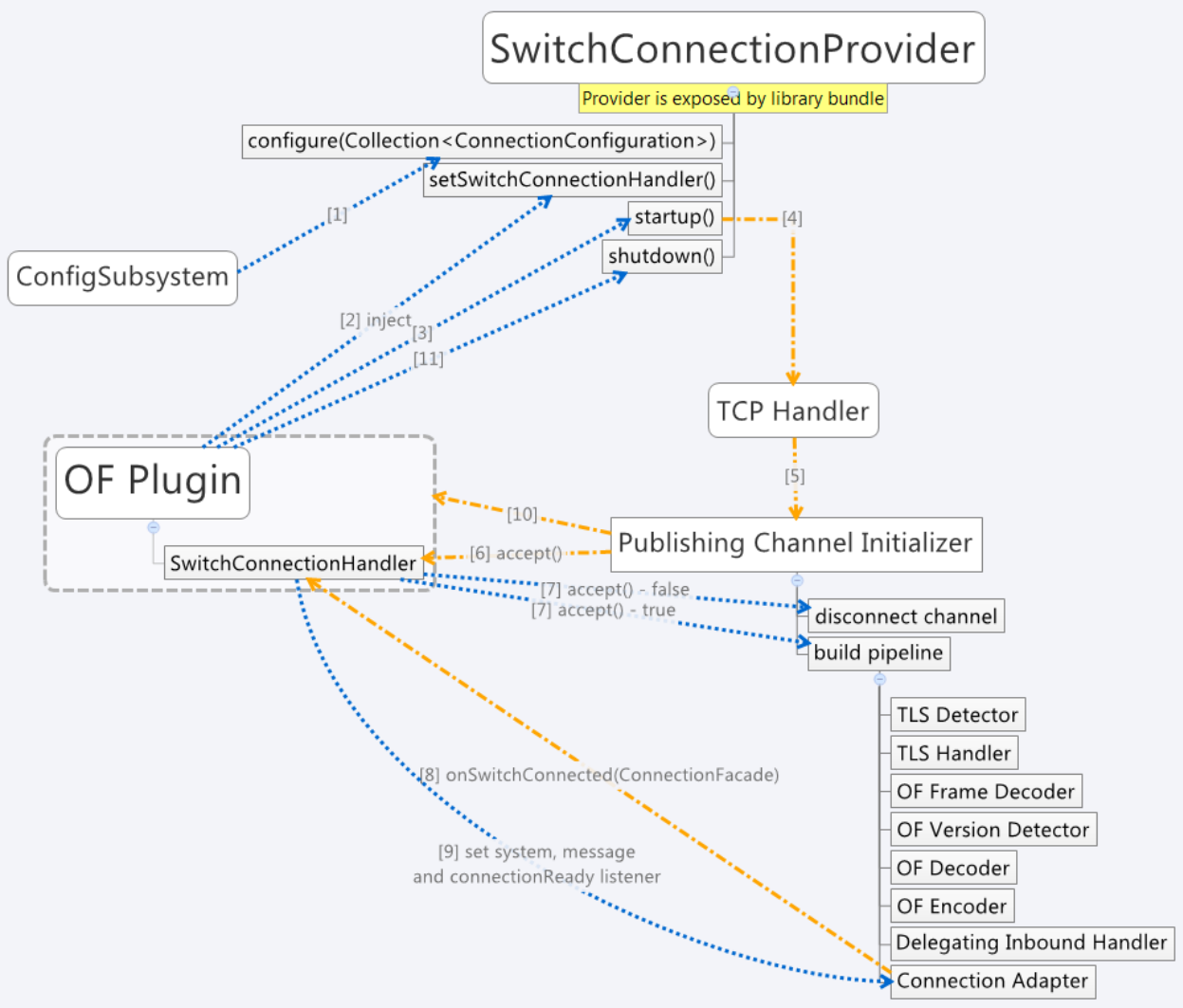


Fig. 2.36: Library lifecycle

Statistics collection

Introduction

Statistics collection collects message statistics. Current collected statistics (`DS` - downstream, `US` - upstream):

- DS_ENTERED_OFJAVA - all messages that entered openflowjava (picked up from openflowplugin)
- DS_ENCODE_SUCCESS - successfully encoded messages
- DS_ENCODE_FAIL - messages that failed during encoding (serialization) process
- DS_FLOW_MODS_ENTERED - all flow-mod messages that entered openflowjava
- DS_FLOW_MODS_SENT - all flow-mod messages that were successfully sent
- US_RECEIVED_IN_OFJAVA - messages received from switch
- US_DECODE_SUCCESS - successfully decoded messages
- US_DECODE_FAIL - messages that failed during decoding (deserialization) process
- US_MESSAGE_PASS - messages handed over to openflowplugin

Karaf

In orded to start statistics, it is needed to feature:install odl-openflowjava-stats. To see the logs one should use log:set DEBUG org.opendaylight.openflowjava.statistics and than probably log:display (you can log:list to see if the logging has been set). To adjust collection settings it is enough to modify 45-openflowjava-stats.xml.

JConsole

JConsole provides two commands for the statistics collection:

- printing current statistics
- resetting statistic counters

After attaching JConsole to correct process, one only needs to go into MBeans `tab` → `org.opendaylight.controller` → `RuntimeBean` → `statistics-collection-service-impl` → `statistics-collection-service-impl` → `Operations` to be able to use this commands.

TLS Support

Note: see OpenFlow Plugin Developper Guide

Extensibility

Introduction

Entry point for the extensibility is `SwitchConnectionProvider`. `SwitchConnectionProvider` contains methods for (de)serializer registration. To register deserializer it is needed to use .register*Deserializer(key, impl). To register serializer one must use .register*Serializer(key, impl). Registration can occur either during configuration or at runtime.

NOTE: In case when experimenter message is received and no (de)serializer was registered, the library will throw `IllegalArgumentException`.

Basic Principle

In order to use extensions it is needed to augment existing model and register new (de)serializers.

Augmenting the model: 1. Create new augmentation

Register (de)serializers: 1. Create your (de)serializer 2. Let it implement `OFDeserializer<>` / `OFSerializer<>` - in case the structure you are (de)serializing needs to be used in Multipart TableFeatures messages, let it implement `HeaderDeserializer<>` / `HeaderSerializer` 3. Implement prescribed methods 4. Register your deserializer under appropriate key (in our case `ExperimenterActionDeserializerKey`) 5. Register your serializer under appropriate key (in our case `ExperimenterActionSerializerKey`) 6. Done, test your implementation

NOTE: If you don't know what key should be used with your (de)serializer implementation, please visit *Registration keys* page.

Example

Let's say we have vendor / experimenter action represented by this structure:

```
struct foo_action {
    uint16_t type;
    uint16_t length;
    uint32_t experimenter;
    uint16_t first;
    uint16_t second;
    uint8_t  pad[4];
}
```

First, we have to augment existing model. We create new module, which imports "`openflow-types.yang`" (don't forget to update your `pom.xml` with api dependency). Now we create foo action identity:

```
import openflow-types {prefix oft;}
identity foo {
    description "Foo action description";
    base oft:action-base;
}
```

This will be used as type in our structure. Now we must augment existing action structure, so that we will have the desired fields first and second. In order to create new augmentation, our module has to import "`openflow-action.yang`". The augment should look like this:

```
import openflow-action {prefix ofaction;}
augment "/ofaction:actions-container/ofaction:action" {
    ext:augment-identifier "foo-action";
        leaf first {
            type uint16;
        }
        leaf second {
            type uint16;
        }
    }
```

We are finished with model changes. Run mvn clean compile to generate sources. After generation is done, we need to implement our (de)serializer.

Deserializer:

```
public class FooActionDeserializer extends OFDeserializer<Action> {
   @Override
   public Action deserialize(ByteBuf input) {
       ActionBuilder builder = new ActionBuilder();
       input.skipBytes(SIZE_OF_SHORT_IN_BYTES); *// we know the type of action*
       builder.setType(Foo.class);
       input.skipBytes(SIZE_OF_SHORT_IN_BYTES); *// we don't need length*
       *// now create experimenterIdAugmentation - so that openflowplugin can
       differentiate correct vendor codec*
       ExperimenterIdActionBuilder expIdBuilder = new ExperimenterIdActionBuilder();
       expIdBuilder.setExperimenter(new ExperimenterId(input.readUnsignedInt()));
       builder.addAugmentation(ExperimenterIdAction.class, expIdBuilder.build());
       FooActionBuilder fooBuilder = new FooActionBuilder();
       fooBuilder.setFirst(input.readUnsignedShort());
       fooBuilder.setSecond(input.readUnsignedShort());
       builder.addAugmentation(FooAction.class, fooBuilder.build());
       input.skipBytes(4); *// padding*
       return builder.build();
   }
}
```

Serializer:

```
public class FooActionSerializer extends OFSerializer<Action> {
   @Override
   public void serialize(Action action, ByteBuf outBuffer) {
       outBuffer.writeShort(FOO_CODE);
       outBuffer.writeShort(16);
       *// we don't have to check for ExperimenterIdAction augmentation - our
       serializer*
       *// was called based on the vendor / experimenter ID, so we simply write
       it to buffer*
       outBuffer.writeInt(VENDOR / EXPERIMENTER ID);
       FooAction foo = action.getAugmentation(FooAction.class);
       outBuffer.writeShort(foo.getFirst());
       outBuffer.writeShort(foo.getSecond());
       outBuffer.writeZero(4); //write padding
   }
}
```

Register both deserializer and serializer: `SwitchConnectionProvider.registerDeserializer(new ExperimenterActionDeserializerKey(0x04, VENDOR / EXPERIMENTER ID), new FooActionDeserializer());` `SwitchConnectionProvider.registerSerializer(new ExperimenterActionSerializerKey(0x04, VENDOR / EXPERIMENTER ID), new FooActionSerializer());`

We are ready to test our implementation.

NOTE: Vendor / Experimenter structures define only vendor / experimenter ID as common distinguisher (besides action type). Vendor / Experimenter ID is unique for all vendor messages - that's why vendor is able to register only one class under ExperimenterAction(De)SerializerKey. And that's why vendor has to switch / choose between his subclasses / subtypes on his own.

Detailed walkthrough: Deserialization extensibility

External interface & class description.

OFGeneralDeserializer:

- `OFDeserializer<E extends DataObject>`
  - *deserialize(ByteBuf)* - deserializes given ByteBuf
- `HeaderDeserializer<E extends DataObject>`
  - *deserializeHeaders(ByteBuf)* - deserializes only E headers (used in Multipart TableFeatures messages)

DeserializerRegistryInjector

- `injectDeserializerRegistry(DeserializerRegistry)` - injects deserializer registry into deserializer. Useful when custom deserializer needs access to other deserializers.

NOTE: DeserializerRegistryInjector is not OFGeneralDeserializer descendand. It is a standalone interface.

MessageCodeKey and its descendants These keys are used as for deserializer lookup in DeserializerRegistry. MessageCodeKey should is used in general, while its descendants are used in more special cases. For Example ActionDeserializerKey is used for Action deserializer lookup and (de)registration. Vendor is provided with special keys, which contain only the most necessary fields. These keys usually start with "Experimenter" prefix (MatchEntryDeserializerKey is an exception).

MessageCodeKey has these fields:

- short version - Openflow wire version number
- int value - value read from byte message
- Class<?> clazz - class of object being creating
- [1] The scenario starts in a custom bundle which wants to extend library's functionality. The custom bundle creates deserializers which implement exposed `OFDeserializer` / `HeaderDeserializer` interfaces (wrapped under `OFGeneralDeserializer` unifying super interface).
- [2] Created deserializers are paired with corresponding ExperimenterKeys, which are used for deserializer lookup. If you don't know what key should be used with your (de)serializer implementation, please visit *Registration keys* page.
- [3] Paired deserializers are passed to the OF Library via **SwitchConnectionProvider**.*registerCustomDeserializer(key, impl)*. Library registers the deserializer.
  - While registering, Library checks if the deserializer is an instance of **DeserializerRegistryInjector** interface. If yes, the DeserializerRegistry (which stores all deserializer references) is injected into the deserializer.

This is particularly useful when the deserializer needs access to other deserializers. For example `IntructionsDeserializer` needs access to `ActionsDeserializer` in order to be able to process OFPIT_WRITE_ACTIONS/OFPIT_APPLY_ACTIONS instructions.

Detailed walkthrough: Serialization extensibility

External interface & class description.

OFGeneralSerializer:

- OFSerializer<E extends DataObject>
  - *serialize(E,ByteBuf)* - serializes E into given ByteBuf
- `HeaderSerializer<E extends DataObject>`
  - *serializeHeaders(E,ByteBuf)* - serializes E headers (used in Multipart TableFeatures messages)

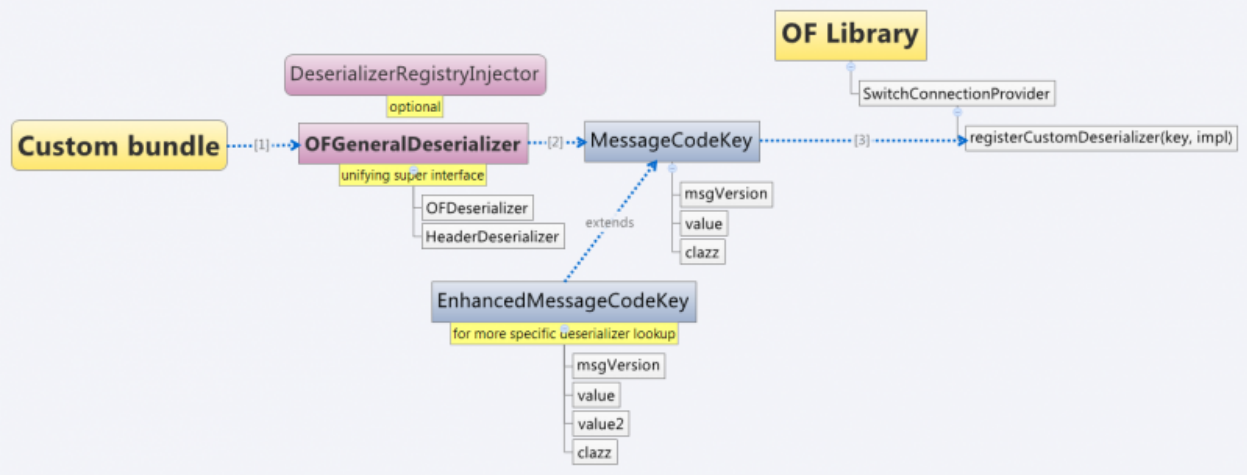


Fig. 2.37: Deserialization scenario walkthrough

SerializerRegistryInjector * `injectSerializerRegistry(SerializerRegistry)` - injects serializer registry into serializer. Useful when custom serializer needs access to other serializers.

NOTE: SerializerRegistryInjector is not OFGeneralSerializer descendand.

MessageTypeKey and its descendants These keys are used as for serializer lookup in SerializerRegistry. MessageTypeKey should is used in general, while its descendants are used in more special cases. For Example ActionSerializerKey is used for Action serializer lookup and (de)registration. Vendor is provided with special keys, which contain only the most necessary fields. These keys usually start with "Experimenter" prefix (MatchEntrySerializerKey is an exception).

MessageTypeKey has these fields:

- *short version* - Openflow wire version number
- *Class<E> msgType* - DTO class

Scenario walkthrough

- [1] Serialization extensbility principles are similar to the deserialization principles. The scenario starts in a custom bundle. The custom bundle creates serializers which implement exposed OFSerializer / HeaderSerializer interfaces (wrapped under OFGeneralSerializer unifying super interface).
- [2] Created serializers are paired with their ExperimenterKeys, which are used for serializer lookup. If you don't know what key should be used with your serializer implementation, please visit *Registration keys* page.
- [3] Paired serializers are passed to the OF Library via **SwitchConnectionProvider**.*registerCustomSerializer(key, impl)*. Library registers the serializer.
- While registering, Library checks if the serializer is an instance of **SerializerRegistryInjector** interface. If yes, the SerializerRegistry (which stores all serializer references) is injected into the serializer.

This is particularly useful when the serializer needs access to other deserializers. For example IntructionsSerializer needs access to ActionsSerializer in order to be able to process OFPIT_WRITE_ACTIONS/OFPIT_APPLY_ACTIONS instructions.

Internal description

SwitchConnectionProvider `SwitchConnectionProvider` constructs and initializes both deserializer and serializer registries with default (de)serializers. It also injects the `DeserializerRegistry` into the

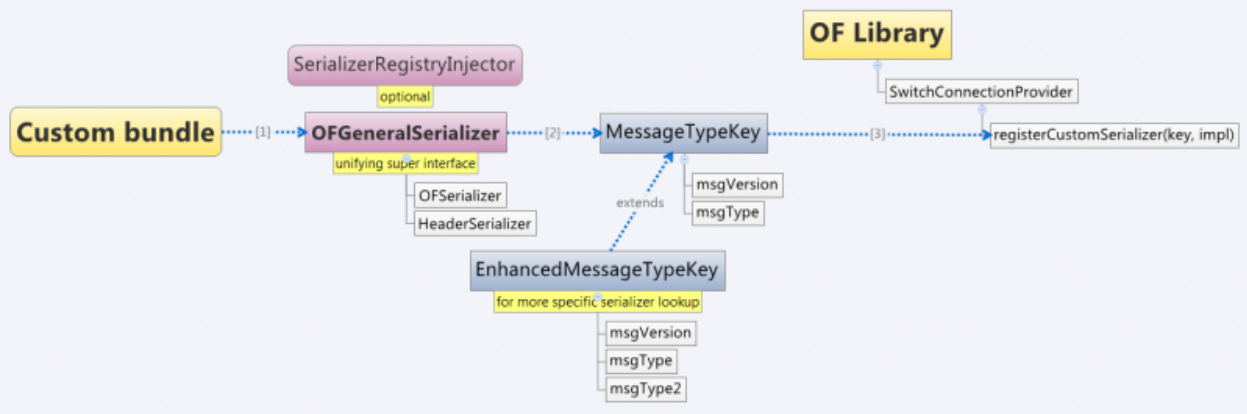


Fig. 2.38: Serialization scenario walkthrough

`DeserializationFactory`, the `SerializerRegistry` into the `SerializationFactory`. When call to register custom (de)serializer is made, `SwitchConnectionProvider` calls register method on appropriate registry.

DeserializerRegistry / SerializerRegistry Both registries contain init() method to initialize default (de)serializers. Registration checks if key or (de)serializer implementation are not `null`. If at least one of the is `null`, `NullPointerException` is thrown. Else the (de)serializer implementation is checked if it is `(De)SerializerRegistryInjector` instance. If it is an instance of this interface, the registry is injected into this (de)serializer implementation.

`GetSerializer(key)` or `GetDeserializer(key)` performs registry lookup. Because there are two separate interfaces that might be put into the registry, the registry uses their unifying super interface. Get(De)Serializer(key) method casts the super interface to desired type. There is also a null check for the (de)serializer received from the registry. If the deserializer wasn't found, `NullPointerException` with key description is thrown.

Registration keys

Deserialization.

Possible openflow extensions and their keys

There are three vendor specific extensions in Openflow v1.0 and eight in Openflow v1.3. These extensions are registered under registration keys, that are shown in table below:

Extension type	Open-Flow	Registration key	Utility class
Vendor message	1.0	ExperimenterIdDeserializerKey(1, experimenterId, ExperimenterMessage.class)	ExperimenterDeserializerKeyFactory
Action	1.0	ExperimenterActionDeserializerKey(1, experimenter ID)	.
Stats message	1.0	ExperimenterMultipartReplyMessageDeserializerKey(1, experimenter ID)	ExperimenterDeserializerKeyFactory
Experimenter message	1.3	ExperimenterIdDeserializerKey(4, experimenterId, ExperimenterMessage.class)	ExperimenterDeserializerKeyFactory
Match entry	1.3	MatchEntryDeserializerKey(4, (number) ${oxm_class}, (number) ${oxm_field});	.
		key.setExperimenterId(experimenter ID);	.
Action	1.3	ExperimenterActionDeserializerKey(4, experimenter ID)	.
Instruction	1.3	ExperimenterInstructionDeserializerKey(4, experimenter ID)	.
Multipart	1.3	ExperimenterIdDeserializerKey(4, experimenterId, MultipartReplyMessage.class)	ExperimenterDeserializerKeyFactory
Multipart - Table features	1.3	ExperimenterIdDeserializerKey(4, experimenterId, TableFeatureProperties.class)	ExperimenterDeserializerKeyFactory
Error	1.3	ExperimenterIdDeserializerKey(4, experimenterId, ErrorMessage.class)	ExperimenterDeserializerKeyFactory
Queue property	1.3	ExperimenterIdDeserializerKey(4, experimenterId, QueueProperty.class)	ExperimenterDeserializerKeyFactory
Meter band type	1.3	ExperimenterIdDeserializerKey(4, experimenterId, MeterBandExperimenterCase.class)	ExperimenterDeserializerKeyFactory

Table: **Deserialization**

Serialization.

Possible openflow extensions and their keys

There are three vendor specific extensions in Openflow v1.0 and seven Openflow v1.3. These extensions are registered under registration keys, that are shown in table below:

Extension type	Open-Flow	Registration key	Utility class
Vendor message	1.0	ExperimenterIdSerializerKey< >(1, experimenterId, ExperimenterInput.class)	ExperimenterSerialize rKeyFactory
Action	1.0	ExperimenterActionSerializer Key(1, experimenterId, sub-type)	.
Stats message	1.0	ExperimenterMultipartRequest SerializerKey(1, experimenter ID)	ExperimenterSerialize rKeyFactory
Experimenter message	1.3	ExperimenterIdSerializerKey< >(4, experimenterId, ExperimenterInput.class)	ExperimenterSerialize rKeyFactory
Match entry	1.3	MatchEntrySerializerKey<>(4, (class) ${oxm_class}, (class) ${oxm_field});	.
		key.setExperimenterId(experi menter ID)	.
Action	1.3	ExperimenterActionSerializer Key(4, experimenterId, sub-type)	.
Instruction	1.3	ExperimenterInstructionSeria lizerKey(4, experimenter ID)	.
Multipart	1.3	ExperimenterIdSerializerKey< >(4, experimenterId, MultipartRequestExperimenter Case.class)	ExperimenterSerialize rKeyFactory
Multipart - Table features	1.3	ExperimenterIdSerializerKey< >(4, experimenterId, TableFeatureProperties.class)	ExperimenterSerialize rKeyFactory
Meter band type	1.3	ExperimenterIdSerializerKey< >(4, experimenterId, MeterBandExperimenterCase.cl ass)	ExperimenterSerialize rKeyFactory

Table: **Serialization**

OpenFlow Plugin Project Developer Guide

This section covers topics which are developer specific and which have not been covered in the user guide. Please see the OpenFlow plugin user guide first.

It can be found on the OpenDaylight software download page.

Event Sequences

Session Establishment

The OpenFlow Protocol Library provides interface **SwitchConnectionHandler** which contains method *onSwitchConnected* (step 1). This event is raised in the OpenFlow Protocol Library when an OpenFlow device connects to OpenDaylight and caught in the **ConnectionManagerImpl** class in the OpenFlow plugin.

There the plugin creates a new instance of the **ConnectionContextImpl** class (step 1.1) and also instances of **HandshakeManagerImpl** (which uses **HandshakeListenerImpl**) and **ConnectionReadyListenerImpl**. **ConnectionReadyListenerImpl** contains method *onConnectionReady()* which is called when connection is prepared. This method starts the handshake with the OpenFlow device (switch) from the OpenFlow plugin side. Then handshake can be also started from device side. In this case method *shake()* from **HandshakeManagerImpl** is called (steps 1.1.1 and 2).

The handshake consists of an exchange of HELLO messages in addition to an exchange of device features (steps 2.1. and 3). The handshake is completed by **HandshakeManagerImpl**. After receiving device features, the **HandshakeListenerImpl** is notifed via the *onHanshakeSuccessfull()* method. After this, the device features, node id and connection state are stored in a **ConnectionContext** and the method *deviceConnected()* of **DeviceManagerImpl** is called.

When *deviceConnected()* is called, it does the following:

1. creates a new transaction chain (step 4.1)
2. creates a new instance of **DeviceContext** (step 4.2.2)
3. initializes the device context: the static context of device is populated by calling *createDeviceFeaturesForOF<version>()* to populate table, group, meter features and port descriptions (step 4.2.1 and 4.2.1.1)
4. creates an instance of **RequestContext** for each type of feature

When the OpenFlow device responds to these requests (step 4.2.1.1) with multipart replies (step 5) they are processed and stored to MD-SAL operational datastore. The *createDeviceFeaturesForOF<version>()* method returns a **Future** which is processed in the callback (step 5.1) (part of *initializeDeviceContext()* in the *deviceConnected()* method) by calling the method *onDeviceCtxLevelUp()* from **StatisticsManager** (step 5.1.1).

The call to *createDeviceFeaturesForOF<version>()*: . creates a new instance of **StatisticsContextImpl** (step 5.1.1.1).

1. calls *gatherDynamicStatistics()* on that instance which returns a **Future** which will produce a value when done
   (a) this method calls methods to get dynamic data (flows, tables, groups) from the device (step 5.1.1.2, 5.1.1.2.1, 5.1.1.2.1.1)
   (b) if everything works, this data is also stored in the MD-SAL operational datastore

If the **Future** is successful, it is processed (step 6.1.1) in a callback in **StatisticsManagerImpl** which:

1. schedules the next time to poll the device for statistics
2. sets the device state to synchronized (step 6.1.1.2)
3. calls *onDeviceContextLevelUp()* in **RpcManagerImpl**

The *onDeviceContextLevelUp()* call:

1. creates a new instance of **RequestContextImpl**
2. registers implementation for supported services
3. calls *onDeviceContextLevelUp()* in **DeviceManagerImpl** (step 6.1.1.2.1.2) which causes the information about the new device be be written to the MD-SAL operational datastore (step 6.1.1.2.2)

Handshake

The first thing that happens when an OpenFlow device connects to OpenDaylight is that the OpenFlow plugin gathers basic information about the device and establishes agreement on key facts like the version of OpenFlow which will be used. This process is called the handshake.

The handshake starts with HELLO message which can be sent either by the OpenFlow device or the OpenFlow plugin. After this, there are several scenarios which can happen:

1. if the first HELLO message contains a *version bitmap*, it is possible to determine if there is a common version of OpenFlow or not:
   (a) if there is a single common version use it and the **VERSION IS SETTLED**
   (b) if there are more than one common versions, use the highest (newest) protocol and the **VERSION IS SETTLED**

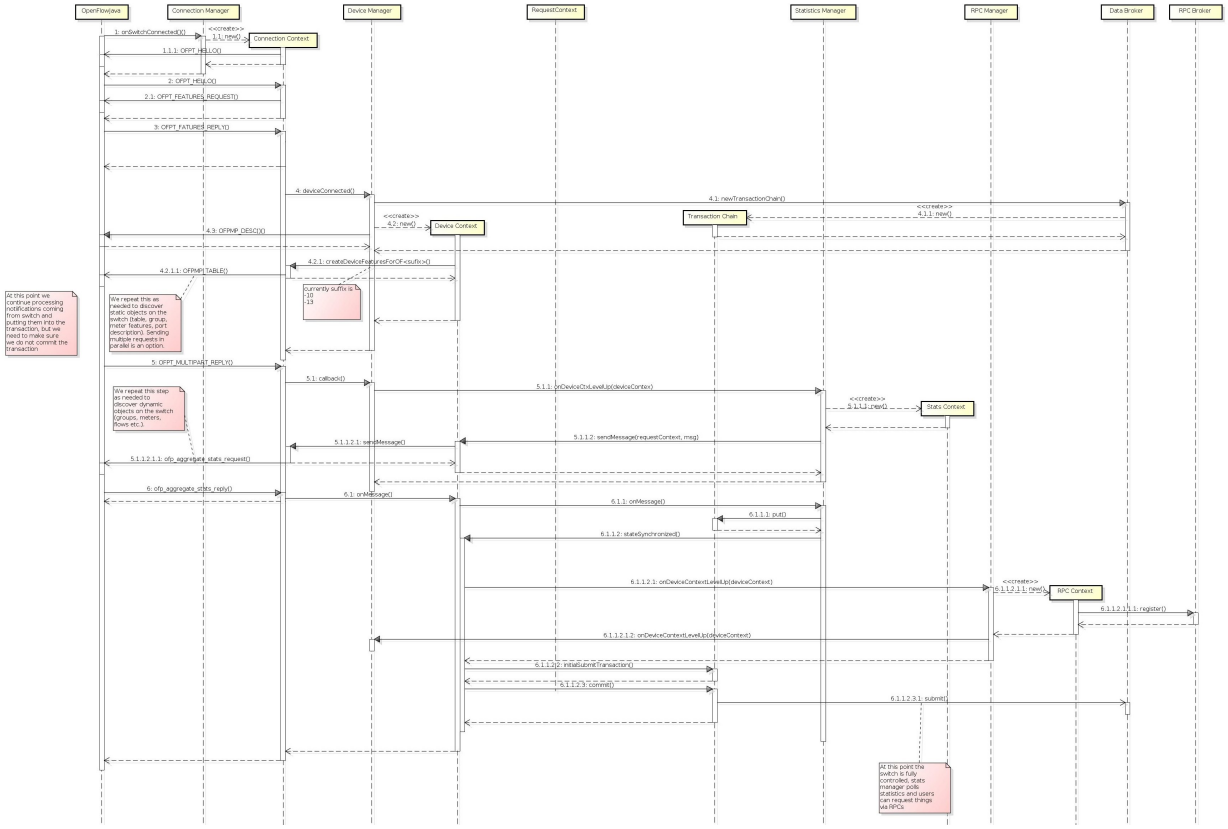


Fig. 2.39: Session establishment

(c) if there are no common versions, the device is **DISCONNECTED**

2. if the first HELLO message does not contain a *version bitmap*, then STEB-BY-STEP negotiation is used
3. if second (or more) HELLO message is received, then STEP-BY-STEP negotiation is used

STEP-BY-STEP negotiation:

- if last version proposed by the OpenFlow plugin is the same as the version received from the OpenFlow device, then the **VERSION IS SETTLED**
- if the version received in the current HELLO message from the device is the same as from previous then negotiation has failed and the device is **DISCONNECTED**
- if the last version from the device is greater than the last version proposed from the plugin, wait for the next HELLO message in the hope that it will advertise support for a lower version
- if the last version from the device is is less than the last version proposed from the plugin:
  - propose the highest version the plugin supports that is less than or equal to the version received from the device and wait for the next HELLO message
  - if if the plugin doesn't support a lower version, the device is **DISCONNECTED**

After selecting of version we can say that the **VERSION IS SETTLED** and the OpenFlow plugin can ask device for its features. At this point handshake ends.

Adding a Flow

There are two ways to add a flow in in the OpenFlow plugin: adding it to the MD-SAL config datastore or calling an RPC. Both of these can either be done using the native MD-SAL interfaces or using RESTCONF. This discussion focuses on calling the RPC.

If user send flow via REST interface (step 1) it will cause that *invokeRpc()* is called on **RpcBroker**. The **RpcBroker** then looks for an appropriate implementation of the interface. In the case of the OpenFlow plugin, this is the *addFlow()* method of **SalFlowServiceImpl** (step 1.1). The same thing happens if the RPC is called directly from the native MD-SAL interfaces.

The *addFlow()* method then

1. calls the *commitEntry()* method (step 2) from the OpenFlow Protocol Library which is responsible for sending the flow to the device
2. creates a new **RequestContext** by calling *createRequestContext()* (step 3)
3. creates a callback to handle any events that happen because of sending the flow to the device

The callback method is triggered when a barrier reply message (step 2.1) is received from the device indicating that the flow was either installed or an appropriate error message was sent. If the flow was successfully sent to the device, the RPC result is set to success (step 5). // **SalFlowService** contains inside method *addFlow()* other callback which caught notification from callback for barrier message.

At this point, no information pertaining to the flow has been added to the MD-SAL operational datastore. That is accomplished by the periodic gathering of statistics from OpenFlow devices.

The **StatisticsContext** for each given OpenFlow device periodically polls it using *gatherStatistics()* of **StatisticsGatheringUtil** which issues an OpenFlow OFPT_MULTIPART_REQUEST - OFPMP_FLOW. The response to this request (step 7) is processed in **StatisticsGatheringUtil** class where flow data is written to the MD-SAL operational datastore via the *writeToTransaction()* method of **DeviceContext**.

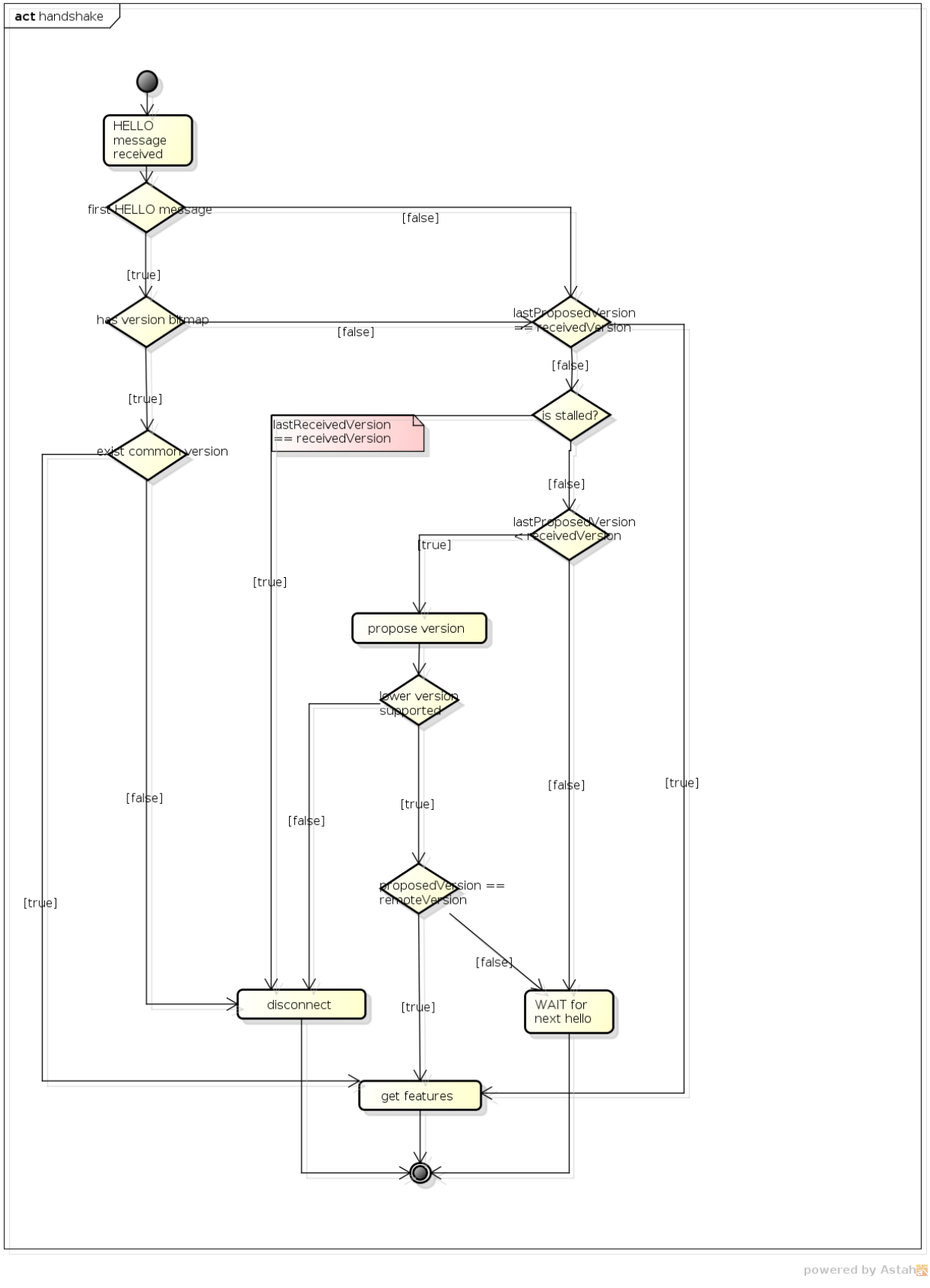


Fig. 2.40: Handshake process

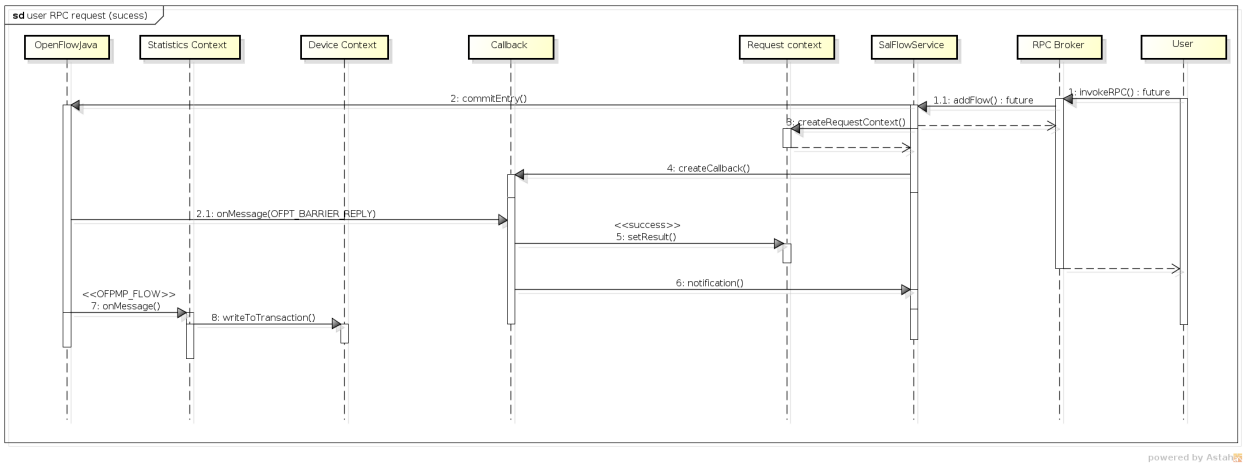


Fig. 2.41: Add flow

Description of OpenFlow Plugin Modules

The OpenFlow plugin project contains a variety of OpenDaylight modules, which are loaded using the configuration subsystem. This section describes the YANG files used to model each module.

General model (interfaces) - openflow-plugin-cfg.yang.

- the provided module is defined (`identity openflow-provider`)
- and target implementation is assigned (`...OpenflowPluginProvider`)

Implementation model - openflow-plugin-cfg-impl.yang

- the implementation of module is defined (`identity openflow-provider-impl`)
  - class name of generated implementation is defined (ConfigurableOpenFlowProvider)
- via augmentation the configuration of module is defined:
  - this module requires instance of binding-aware-broker (`container binding-aware-broker`)
  - and list of openflow-switch-connection-provider (those are provided by openflowjava, one plugin instance will orchestrate multiple openflowjava modules)

Generating config and sal classes out of yangs

In order to involve suitable code generators, this is needed in pom:

```
<build> ...
  <plugins>
    <plugin>
      <groupId>org.opendaylight.yangtools</groupId>
      <artifactId>yang-maven-plugin</artifactId>
      <executions>
        <execution>
          <goals>
            <goal>generate-sources</goal>
          </goals>
          <configuration>
            <codeGenerators>
```

```
              <generator>
                <codeGeneratorClass>
                  org.opendaylight.controller.config.yangjmxgenerator.plugin.
↪JMXGenerator
                </codeGeneratorClass>
                <outputBaseDir>${project.build.directory}/generated-sources/config</
↪outputBaseDir>
                <additionalConfiguration>
                  <namespaceToPackage1>
                    urn:opendaylight:params:xml:ns:yang:controller==org.opendaylight.
↪controller.config.yang
                  </namespaceToPackage1>
                </additionalConfiguration>
              </generator>
              <generator>
                <codeGeneratorClass>
                  org.opendaylight.yangtools.maven.sal.api.gen.plugin.
↪CodeGeneratorImpl
                </codeGeneratorClass>
                <outputBaseDir>${project.build.directory}/generated-sources/sal</
↪outputBaseDir>
              </generator>
              <generator>
                <codeGeneratorClass>org.opendaylight.yangtools.yang.unified.doc.
↪generator.maven.DocumentationGeneratorImpl</codeGeneratorClass>
                <outputBaseDir>${project.build.directory}/site/models</outputBaseDir>
              </generator>
            </codeGenerators>
            <inspectDependencies>true</inspectDependencies>
          </configuration>
        </execution>
      </executions>
      <dependencies>
        <dependency>
          <groupId>org.opendaylight.controller</groupId>
          <artifactId>yang-jmx-generator-plugin</artifactId>
          <version>0.2.5-SNAPSHOT</version>
        </dependency>
        <dependency>
          <groupId>org.opendaylight.yangtools</groupId>
          <artifactId>maven-sal-api-gen-plugin</artifactId>
          <version>${yangtools.version}</version>
          <type>jar</type>
        </dependency>
      </dependencies>
    </plugin>
    ...
```

- JMX generator (target/generated-sources/config)
- sal CodeGeneratorImpl (target/generated-sources/sal)

Altering generated files

Those files were generated under src/main/java in package as referred in yangs (if exist, generator will not overwrite them):

- ConfigurableOpenFlowProviderModuleFactory

  here the **instantiateModule** methods are extended in order to capture and inject osgi BundleContext into module, so it can be injected into final implementation - **OpenflowPluginProvider** + `module.setBundleContext(bundleContext);`

- ConfigurableOpenFlowProviderModule

  here the **createInstance** method is extended in order to inject osgi BundleContext into module implementation + `pluginProvider.setContext(bundleContext);`

Configuration xml file

Configuration file contains

- required capabilities
  - modules definitions from openflowjava
  - modules definitions from openflowplugin
- modules definition
  - openflow:switch:connection:provider:impl (listening on port 6633, name=openflow-switch-connection-provider-legacy-impl)
  - openflow:switch:connection:provider:impl (listening on port 6653, name=openflow-switch-connection-provider-default-impl)
  - openflow:common:config:impl (having 2 services (wrapping those 2 previous modules) and binding-broker-osgi-registry injected)
- provided services
  - openflow-switch-connection-provider-default
  - openflow-switch-connection-provider-legacy
  - openflow-provider

```
<snapshot>
 <required-capabilities>
   <capability>
↪urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:provider:impl?
↪module=openflow-switch-connection-provider-impl&revision=2014-03-28</capability>
   <capability>
↪urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:provider?
↪module=openflow-switch-connection-provider&revision=2014-03-28</capability>
   <capability>urn:opendaylight:params:xml:ns:yang:openflow:common:config:impl?
↪module=openflow-provider-impl&revision=2014-03-26</capability>
   <capability>urn:opendaylight:params:xml:ns:yang:openflow:common:config?
↪module=openflow-provider&revision=2014-03-26</capability>
 </required-capabilities>

 <configuration>


     <modules xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
       <module>
         <type xmlns:prefix=
↪"urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:provider:impl">
↪prefix:openflow-switch-connection-provider-impl</type>
```

```
        <name>openflow-switch-connection-provider-default-impl</name>
        <port>6633</port>
        <switch-idle-timeout>15000</switch-idle-timeout>
      </module>
      <module>
        <type xmlns:prefix=
↪"urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:provider:impl">
↪prefix:openflow-switch-connection-provider-impl</type>
        <name>openflow-switch-connection-provider-legacy-impl</name>
        <port>6653</port>
        <switch-idle-timeout>15000</switch-idle-timeout>
      </module>


      <module>
        <type xmlns:prefix=
↪"urn:opendaylight:params:xml:ns:yang:openflow:common:config:impl">prefix:openflow-
↪provider-impl</type>
        <name>openflow-provider-impl</name>

        <openflow-switch-connection-provider>
          <type xmlns:ofSwitch=
↪"urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:provider">
↪ofSwitch:openflow-switch-connection-provider</type>
          <name>openflow-switch-connection-provider-default</name>
        </openflow-switch-connection-provider>
        <openflow-switch-connection-provider>
          <type xmlns:ofSwitch=
↪"urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:provider">
↪ofSwitch:openflow-switch-connection-provider</type>
          <name>openflow-switch-connection-provider-legacy</name>
        </openflow-switch-connection-provider>


        <binding-aware-broker>
          <type xmlns:binding=
↪"urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding">binding:binding-
↪broker-osgi-registry</type>
          <name>binding-osgi-broker</name>
        </binding-aware-broker>
      </module>
    </modules>

    <services xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
      <service>
        <type xmlns:prefix=
↪"urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:provider">
↪prefix:openflow-switch-connection-provider</type>
        <instance>
          <name>openflow-switch-connection-provider-default</name>
          <provider>/modules/module[type='openflow-switch-connection-provider-impl
↪'][name='openflow-switch-connection-provider-default-impl']</provider>
        </instance>
        <instance>
          <name>openflow-switch-connection-provider-legacy</name>
          <provider>/modules/module[type='openflow-switch-connection-provider-impl
↪'][name='openflow-switch-connection-provider-legacy-impl']</provider>
        </instance>
```

```
      </service>

      <service>
        <type xmlns:prefix=
↪"urn:opendaylight:params:xml:ns:yang:openflow:common:config">prefix:openflow-
↪provider</type>
        <instance>
          <name>openflow-provider</name>
          <provider>/modules/module[type='openflow-provider-impl'][name='openflow-
↪provider-impl']</provider>
        </instance>
      </service>
    </services>


 </configuration>
</snapshot>
```

API changes

In order to provide multiple instances of modules from openflowjava there is an API change. Previously OFPlugin got access to SwitchConnectionProvider exposed by OFJava and injected collection of configurations so that for each configuration new instance of tcp listening server was created. Now those configurations are provided by configSubsystem and configured modules (wrapping the original SwitchConnectionProvider) are injected into OFPlugin (wrapping SwitchConnectionHandler).

Providing config file (IT, local distribution/base, integration/distributions/base)

openflowplugin-it

Here the whole configuration is contained in one file (controller.xml). Required entries needed in order to startup and wire OEPlugin + OFJava are simply added there.

OFPlugin/distribution/base

Here new config file has been added (src/main/resources/configuration/initial/42-openflow-protocol-impl.xml) and is being copied to config/initial subfolder of build.

integration/distributions/build

In order to push the actual config into config/initial subfolder of distributions/base in integration project there was a new artifact in OFPlugin created - **openflowplugin-controller-config**, containing only the config xml file under src/main/resources. Another change was committed into integration project. During build this config xml is being extracted and copied to the final folder in order to be accessible during controller run.

Internal message statistics API

To aid in testing and diagnosis, the OpenFlow plugin provides information about the number and rate of different internal events.

The implementation does two things: collects event counts and exposes counts. Event counts are grouped by message type, e.g., **PacketInMessage**, and checkpoint, e.g., *TO_SWITCH_ENQUEUED_SUCCESS*. Once gathered, the results are logged as well as being exposed using OSGi command line (deprecated) and JMX.

Collect

Each message is counted as it passes through various processing checkpoints. The following checkpoints are defined as a Java enum and tracked:

```
/**
  * statistic groups overall in OFPlugin
  */
enum STATISTIC_GROUP {
     /** message from switch, enqueued for processing */
     FROM_SWITCH_ENQUEUED,
     /** message from switch translated successfully - source */
     FROM_SWITCH_TRANSLATE_IN_SUCCESS,
     /** message from switch translated successfully - target */
     FROM_SWITCH_TRANSLATE_OUT_SUCCESS,
     /** message from switch where translation failed - source */
     FROM_SWITCH_TRANSLATE_SRC_FAILURE,
     /** message from switch finally published into MD-SAL */
     FROM_SWITCH_PUBLISHED_SUCCESS,
     /** message from switch - publishing into MD-SAL failed */
     FROM_SWITCH_PUBLISHED_FAILURE,

     /** message from MD-SAL to switch via RPC enqueued */
     TO_SWITCH_ENQUEUED_SUCCESS,
     /** message from MD-SAL to switch via RPC NOT enqueued */
     TO_SWITCH_ENQUEUED_FAILED,
     /** message from MD-SAL to switch - sent to OFJava successfully */
     TO_SWITCH_SUBMITTED_SUCCESS,
     /** message from MD-SAL to switch - sent to OFJava but failed*/
     TO_SWITCH_SUBMITTED_FAILURE
}
```

When a message passes through any of those checkpoints then counter assigned to corresponding checkpoint and message is incremented by 1.

Expose statistics

As described above, there are three ways to access the statistics:

- OSGi command line (this is considered deprecated)

  ```
  osgi> dumpMsgCount
  ```

- OpenDaylight logging console (statistics are logged here every 10 seconds)

  required logback settings : `<logger name="org.opendaylight.openflowplugin.openflow.md.queue.MessageSpyCounterImpl" level="DEBUG"\/>`

- JMX (via JConsole)

  start OpenFlow plugin with the `-jmx` parameter

  start JConsole by running `jconsole`

the JConsole MBeans tab should contain org.opendaylight.controller

RuntimeBean has a msg-spy-service-impl

Operations provides makeMsgStatistics report functionality

Example results

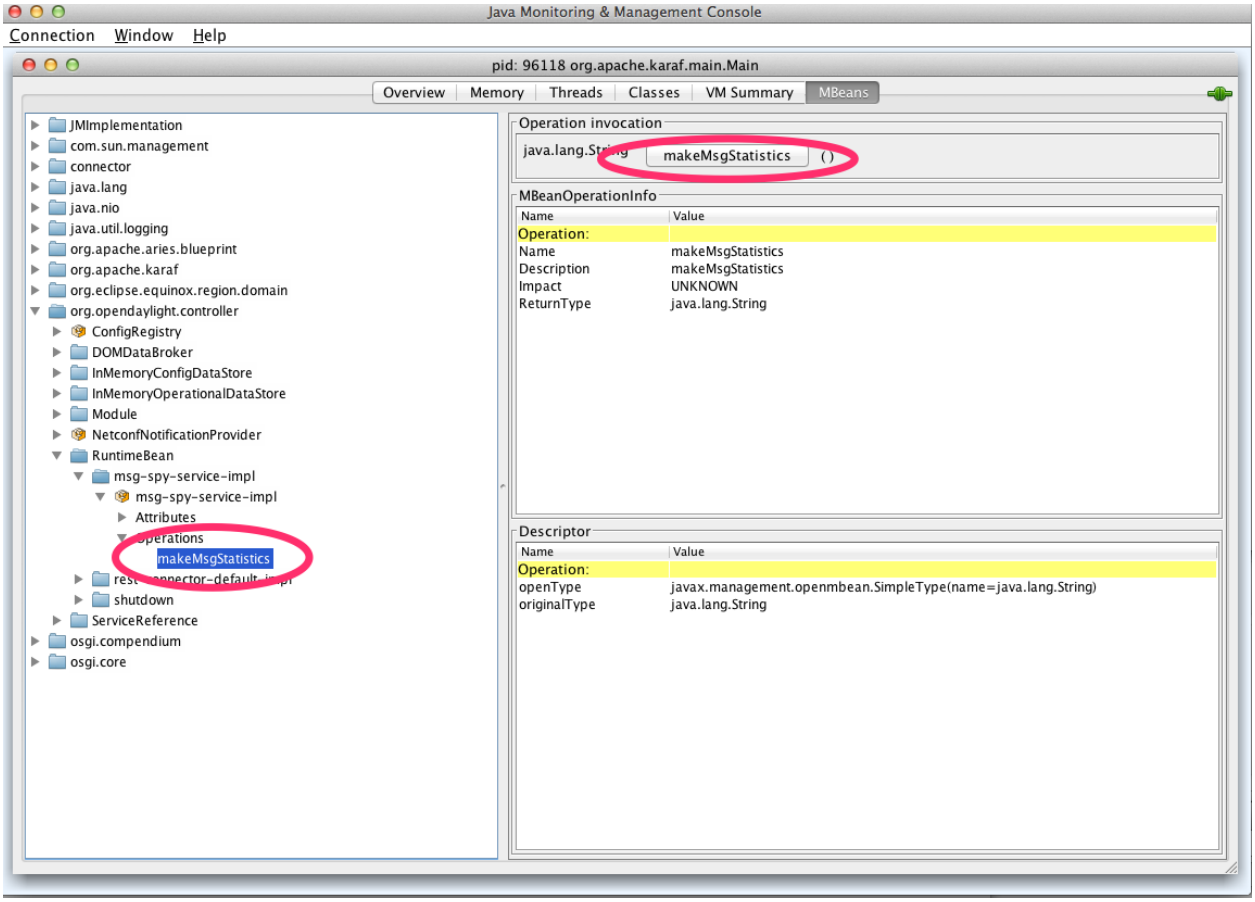


Fig. 2.42: OFplugin Debug stats.png

```
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_ENQUEUED: MSG[PortStatusMessage] ->␣
↪+0 | 1
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_ENQUEUED:␣
↪MSG[MultipartReplyMessage] -> +24 | 81
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_ENQUEUED: MSG[PacketInMessage] ->␣
↪+8 | 111
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_IN_SUCCESS:␣
↪MSG[PortStatusMessage] -> +0 | 1
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_IN_SUCCESS:␣
↪MSG[MultipartReplyMessage] -> +24 | 81
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_IN_SUCCESS:␣
↪MSG[PacketInMessage] -> +8 | 111
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_OUT_SUCCESS:␣
↪MSG[QueueStatisticsUpdate] -> +3 | 7
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_OUT_SUCCESS:␣
↪MSG[NodeUpdated] -> +0 | 3
```

```
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_OUT_SUCCESS:␣
↪MSG[NodeConnectorStatisticsUpdate] -> +3 | 7
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_OUT_SUCCESS:␣
↪MSG[GroupDescStatsUpdated] -> +3 | 7
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_OUT_SUCCESS:␣
↪MSG[FlowsStatisticsUpdate] -> +3 | 19
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_OUT_SUCCESS:␣
↪MSG[PacketReceived] -> +8 | 111
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_OUT_SUCCESS:␣
↪MSG[MeterFeaturesUpdated] -> +0 | 3
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_OUT_SUCCESS:␣
↪MSG[GroupStatisticsUpdated] -> +3 | 7
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_OUT_SUCCESS:␣
↪MSG[GroupFeaturesUpdated] -> +0 | 3
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_OUT_SUCCESS:␣
↪MSG[MeterConfigStatsUpdated] -> +3 | 7
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_OUT_SUCCESS:␣
↪MSG[MeterStatisticsUpdated] -> +3 | 7
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_OUT_SUCCESS:␣
↪MSG[NodeConnectorUpdated] -> +0 | 12
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_OUT_SUCCESS:␣
↪MSG[FlowTableStatisticsUpdate] -> +3 | 8
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_TRANSLATE_SRC_FAILURE: no activity␣
↪detected
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_PUBLISHED_SUCCESS:␣
↪MSG[QueueStatisticsUpdate] -> +3 | 7
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_PUBLISHED_SUCCESS: MSG[NodeUpdated]␣
↪-> +0 | 3
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_PUBLISHED_SUCCESS:␣
↪MSG[NodeConnectorStatisticsUpdate] -> +3 | 7
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_PUBLISHED_SUCCESS:␣
↪MSG[GroupDescStatsUpdated] -> +3 | 7
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_PUBLISHED_SUCCESS:␣
↪MSG[FlowsStatisticsUpdate] -> +3 | 19
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_PUBLISHED_SUCCESS:␣
↪MSG[PacketReceived] -> +8 | 111
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_PUBLISHED_SUCCESS:␣
↪MSG[MeterFeaturesUpdated] -> +0 | 3
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_PUBLISHED_SUCCESS:␣
↪MSG[GroupStatisticsUpdated] -> +3 | 7
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_PUBLISHED_SUCCESS:␣
↪MSG[GroupFeaturesUpdated] -> +0 | 3
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_PUBLISHED_SUCCESS:␣
↪MSG[MeterConfigStatsUpdated] -> +3 | 7
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_PUBLISHED_SUCCESS:␣
↪MSG[MeterStatisticsUpdated] -> +3 | 7
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_PUBLISHED_SUCCESS:␣
↪MSG[NodeConnectorUpdated] -> +0 | 12
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_PUBLISHED_SUCCESS:␣
↪MSG[FlowTableStatisticsUpdate] -> +3 | 8
DEBUG o.o.o.s.MessageSpyCounterImpl - FROM_SWITCH_PUBLISHED_FAILURE: no activity␣
↪detected
DEBUG o.o.o.s.MessageSpyCounterImpl - TO_SWITCH_ENQUEUED_SUCCESS: MSG[AddFlowInput] ->
↪ +0 | 12
DEBUG o.o.o.s.MessageSpyCounterImpl - TO_SWITCH_ENQUEUED_FAILED: no activity detected
DEBUG o.o.o.s.MessageSpyCounterImpl - TO_SWITCH_SUBMITTED_SUCCESS: MSG[AddFlowInput] -
↪> +0 | 12
DEBUG o.o.o.s.MessageSpyCounterImpl - TO_SWITCH_SUBMITTED_FAILURE: no activity␣
↪detected
```

Application: Forwarding Rules Synchronizer

Basics

Description

Forwarding Rules Synchronizer (FRS) is a newer version of Forwarding Rules Manager (FRM). It was created to solve most shortcomings of FRM. FRS solving errors with retry mechanism. Sending barrier if needed. Using one service for flows, groups and meters. And it has less changes requests send to device since calculating difference and using compression queue.

It is located in the Java package:

```
package org.opendaylight.openflowplugin.applications.frsync;
```

Listeners

- 1x config - FlowCapableNode
- 1x operational - Node

System of work

- one listener in config datastore waiting for changes
  - update cache
  - skip event if operational not present for node
  - send syncup entry to reactor for synchronization
    - node added: after part of modification and whole operational snapshot
    - node updated: after and before part of modification
    - node deleted: null and before part of modification
- one listener in operational datastore waiting for changes
  - update cache
  - on device connected
    - register for cluster services
  - on device disconnected remove from cache
    - remove from cache
    - unregister for cluster services
  - if registered for reconciliation
    - do reconciliation through syncup (only when config present)
- reactor *(provides syncup w/decorators assembled in this order)*

  - Cluster decorator - skip action if not master for device
  - FutureZip decorator (FutureZip extends Future decorator)
    * Future - run delegate syncup in future - submit task to executor service
    * FutureZip - provides state compression - compress optimized config delta if waiting for execution with new one
  - Guard decorator - per device level locking
  - Retry decorator - register for reconciliation if syncup failed
  - Reactor impl - calculate diff from after/before parts of syncup entry and execute

Strategy

In the *old* FRM uses an incremental strategy with all changes made one by one, where FRS uses a flat batch system with changes made in bulk. It uses one service SalFlatBatchService instead of three (flow, group, meter).

Boron release

FRS is used in Boron as separate feature and it is not loaded by any other feature. It has to be run separately.

```
odl-openflowplugin-app-forwardingrules-sync
```

FRS additions

Retry mechanism

- is started when change request to device return as failed (register for reconcile)
- wait for next consistent operational and do reconciliation with actual config (not only diff)

ZipQueue

- only the diff (before/after) between last config changes is sent to device
- when there are more config changes for device in a row waiting to be processed they are compressed into one entry (after is still replaced with the latest)

Cluster-aware

- FRS is cluster aware using ClusteringSingletonServiceProvider from the MD-SAL
- on mastership change reconciliation is done (register for reconcile)

SalFlatBatchService

FRS uses service with implemented barrier waiting logic between dependent objects

Service: SalFlatBatchService

Basics

SalFlatBatchService was created along forwardingrules-sync application as the service that should application used by default. This service uses only one input with bag of flow/group/meter objects and their common add/update/remove action. So you practically send only one input (of specific bags) to this service.

- interface: *org.opendaylight.yang.gen.v1.urn.opendaylight.flat.batch.service.rev160321.SalFlatBatchService*
- implementation: *org.opendaylight.openflowplugin.impl.services.SalFlatBatchServiceImpl*
- method: *processFlatBatch(input)*
- input: *org.opendaylight.yang.gen.v1.urn.opendaylight.flat.batch.service.rev160321.ProcessFlatBatchInput*

Usage benefits

- possibility to use only one input bag with particular failure analysis preserved
- automatic barrier decision (chain+wait)
- less RPC routing in cluster environment (since one call encapsulates all others)

ProcessFlatBatchInput

Input for SalFlatBatchService (ProcessFlatBatchInput object) consists of:

- node - NodeRef
- batch steps - List<Batch> - defined action + bag of objects + order for failures analysis
  - BatchChoice - yang-modeled action choice (e.g. FlatBatchAddFlowCase) containing batch bag of objects (e.g. flows to be added)
  - BatchOrder - (integer) order of batch step (should be incremented by single action)
- exitOnFirstError - boolean flag

Workflow

1. prepare **list of steps** based on input
2. **mark barriers** in steps where needed
3. prepare particular **F/G/M-batch** service calls from **Flat-batch** steps
   - F/G/M-batch services encapsulate bulk of single service calls
   - they actually chain barrier after processing all single calls if actual step is marked as barrier-needed
4. **chain** futures and **start** executing
   - start all actions that can be run simultaneously (chain all on one starting point)
   - in case there is a step marked as barrier-needed
     - wait for all fired jobs up to one with barrier
     - merge rpc results (status, errors, batch failures) into single one

  - the latest job with barrier is new starting point for chaining

Services encapsulation

- SalFlatBatchService
  - SalFlowBatchService
    - SalFlowService
  - SalGroupBatchService
    - SalGroupService
  - SalMeterBatchService
    - SalMeterService

Barrier decision

- decide on actual step and all previous steps since the latest barrier
- if condition in table below is satisfied the latest step before actual is marked as barrier-needed

actual step	previous steps contain
FLOW_ADD *or* FLOW_UPDATE	GROUP_ADD *or* METER_ADD
GROUP_ADD	GROUP_ADD *or* GROUP_UPDATE
GROUP_REMOVE	FLOW_UPDATE *or* FLOW_REMOVE *or* GROUP_UPDATE *or* GROUP_REMOVE
METER_REMOVE	FLOW_UPDATE *or* FLOW_REMOVE

Error handling

There is flag in ProcessFlatBatchInput to stop process on the first error.

- *true* - if partial step is not successful stop whole processing
- *false* (default) - try to process all steps regardless partial results

If error occurs in any of partial steps upper FlatBatchService call will return as unsuccessful in both cases. However every partial error is attached to general flat batch result along with BatchFailure (contains BatchOrder and BatchItemIdChoice to identify failed step).

Cluster singleton approach in plugin

Basics

Description

The existing OpenDaylight service deployment model assumes symmetric clusters, where all services are activated on all nodes in the cluster. However, many services require that there is a single active service instance per cluster. We call such services *singleton services*. The Entity Ownership Service (EOS) represents the base Leadership choice for one Entity instance. Every Cluster Singleton service **type** must have its own Entity and every Cluster Singleton service **instance** must have its own Entity Candidate. Every registered Entity Candidate should be notified about

its actual role. All this "work" is done by MD-SAL so the Openflowplugin need "only" to register as service in **SingletonClusteringServiceProvider** given by MD-SAL.

Change against using EOS service listener

In this new clustering singleton approach plugin uses API from the MD-SAL project: SingletonClusteringService which comes with three methods.

```
instantiateServiceInstance()
closeServiceInstance()
getIdentifier()
```

This service has to be registered to a SingletonClusteringServiceProvider from MD-SAL which take care if mastership is changed in cluster environment.

First method in SingletonClusteringService is being called when the cluster node becomes a MASTER. Second is being called when status changes to SLAVE or device is disconnected from cluster. Last method plugins returns NodeId as ServiceGroupIdentifier Startup after device is connected

On the start up the plugin we need to initialize first four managers for each working area providing information and services

- Device manager
- RPC manager
- Role manager
- Statistics manager

After connection the device the listener Device manager get the event and start up to creating the context for this connection. Startup after device connection

Services are managed by SinlgetonClusteringServiceProvider from MD-SAL project. So in startup we simply create a instance of LifecycleService and register all contexts into it.

Role change

Plugin is no longer registered as Entity Ownership Service (EOS) listener therefore does not need to and cannot respond on EOS ownership changes.

Service start

Services start asynchronously but the start is managed by LifecycleService. If something goes wrong LifecycleService stop starting services in context and this speeds up the reconnect process. But the services haven't changed and plugin need to start all this:

- Activating transaction chain manager
- Initial gathering of device statistics
- Initial submit to DS
- Sending role MASTER to device
- RPC services registration
- Statistics gathering start

Service stop

If closeServiceInstance occurred plugin just simply try to store all unsubmitted transactions and close the transaction chain manager, stop RPC services, stop Statistics gathering and after that all unregister txEntity from EOS.

Yang models and API

Model
Openflow basic types
opendaylight-table-types.yang
opendaylight-action-types.yang
opendaylight-flow-types.yang
opendaylight-meter-types.yang
opendaylight-group-types.yang
opendaylight-match-types.yang
opendaylight-port-types.yang
opendaylight-queue-types.yang
Openflow services
sal-table.yang
sal-group.yang
sal-queue.yang
flow-errors.yang
flow-capable-transaction.yang
sal-flow.yang
sal-meter.yang
flow-topology-discovery.yang
node-errors.yang
node-config.yang
sal-echo.yang
sal-port.yang
packet-processing.yang
flow-node-inventory.yang
Openflow statistics
opendaylight-queue-statistics.yang
opendaylight-flow-table-statistics.yang
opendaylight-port-statistics.yang
opendaylight-statistics-types.yang
opendaylight-group-statistics.yang
opendaylight-flow-statistics.yang
opendaylight-meter-statistics.yang

Karaf feature tree

Short HOWTO create such a tree.

Fig. 2.43: Openflow plugin karaf feature tree

Wiring up notifications

Introduction

We need to translate OpenFlow messages coming up from the OpenFlow Protocol Library into MD-SAL Notification objects and then publish them to the MD-SAL.

Mechanics

1. Create a Translator class
2. Register the Translator
3. Register the notificationPopListener to handle your Notification Objects

Create a Translator class

You can see an example in PacketInTranslator.java.

First, simply create the class

```
public class PacketInTranslator implements IMDMessageTranslator<OfHeader, List
↪<DataObject>> {
```

Then implement the translate function:

```
public class PacketInTranslator implements IMDMessageTranslator<OfHeader, List
↪<DataObject>> {

    protected static final Logger LOG = LoggerFactory
            .getLogger(PacketInTranslator.class);
    @Override
    public PacketReceived translate(SwitchConnectionDistinguisher cookie,
            SessionContext sc, OfHeader msg) {
            ...
    }
```

Make sure to check that you are dealing with the expected type and cast it:

```
if(msg instanceof PacketInMessage) {
    PacketInMessage message = (PacketInMessage)msg;
    List<DataObject> list = new CopyOnWriteArrayList<DataObject>();
```

Do your transation work and return

```
PacketReceived pktInEvent = pktInBuilder.build();
list.add(pktInEvent);
return list;
```

Register your Translator Class

Next you need to go to MDController.java and in init() add register your Translator:

```
public void init() {
        LOG.debug("Initializing!");
        messageTranslators = new ConcurrentHashMap<>();
        popListeners = new ConcurrentHashMap<>();
        //TODO: move registration to factory
        addMessageTranslator(ErrorMessage.class, OF10, new ErrorTranslator());
        addMessageTranslator(ErrorMessage.class, OF13, new ErrorTranslator());
        addMessageTranslator(PacketInMessage.class,OF10, new PacketInTranslator());
        addMessageTranslator(PacketInMessage.class,OF13, new PacketInTranslator());
```

Notice that there is a separate registration for each of OpenFlow 1.0 and OpenFlow 1.3. Basically, you indicate the type of OpenFlow Protocol Library message you wish to translate for, the OpenFlow version, and an instance of your Translator.

Register your MD-SAL Message for Notification to the MD-SAL

Now, also in MDController.init() register to have the notificationPopListener handle your MD-SAL Message:

```
addMessagePopListener(PacketReceived.class, new NotificationPopListener<DataObject>
↪());
```

You are done

That's all there is to it. Now when a message comes up from the OpenFlow Protocol Library, it will be translated and published to the MD-SAL.

Message Order Preservation

While the Helium release of OpenFlow Plugin relied on queues to ensure messages were delivered in order, subsequent releases instead ensure that all the messages from a given device are delivered using the same thread and thus message order is guaranteed without queues. The OpenFlow plugin allocates a number of threads equal to twice the number of processor cores on machine it is run, e.g., 8 threads if the machine has 4 cores.

Note: While each device is assigned to one thread, multiple devices can be assigned to the same thread.

OpFlex agent-ovs Developer Guide

Overview

agent-ovs is a policy agent that works with OVS to enforce a group-based policy networking model with locally attached virtual machines or containers. The policy agent is designed to work well with orchestration tools like OpenStack.

agent-ovs Architecture

agent-ovs uses libopflex to communicate with an OpFlex-based policy repository to enforce policy on network endpoints attached to OVS by an orchestration system.

The key components are:

- Agent - coordinates startup and configuration
- Renderers - Renderers are responsible for rendering policy. This is a very general mechanism but the currently-implemented renderer is the stitched-mode renderer that can work along with with hardware fabrics such as ACI that support policy enforcement.
- EndpointManager - Keep track of network endpoints and declare them to the endpoint repository
- PolicyManager - Keep track of and index policies
- IntFlowManager - render policies to OVS integration bridge
- AccessFlowManager - render policies to OVS access bridge

API Reference Documentation

Internal API documentation can be found by in doc/html/index.html in any build.

OpFlex genie Developer Guide

Overview

Genie is a tool for code generation from a model. It supports generating C++ and Java code. C++ can be generated suitable for use with libopflex. C++ and Java can be generated as a plain set of objects.

Group-based Policy Model

The group-based policy model is included with the genie tool and can be found under the MODEL directory. By running mvn exec:java, libmodelgbp will be generated as a library project that, when built and installed, will work with libopflex. This model is used by the OVS agent.

API Reference Documentation

Complete API documentation for the generated libmodelgbp can be found in doc/html/index.html in any build

OpFlex libopflex Developer Guide

Overview

The OpFlex framework allows you to develop agents that can communicate using the OpFlex protocol and act as a policy element in an OpFlex-based distributed control system. The OpFlex architecture provides a distributed control system based on a declarative policy information model. The policies are defined at a logically centralized policy repository and enforced within a set of distributed policy elements. The policy repository communicates with the subordinate policy elements using the OpFlex control protocol. This protocol allows for bidirectional communication of policy, events, statistics, and faults.

Rather than simply providing access to the OpFlex protocol, this framework allows you to directly manipulate a management information tree containing a hierarchy of managed objects. This tree is kept in sync as needed with other policy elements in the system, and you are automatically notified when important changes to the model occur. Additionally, we can ensure that only those managed objects that are important to the local policy element are synchronized locally.

Object Model

Interactions with the OpFlex framework happen through the management information tree. This is a tree of managed objects defined by an object model specific to your application. There are a few important major category of objects that can appear in the model.

- First, there is the policy object. A policy object represents some data related to a policy that describes a user intent for how the system should behave. A policy object is stored in the policy repository which is the source of "truth" for this object.
- Second, there is an endpoint object. A endpoint represents an entity in the system to which we want to apply policy, which could be a network interface, a storage array, or other relevent policy endpoint. Endpoints are discovered and reported by policy elements locally, and are synchronized into the endpoint repository. The originating policy element is the source of truth for the endpoints it discovers. Policy elements can retrieve information about endpoints discovered by other policy elements by resolving endpoints from the endpoint repository.
- Third, there is the observable object. An observable object represents some state related to the operational status or health of the policy element. Observable objects will be reported to the observer.
- Finally, there is the local-only object. This is the simplest object because it exists only local to a particular policy element. These objects can be used to store state specific to that policy element, or as helpers to resolve other objects. Read on to learn more.

You can use the genie tool that is included with the framework to produce your application model along with a set of generated accessor classes that can work with this framework library. You should refer to the documentation that accompanies the genie tool for information on how to use to to generate your object model. Later in this guide, we'll go through examples of how to use the generated managed object accessor classes.

Programming by Side Effect

When developing software on the OpFlex framework, you'll need to think in a slightly different way. Rather than calling an API function that would perform some specific action, you'll need to write a managed object to the managed object database. Writing that object to the store will trigger the side effect of performing the action that you want.

For example, a policy element will need to have a component responsible for discovering policy endpoints. When it discovers a policy endpoint, it would write an endpoint object into the managed object database. That endpoint object will contain a reference to policy that is relevant to the endpoint object. This will trigger a whole cascade of events.

First, the framework will notice that an endpoint object has been created and it will write it to the endpoint repository. Second, the framework to will attempt to resolve the unresolved reference to the relevent policy object. There might be a whole chain of policy resolutions that will be automatically performed to download all the relevent policy until there are no longer any dangling references.

As long as there is a locally-created object in the system with a reference to that policy, the framework will continually ensure that the policy and any transitive policies are kept up to date. The policy element can subscribe to updates to these policy classes that will be invoked either the first time the policy is resolved or any time the policy changes.

A consequence of this design is that the managed object database can be temporarily in an inconsistent state with unresolved dangling references. Eventually, however, the inconsistency will be fully resolved. The policy element must be able to cleanly handle partially-resolved or inconsistent state and eventually reach the correct state as it receives these update notifications. Note that, in the OpFlex architecture, when there is no policy that specifically allows a particular action, that action must be prevented.

Let's cover one slightly more complex example. If a policy element needs to discover information about an endpoint that is not local to that policy element, it will need to retrieve that information from the endpoint repository. However, just as there is no API call to retrieve a policy object from the policy repository, there is no API call to retrieve an endpoint from the endpoint repository.

To get this information, the policy element needs to create a local-only object that references the endpoint. Once it creates this local-only object, if the endpoint is not already resolved, the framework will notice the dangling reference and automatically resolve the endpoint from the endpoint respository. When the endpoint resolution completes, the framework deliver an update notification to the policy element. The policy element will continue to receive any updates related to that endpoint until the policy element remove the local-only reference to the object. Once this occurs, the framework can garbage-collect any unreferenced objects.

Threading and Ownership

The OpFlex framework uses a somewhat unique threading model. Each managed object in the system belongs to a particular owner. An owner would typically be a single thread that is reponsible for all updates to objects with that owner. Though anything can read the state of a managed object, only the owner of a managed object is permitted to write to it. Though this is not strictly required for correctness, the performance of the system wil be best if you ensure that only one thread at a time is writing to objects with a particular owner.

Change notifications are delivered in a serialized fashion by a single thread. Blocking this thread from a notification callback will stall delivery of all notifications. It is therefore best practice to ensure that you do not block or perform long-running operations from a notification callback.

Key APIs and Interfaces

Basic Usage and Initialization

The primary interface point into the framework is opflex::ofcore::OFFramework. You can choose to instantiate your own copy of the framework, or you can use the static default instance.

Before you can use the framework, you must initialize it by installing your model metadata. The model metadata is accessible through the generated model library. In this case, it assumes your model is called "mymodel":

```
#include <opflex/ofcore/OFFramework.h>
#include <mymodel/metadata/metadata.hpp>
// ...
using opflex::ofcore::OFFramework;
OFFramework::defaultInstance().setModel(mymodel::getMetadata());
```

The other critical piece of information required for initialization is the OpFlex identity information. The identity information is required in order to successfully connect to OpFlex peers. In OpFlex, each component has a unique name within its policy domain, and each policy domain is identified by a globally unique domain name. You can set this identity information by calling:

```
OFFramework::defaultInstance()
    .setOpflexIdentity("[component name]", "[unique domain]");
```

You can then start the framework simply by calling:

```
OFFramework::defaultInstance().start();
```

Finally, you can add peers after the framework is started by calling the `opflex::ofcore::OFFramework::addPeer` method:

```
OFFramework::defaultInstance().addPeer("192.168.1.5", 1234);
```

When connecting to the peer, that peer may provide an additional list of peers to connect to, which will be automatically added as peers. If the peer does not include itself in the list, then the framework will disconnect from that peer and add the peers in the list. In this way, it is possible to automatically bootstrap the correct set of peers using a known hostname or IP address or a known, fixed anycast IP address.

To cleanly shut down, you can call:

```
OFFramework::defaultInstance().stop();
```

Working with Data in the Tree

Reading from the Tree

You can access data in the managed tree using the generated accessor classes. The details of exactly which classes you'll use will depend on the model you're using, but let's assume that we have a simple model called "simple" with the following classes:

- root - The root node. The URI for the root node is "/"
- foo - A policy object, and a child of root, with a scalar string property called "bar", and a unsigned 64-bit integer property called baz. The bar property is the naming property for foo. The URI for a foo object would be "/foo/[value of bar]/"
- fooref - A local-only child of root, with a reference to a foo, and a scalar string property called "bar". The bar property is the naming property for foo. The URI for a fooref object would be "/fooref/[value of bar]/"

In this example, we'll have a generated class for each of the objects. There are two main ways to get access to an object in the tree.

First, we can get instantiate an accessor class to any node in the tree by calling one of its static resolve functions. The resolve functions can take either an already-built URI that represents the object, or you can call the version that will locate the object by its naming properties.

Second, we can access the object also from its parent object using the appropriate child resolver member functions.

However we read it, the object we get back is an immutable view into the object it references. The properties set locally on that object will not change even though the underlying object may have been updated in the store. Note, however, that its children can change between when you first retrieve the object and when you resolve any children.

Another thing that is critical to note again is that when you attempt to resolve an object, you can get back nothing, even if the object actually does exist on another OpFlex node. You must ensure that some object in the managed object database references the remote managed object you want before it will be visible to you.

To get access to the root node using the default framework instance, we can simply call:

```
using boost::shared_ptr;
using boost::optional;
optional<shared_ptr<simple::root> > r(simple::root::resolve());
```

Note that whenever we can a resolve function, we get back our data in the form of an optional shared pointer to the object instance. If the node does not exist, then the optional will be set to boost::none. Note that if you dereference an optional that has not been set, you'll trigger an assert, so you must check the return as follows:

```
if (!r) {
   // handle missing object
}
```

Now let's get a child node of the root in three different ways:

```
// Get foo1 by constructing its URI from the root
optional<shared_ptr<simple::foo> > foo1(simple::foo::resolve("test"));
// get foo1 by constructing its URI relative to its parent
foo1 = r.get()->resolveFoo("test");
// get foo1 by manually building its URI
foo1 = simple::foo::resolve(opflex::modb::URIBuilder()
                            .addElement("foo")
                            .addElement("test")
                            .build());
```

All three of these calls will give us the same object, which is the "foo" object located at "/foo/test/".

The foo class has a single string property called "bar". We can easily access it as follows:

```
const std::string& barv = foo1.getBar();
```

Writing to the Tree

Writing to the tree is nearly as easy as reading from it. The key concept to understand is the mutator object. If you want to make changes to the tree, you must allocate a mutator object. The mutator will register itself in some thread-local storage in the framework instance you're using. The mutator is specific to a single "owner" for the data, so you can only make changes to data associated with that owner.

Whenever you modify one of the accessor classes, the change is actually forwarded to the currently-active mutator. You won't see any of the changes you make until you call the commit member function on the mutator. When you do that, all the changes you made are written into the store.

Once the changes are written into the store, you will need to call the appropriate resolve function again to see the changes.

Allocating a mutator is simple. To create a mutator for the default framework instance associated with the owner "owner1", just allocate the mutator on the stack. Be sure to call commit() before it goes out of scope or you'll lose your changes.

```
{
    opflex::modb::Mutator mutator("owner1");
    // make changes here
```

```
    mutator.commit();
}
```

Note that if an exception is thrown while making changes but before committing, the mutator will go out of scope and the changes will be discarded.

To create a new node, you must call the appropriate add[Child] member function on its parent. This function takes parameters for each of the naming properties for the object:

```
shared_ptr<simple::foo> newfoo(root->addFoo("test"));
```

This will return a shared pointer to a new foo object that has been registered in the active mutator but not yet committed. The "bar" naming property will be set automatically, but if you want to set the "baz" property now, you can do so by calling:

```
newfoo->setBaz(42);
```

Note that creating the root node requires a call to the special static class method createRootElement:

```
shared_ptr<simple::root> newroot(simple::root::createRootElement());
```

Here's a complete example that ties this all together:

```
{
    opflex::modb::Mutator mutator("owner1");
    shared_ptr<simple::root> newroot(simple::root::createRootElement());
    shared_ptr<simple::root> newfoo(newroot->addFoo("test"));
    newfoo->setBaz(42);

    mutator.commit();
}
```

Update Notifications

When using the OpFlex framework, you're likely to find that most of your time is spend responding to changes in the managed object database. To get these notifications, you're going to need to register some number of listeners.

You can register an object listener to see all changes related to a particular class by calling a static function for that class. You'll then get notifications whenever any object in that class is added, updated, or deleted. The listener should queue a task to read the new state and perform appropriate processing. If this function blocks or peforms a long-running operation, then the dispatching of update notifications will be stalled, but there will not be any other deleterious effects.

If multiple changes happen to the same URI, then at least one notification will be delivered but some events may be consolidated.

The update you get will tell you the URI and the Class ID of the changed object. The class ID is a unique ID for each class. When you get the update, you'll need to call the appropriate resolve function to retrieve the new value.

You'll need to create your own object listener derived from opflex::modb::ObjectListener:

```
class MyListener : public ObjectListener {
public:
    MyListener() { }
    virtual void objectUpdated(class_id_t class_id, const URI& uri) {
        // Your handler here
```

```
    }
};
```

To register your listener with the default framework instance, just call the appropriate class static method:

```
MyListener listener;
simple::foo::registerListener(&listener);
// main loop
simple::foo::unregisterListener(&listener);
```

The listener will now recieve notifications whenever any foo or any children of any foo object changes.

Note that you must ensure that you unregister your listeners before deallocating them.

API Reference Documentation

Complete API documentation can be found by in doc/html/index.html in any build.

OVSDB Developer Guide

OVSDB Integration

The Open vSwitch database (OVSDB) Southbound Plugin component for OpenDaylight implements the OVSDB RFC 7047 management protocol that allows the southbound configuration of switches that support OVSDB. The component comprises a library and a plugin. The OVSDB protocol uses JSON-RPC calls to manipulate a physical or virtual switch that supports OVSDB. Many vendors support OVSDB on various hardware platforms. The OpenDaylight controller uses the library project to interact with an OVS instance.

Note: Read the OVSDB User Guide before you begin development.

OpenDaylight OVSDB southbound plugin architecture and design

OpenVSwitch (OVS) is generally accepted as the unofficial standard for Virtual Switching in the Open hypervisor based solutions. Every other Virtual Switch implementation, properietery or otherwise, uses OVS in some form. For information on OVS, see Open vSwitch.

In Software Defined Networking (SDN), controllers and applications interact using two channels: OpenFlow and OVSDB. OpenFlow addresses the forwarding-side of the OVS functionality. OVSDB, on the other hand, addresses the management-plane. A simple and concise overview of Open Virtual Switch Database(OVSDB) is available at: http://networkstatic.net/getting-started-ovsdb/

Overview of OpenDaylight Controller architecture

The OpenDaylight controller platform is designed as a highly modular and plugin based middleware that serves various network applications in a variety of use-cases. The modularity is achieved through the Java OSGi framework. The controller consists of many Java OSGi bundles that work together to provide the required controller functionalities.

The bundles can be placed in the following broad categories:

- Network Service Functional Modules (Examples: Topology Manager, Inventory Manager, Forwarding Rules Manager,and others)
- NorthBound API Modules (Examples: Topology APIs, Bridge Domain APIs, Neutron APIs, Connection Manager APIs, and others)
- Service Abstraction Layer(SAL)- (Inventory Services, DataPath Services, Topology Services, Network Config, and others)
- SouthBound Plugins (OpenFlow Plugin, OVSDB Plugin, OpenDove Plugin, and others)
- Application Modules (Simple Forwarding, Load Balancer)

Each layer of the Controller architecture performs specified tasks, and hence aids in modularity. While the Northbound API layer addresses all the REST-Based application needs, the SAL layer takes care of abstracting the SouthBound plugin protocol specifics from the Network Service functions.

Each of the SouthBound Plugins serves a different purpose, with some overlapping. For example, the OpenFlow plugin might serve the Data-Plane needs of an OVS element, while the OVSDB plugin can serve the management plane needs of the same OVS element. As the OpenFlow Plugin talks OpenFlow protocol with the OVS element, the OVSDB plugin will use OVSDB schema over JSON-RPC transport.

OVSDB southbound plugin

The Open vSwitch Database Management Protocol-draft-02 and Open vSwitch Manual provide theoretical information about OVSDB. The OVSDB protocol draft is generic enough to lay the groundwork on Wire Protocol and Database Operations, and the OVS Manual currently covers 13 tables leaving space for future OVS expansion, and vendor expansions on proprietary implementations. The OVSDB Protocol is a database records transport protocol using JSON RPC1.0. For information on the protocol structure, see Getting Started with OVSDB. The OpenDaylight OVSDB southbound plugin consists of one or more OSGi bundles addressing the following services or functionalities:

- Connection Service - Based on Netty
- Network Configuration Service
- Bidirectional JSON-RPC Library
- OVSDB Schema definitions and Object mappers
- Overlay Tunnel management
- OVSDB to OpenFlow plugin mapping service
- Inventory Service

Connection service

One of the primary services that most southbound plugins provide in OpenDaylight a Connection Service. The service provides protocol specific connectivity to network elements, and supports the connectivity management services as specified by the OpenDaylight Connection Manager. The connectivity services include:

- Connection to a specified element given IP-address, L4-port, and other connectivity options (such as authentication,. . .)
- Disconnection from an element
- Handling Cluster Mode change notifications to support the OpenDaylight Clustering/High-Availability feature

Network Configuration Service

The goal of the OpenDaylight Network Configuration services is to provide complete management plane solutions needed to successfully install, configure, and deploy the various SDN based network services. These are generic services which can be implemented in part or full by any south-bound protocol plugin. The south-bound plugins can be either of the following:

- The new network virtualization protocol plugins such as OVSDB JSON-RPC
- The traditional management protocols such as SNMP or any others in the middle.

The above definition, and more information on Network Configuration Services, is available at : https://wiki.opendaylight.org/view/OpenDaylight_Controller:NetworkConfigurationServices

Bidirectional JSON-RPC library

The OVSDB plugin implements a Bidirectional JSON-RPC library. It is easy to design the library as a module that manages the Netty connection towards the Element.

The main responsibilities of this Library are:

- Demarshal and marshal JSON Strings to JSON objects
- Demarshal and marshal JSON Strings from and to the Network Element.

OVSDB Schema definitions and Object mappers

The OVSDB Schema definitions and Object Mapping layer sits above the JSON-RPC library. It maps the generic JSON objects to OVSDB schema POJOs (Plain Old Java Object) and vice-versa. This layer mostly provides the Java Object definition for the corresponding OVSDB schema (13 of them) and also will provide much more friendly API abstractions on top of these object data. This helps in hiding the JSON semantics from the functional modules such as Configuration Service and Tunnel management.

On the demarshaling side the mapping logic differentiates the Request and Response messages as follows :

- Request messages are mapped by its “method”
- Response messages are mapped by their IDs which were originally populated by the Request message. The JSON semantics of these OVSDB schema is quite complex. The following figures summarize two of the end-to-end scenarios:

Overlay tunnel management

Network Virtualization using OVS is achieved through Overlay Tunnels. The actual Type of the Tunnel may be GRE, VXLAN, or STT. The differences in the encapsulation and configuration decide the tunnel types. Establishing a tunnel using configuration service requires just the sending of OVSDB messages towards the ovsdb-server. However, the scaling issues that would arise on the state management at the data-plane (using OpenFlow) can get challenging. Also, this module can assist in various optimizations in the presence of Gateways. It can also help in providing Service guarantees for the VMs using these overlays with the help of underlay orchestration.

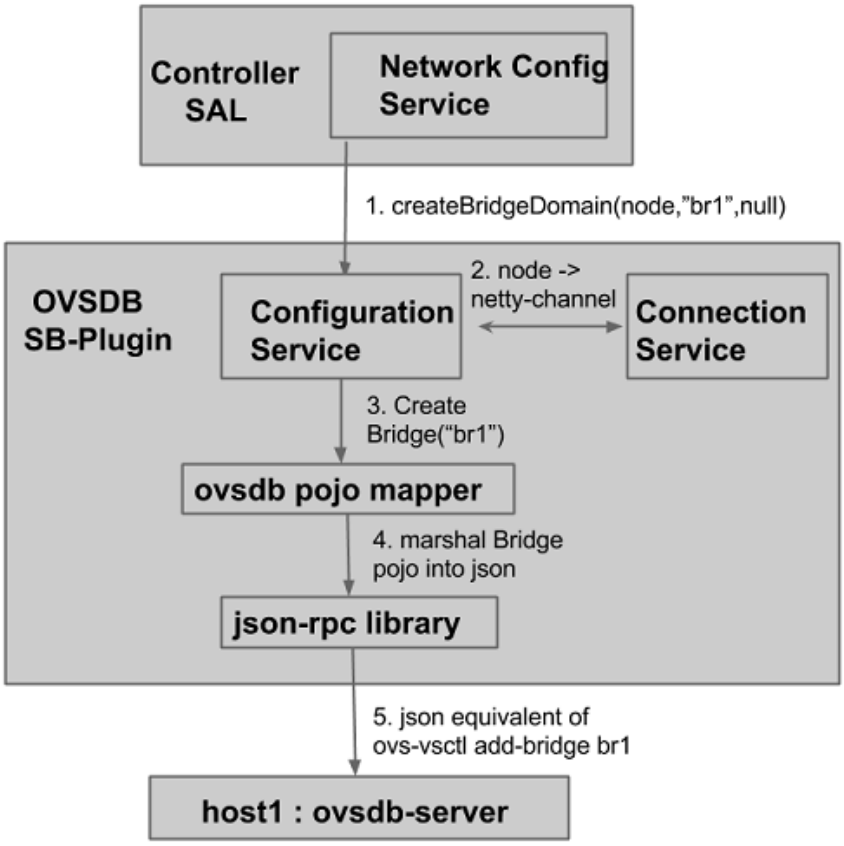


Fig. 2.44: End-to-end handling of a Create Bridge request

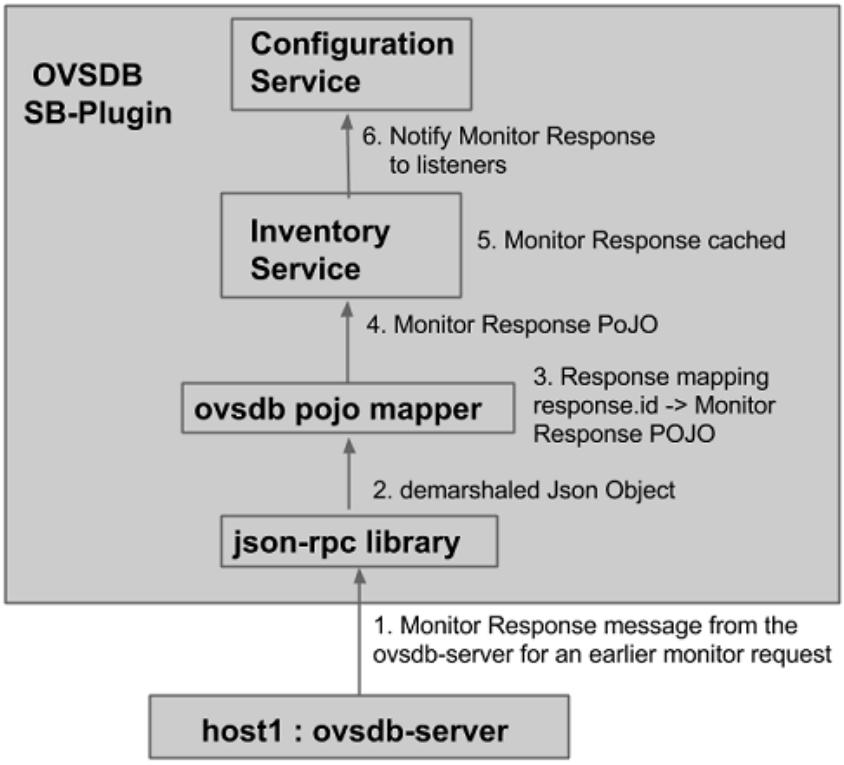


Fig. 2.45: End-to-end handling of a monitor response

OVSDB to OpenFlow plugin mapping service

The connect() of the ConnectionService would result in a Node that represents an ovsdb-server. The CreateBridgeDomain() Configuration on the above Node would result in creating an OVS bridge. This OVS Bridge is an OpenFlow Agent for the OpenDaylight OpenFlow plugin with its own Node represented as (example) OF|xxxx.yyyy.zzzz. Without any help from the OVSDB plugin, the Node Mapping Service of the Controller platform would not be able to map the following:

```
{OVSDB_NODE + BRIDGE_IDENTFIER} <---> {OF_NODE}.
```

Without such mapping, it would be extremely difficult for the applications to manage and maintain such nodes. This Mapping Service provided by the OVSDB plugin would essentially help in providing more value added services to the orchestration layers that sit atop the Northbound APIs (such as OpenStack).

OVSDB: New features

Schema independent library

The OVS connection is a node which can have multiple databases. Each database is represented by a schema. A single connection can have multiple schemas. OSVDB supports multiple schemas. Currently, these are two schemas available in the OVSDB, but there is no restriction on the number of schemas. Owing to the Northbound v3 API, no code changes in ODL are needed for supporting additional schemas.

Schemas:

- openvswitch : Schema wrapper that represents http://openvswitch.org/ovs-vswitchd.conf.db.5.pdf
- hardwarevtep: Schema wrapper that represents http://openvswitch.org/docs/vtep.5.pdf

OVSDB Library Developer Guide

Overview

The OVSDB library manages the Netty connections to network nodes and handles bidirectional JSON-RPC messages. It not only provides OVSDB protocol functionality to OpenDaylight OVSDB plugin but also can be used as standalone JAVA library for OVSDB protocol.

The main responsibilities of OVSDB library include:

- Manage connections to peers
- Marshal and unmarshal JSON Strings to JSON objects.
- Marshal and unmarshal JSON Strings from and to the Network Element.

Connection Service

The OVSDB library provides connection management through the OvsdbConnection interface. The OvsdbConnection interface provides OVSDB connection management APIs which include both active and passive connections. From the library perspective, active OVSDB connections are initiated from the controller to OVS nodes while passive OVSDB

connections are initiated from OVS nodes to the controller. In the active connection scenario an application needs to provide the IP address and listening port of OVS nodes to the library management API. On the other hand, the library management API only requires the info of the controller listening port in the passive connection scenario.

For a passive connection scenario, the library also provides a connection event listener through the OvsdbConnectionListener interface. The listener interface has connected() and disconnected() methods to notify an application when a new passive connection is established or an existing connection is terminated.

SSL Connection

In addition to a regular TCP connection, the OvsdbConnection interface also provides a connection management API for an SSL connection. To start an OVSDB connection with SSL, an application will need to provide a Java SSLContext object to the management API. There are different ways to create a Java SSLContext, but in most cases a Java KeyStore with certificate and private key provided by the application is required. Detailed steps about how to create a Java SSLContext is out of the scope of this document and can be found in the Java documentation for JAVA Class SSlContext.

In the active connection scenario, the library uses the given SSLContext to create a Java SSLEngine and configures the SSL engine with the client mode for SSL handshaking. Normally clients are not required to authenticate themselves.

In the passive connection scenario, the library uses the given SSLContext to create a Java SSLEngine which will operate in server mode for SSL handshaking. For security reasons, the SSLv3 protocol and some cipher suites are disabled. Currently the OVSDB server only supports the TLS_RSA_WITH_AES_128_CBC_SHA cipher suite and the following protocols: SSLv2Hello, TLSv1, TLSv1.1, TLSv1.2.

The SSL engine is also configured to operate on two-way authentication mode for passive connection scenarios, i.e., the OVSDB server (controller) will authenticate clients (OVS nodes) and clients (OVS nodes) are also required to authenticate the server (controller). In the two-way authentication mode, an application should keep a trust manager to store the certificates of trusted clients and initialize a Java SSLContext with this trust manager. Thus during the SSL handshaking process the OVSDB server (controller) can use the trust manager to verify clients and only accept connection requests from trusted clients. On the other hand, users should also configure OVS nodes to authenticate the controller. Open vSwitch already supports this functionality in the ovsdb-server command with option `--ca-cert=cacert.pem` and `--bootstrap-ca-cert=cacert.pem`. On the OVS node, a user can use the option `--ca-cert=cacert.pem` to specify a controller certificate directly and the node will only allow connections to the controller with the specified certificate. If the OVS node runs ovsdb-server with option `--bootstrap-ca-cert=cacert.pem`, it will authenticate the controller with the specified certificate cacert.pem. If the certificate file doesn't exist, it will attempt to obtain a certificate from the peer (controller) on its first SSL connection and save it to the named PEM file `cacert.pem`. Here is an example of ovsdb-server with `--bootstrap-ca-cert=cacert.pem` option:

```
ovsdb-server --pidfile --detach --log-file --remote punix:/var/run/
openvswitch/db.sock --remote=db:hardware_vtep,Global,managers --private-key=/
etc/openvswitch/ovsclient-privkey.pem -- certificate=/etc/openvswitch/
ovsclient-cert.pem --bootstrap-ca-cert=/etc/openvswitch/vswitchd.cacert
```

OVSDB protocol transactions

The OVSDB protocol defines the RPC transaction methods in RFC 7047. The following RPC methods are supported in OVSDB protocol:

- List databases
- Get schema
- Transact

- Cancel
- Monitor
- Update notification
- Monitor cancellation
- Lock operations
- Locked notification
- Stolen notification
- Echo

According to RFC 7047, an OVSDB server must implement all methods, and an OVSDB client is only required to implement the "Echo" method and otherwise free to implement whichever methods suit its needs. However, the OVSDB library currently doesn't support all RPC methods. For the "Echo" method, the library can handle "Echo" messages from a peer and send a JSON response message back, but the library doesn't support actively sending an "Echo" JSON request to a peer. Other unsupported RPC methods are listed below:

- Cancel
- Lock operations
- Locked notification
- Stolen notification

In the OVSDB library the RPC methods are defined in the Java interface OvsdbRPC. The library also provides a high-level interface OvsdbClient as the main interface to interact with peers through the OVSDB protocol. In the passive connection scenario, each connection will have a corresponding OvsdbClient object, and the application can obtain the OvsdbClient object through connection listener callback methods. In other words, if the application implements the OvsdbConnectionListener interface, it will get notifications of connection status changes with the corresponding OvsdbClient object of that connection.

OVSDB database operations

RFC 7047 also defines database operations, such as insert, delete, and update, to be performed as part of a "transact" RPC request. The OVSDB library defines the data operations in Operations.java and provides the TransactionBuilder class to help build "transact" RPC requests. To build a JSON-RPC transact request message, the application can obtain the TransactionBuilder object through a transactBuilder() method in the OvsdbClient interface.

The TransactionBuilder class provides the following methods to help build transactions:

- getOperations(): Get the list of operations in this transaction.
- add(): Add data operation to this transaction.
- build(): Return the list of operations in this transaction. This is the same as the getOperations() method.
- execute(): Send the JSON RPC transaction to peer.
- getDatabaseSchema(): Get the database schema of this transaction.

If the application wants to build and send a "transact" RPC request to modify OVSDB tables on a peer, it can take the following steps:

1. Statically import parameter "op" in Operations.java

```
import static org.opendaylight.ovsdb.lib.operations.Operations.op;
```

2. Obtain transaction builder through transacBuilder() method in OvsdbClient:

```
TransactionBuilder transactionBuilder = ovsdbClient.transactionBuilder(dbSchema);
```

3. Add operations to transaction builder:

```
transactionBuilder.add(op.insert(schema, row));
```

4. Send transaction to peer and get JSON RPC response:

```
operationResults = transactionBuilder.execute().get();
```

Note: Although the "select" operation is supported in the OVSDB library, the library implementation is a little different from RFC 7047. In RFC 7047, section 5.2.2 describes the "select" operation as follows:

"The "rows" member of the result is an array of objects. Each object corresponds to a matching row, with each column specified in "columns" as a member, the column's name as the member name, and its value as the member value. If "columns" is not specified, all the table's columns are included (including the internally generated "_uuid" and "_version" columns)."

The OVSDB library implementation always requires the column's name in the "columns" field of a JSON message. If the "columns" field is not specified, none of the table's columns are included. If the application wants to get the table entry with all columns, it needs to specify all the columns' names in the "columns" field.

Reference Documentation

RFC 7047 The Open vSwitch Databse Management Protocol https://tools.ietf.org/html/rfc7047

OVSDB MD-SAL Southbound Plugin Developer Guide

Overview

The Open vSwitch Database (OVSDB) Model Driven Service Abstraction Layer (MD-SAL) Southbound Plugin provides an MD-SAL based interface to Open vSwitch systems. This is done by augmenting the MD-SAL topology node with a YANG model which replicates some (but not all) of the Open vSwitch schema.

OVSDB MD-SAL Southbound Plugin Architecture and Operation

The architecture and operation of the OVSDB MD-SAL Southbound plugin is illustrated in the following set of diagrams.

Connecting to an OVSDB Node

An OVSDB node is a system which is running the OVS software and is capable of being managed by an OVSDB manager. The OVSDB MD-SAL Southbound plugin in OpenDaylight is capable of operating as an OVSDB manager. Depending on the configuration of the OVSDB node, the connection of the OVSDB manager can be active or passive.

Active OVSDB Node Manager Workflow

An active OVSDB node manager connection is made when OpenDaylight initiates the connection to the OVSDB node. In order for this to work, you must configure the OVSDB node to listen on a TCP port for the connection (i.e. OpenDaylight is active and the OVSDB node is passive). This option can be configured on the OVSDB node using the following command:

```
ovs-vsctl set-manager ptcp:6640
```

The following diagram illustrates the sequence of events which occur when OpenDaylight initiates an active OVSDB manager connection to an OVSDB node.

Step 1 Create an OVSDB node by using RESTCONF or an OpenDaylight plugin. The OVSDB node is listed under the OVSDB topology node.

Step 2 Add the OVSDB node to the OVSDB MD-SAL southbound configuration datastore. The OVSDB southbound provider is registered to listen for data change events on the portion of the MD-SAL topology data store which contains the OVSDB southbound topology node augmentations. The addition of an OVSDB node causes an event which is received by the OVSDB Southbound provider.

Step 3 The OVSDB Southbound provider initiates a connection to the OVSDB node using the connection information provided in the configuration OVSDB node (i.e. IP address and TCP port number).

Step 4 The OVSDB Southbound provider adds the OVSDB node to the OVSDB MD-SAL operational data store. The operational data store contains OVSDB node objects which represent active connections to OVSDB nodes.

Step 5 The OVSDB Southbound provider requests the schema and databases which are supported by the OVSDB node.

Step 6 The OVSDB Southbound provider uses the database and schema information to construct a monitor request which causes the OVSDB node to send the controller any updates made to the OVSDB databases on the OVSDB node.

Passive OVSDB Node Manager Workflow

A passive OVSDB node connection to OpenDaylight is made when the OVSDB node initiates the connection to OpenDaylight. In order for this to work, you must configure the OVSDB node to connect to the IP address and OVSDB port on which OpenDaylight is listening. This option can be configured on the OVSDB node using the following command:

```
ovs-vsctl set-manager tcp:<IP address>:6640
```

The following diagram illustrates the sequence of events which occur when an OVSDB node connects to OpenDaylight.

Step 1 The OVSDB node initiates a connection to OpenDaylight.

Step 2 The OVSDB Southbound provider adds the OVSDB node to the OVSDB MD-SAL operational data store. The operational data store contains OVSDB node objects which represent active connections to OVSDB nodes.

Step 3 The OVSDB Southbound provider requests the schema and databases which are supported by the OVSDB node.

Step 4 The OVSDB Southbound provider uses the database and schema information to construct a monitor request which causes the OVSDB node to send back any updates which have been made to the OVSDB databases on the OVSDB node.

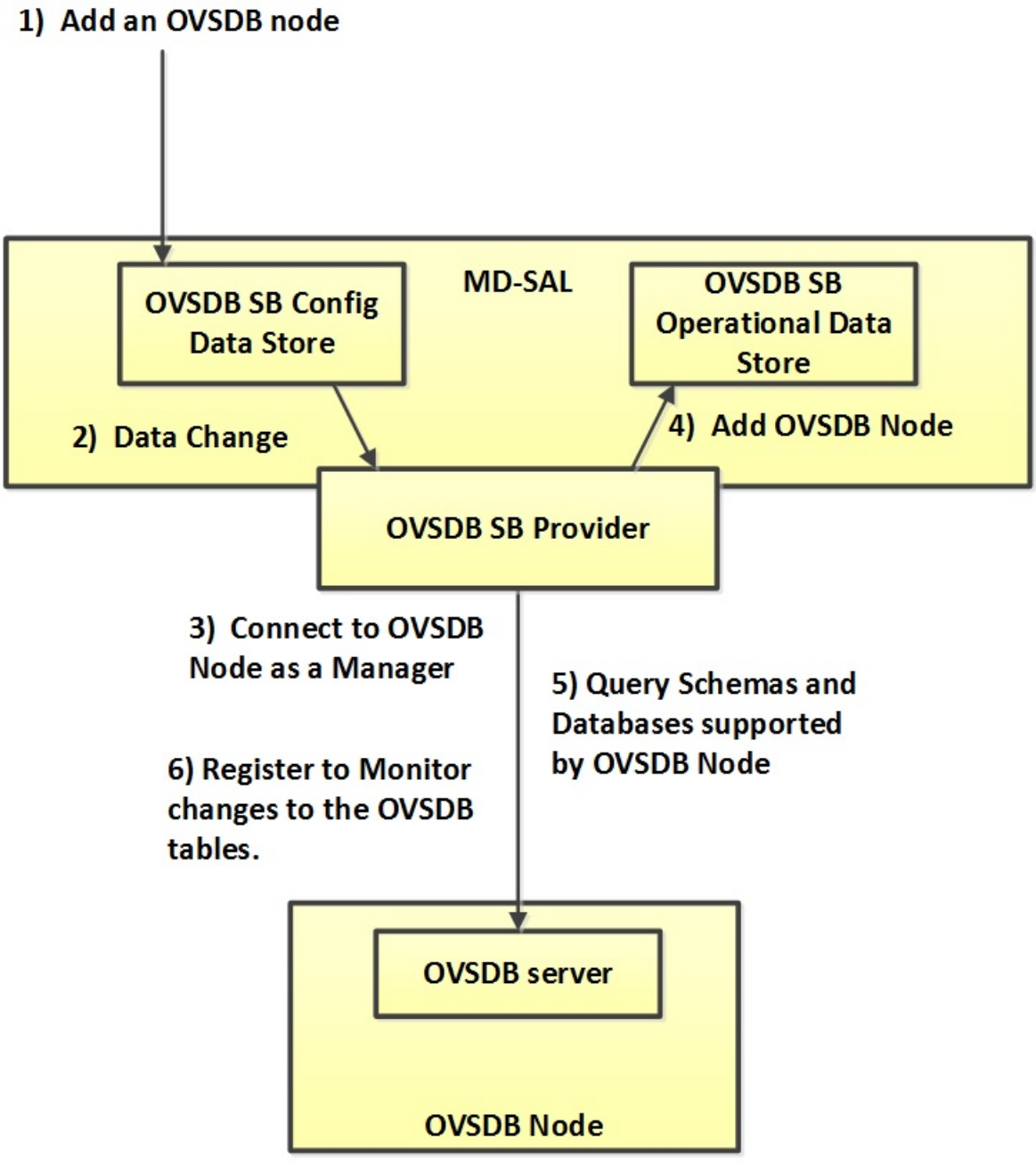


Fig. 2.46: Active OVSDB Manager Connection

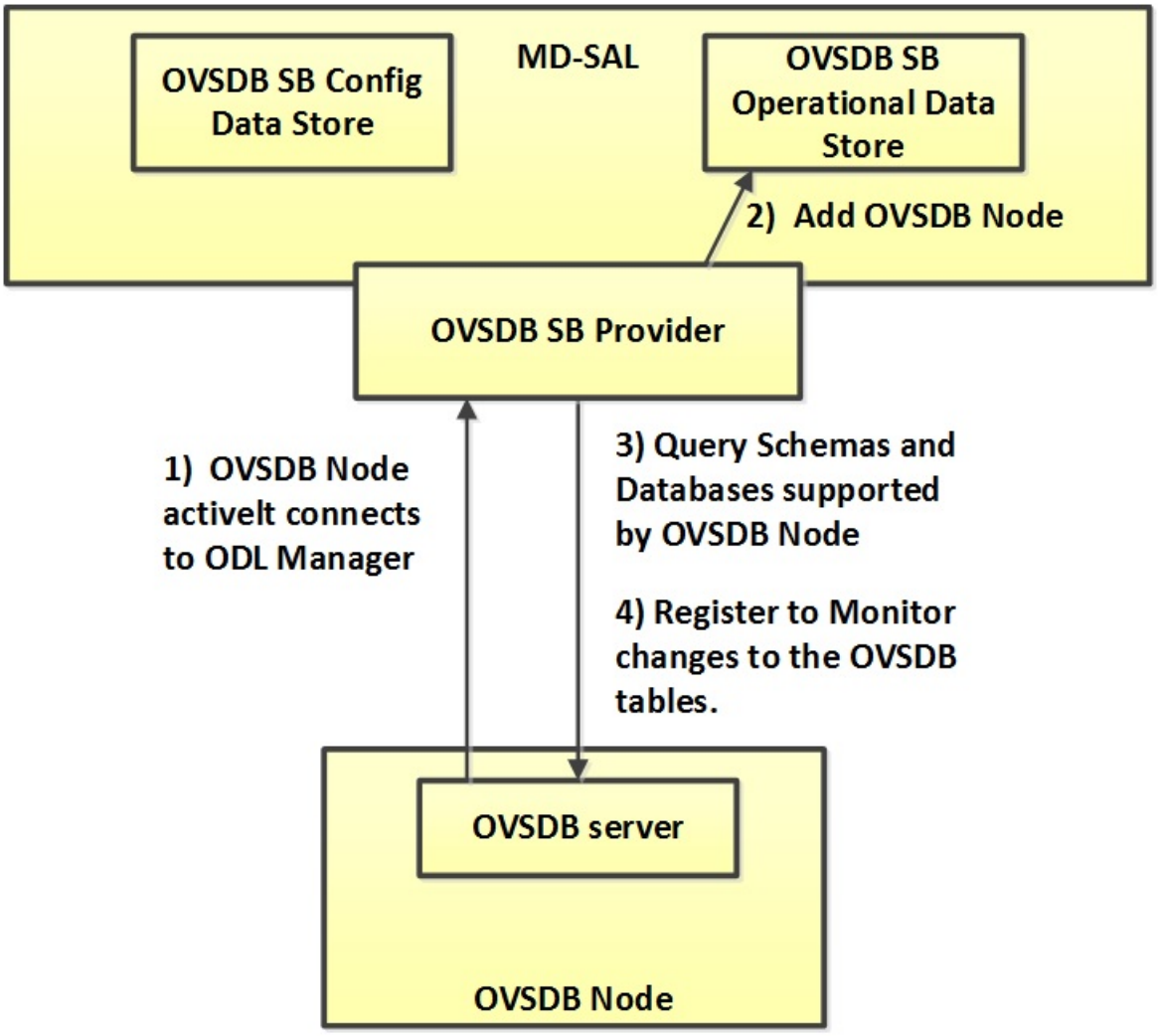


Fig. 2.47: Passive OVSDB Manager Connection

OVSDB Node ID in the Southbound Operational MD-SAL

When OpenDaylight initiates an active connection to an OVSDB node, it writes an external-id to the Open_vSwitch table on the OVSDB node. The external-id is an OpenDaylight instance identifier which identifies the OVSDB topology node which has just been created. Here is an example showing the value of the *opendaylight-iid* entry in the external-ids column of the Open_vSwitch table where the node-id of the OVSDB node is *ovsdb:HOST1*.

```
$ ovs-vsctl list open_vswitch
...
external_ids        : {opendaylight-iid="/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb:HOST1']"}
...
```

The *opendaylight-iid* entry in the external-ids column of the Open_vSwitch table causes the OVSDB node to have same node-id in the operational MD-SAL datastore as in the configuration MD-SAL datastore. This holds true if the OVSDB node manager settings are subsequently changed so that a passive OVSDB manager connection is made.

If there is no *opendaylight-iid* entry in the external-ids column and a passive OVSDB manager connection is made, then the node-id of the OVSDB node in the operational MD-SAL datastore will be constructed using the UUID of the Open_vSwitch table as follows.

```
"node-id": "ovsdb://uuid/b8dc0bfb-d22b-4938-a2e8-b0084d7bd8c1"
```

The *opendaylight-iid* entry can be removed from the Open_vSwitch table using the following command.

```
$ sudo ovs-vsctl remove open_vswitch . external-id "opendaylight-iid"
```

OVSDB Changes by using OVSDB Southbound Config MD-SAL

After the connection has been made to an OVSDB node, you can make changes to the OVSDB node by using the OVSDB Southbound Config MD-SAL. You can make CRUD operations by using the RESTCONF interface or by a plugin using the MD-SAL APIs. The following diagram illustrates the high-level flow of events.

Step 1 A change to the OVSDB Southbound Config MD-SAL is made. Changes include adding or deleting bridges and ports, or setting attributes of OVSDB nodes, bridges or ports.

Step 2 The OVSDB Southbound provider receives notification of the changes made to the OVSDB Southbound Config MD-SAL data store.

Step 3 As appropriate, OVSDB transactions are constructed and transmitted to the OVSDB node to update the OVSDB database on the OVSDB node.

Step 4 The OVSDB node sends update messages to the OVSDB Southbound provider to indicate the changes made to the OVSDB nodes database.

Step 5 The OVSDB Southbound provider maps the changes received from the OVSDB node into corresponding changes made to the OVSDB Southbound Operational MD-SAL data store.

Detecting changes in OVSDB coming from outside OpenDaylight

Changes to the OVSDB nodes database may also occur independently of OpenDaylight. OpenDaylight also receives notifications for these events and updates the Southbound operational MD-SAL. The following diagram illustrates the sequence of events.

Step 1 Changes are made to the OVSDB node outside of OpenDaylight (e.g. ovs-vsctl).

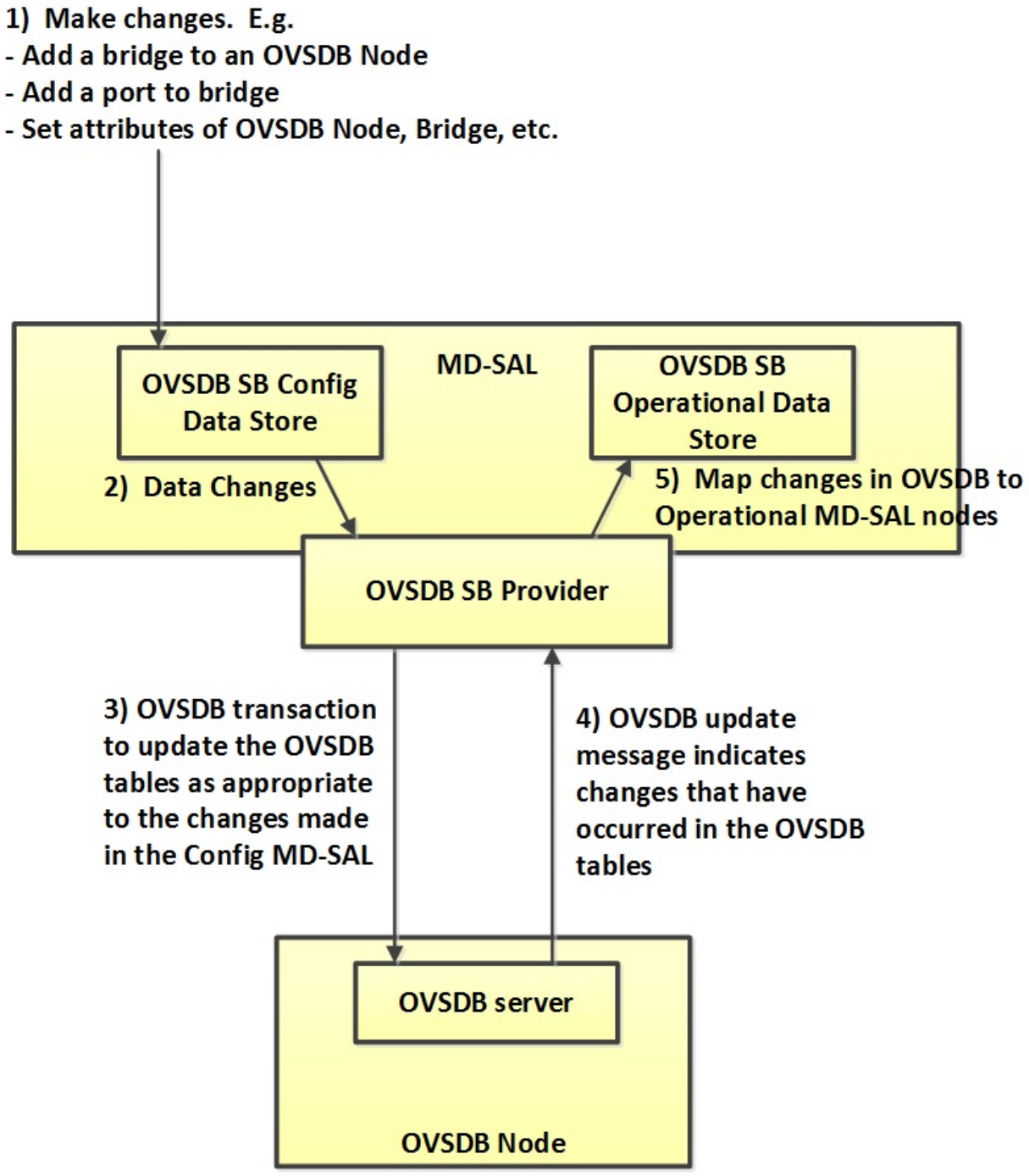


Fig. 2.48: OVSDB Changes by using the Southbound Config MD-SAL

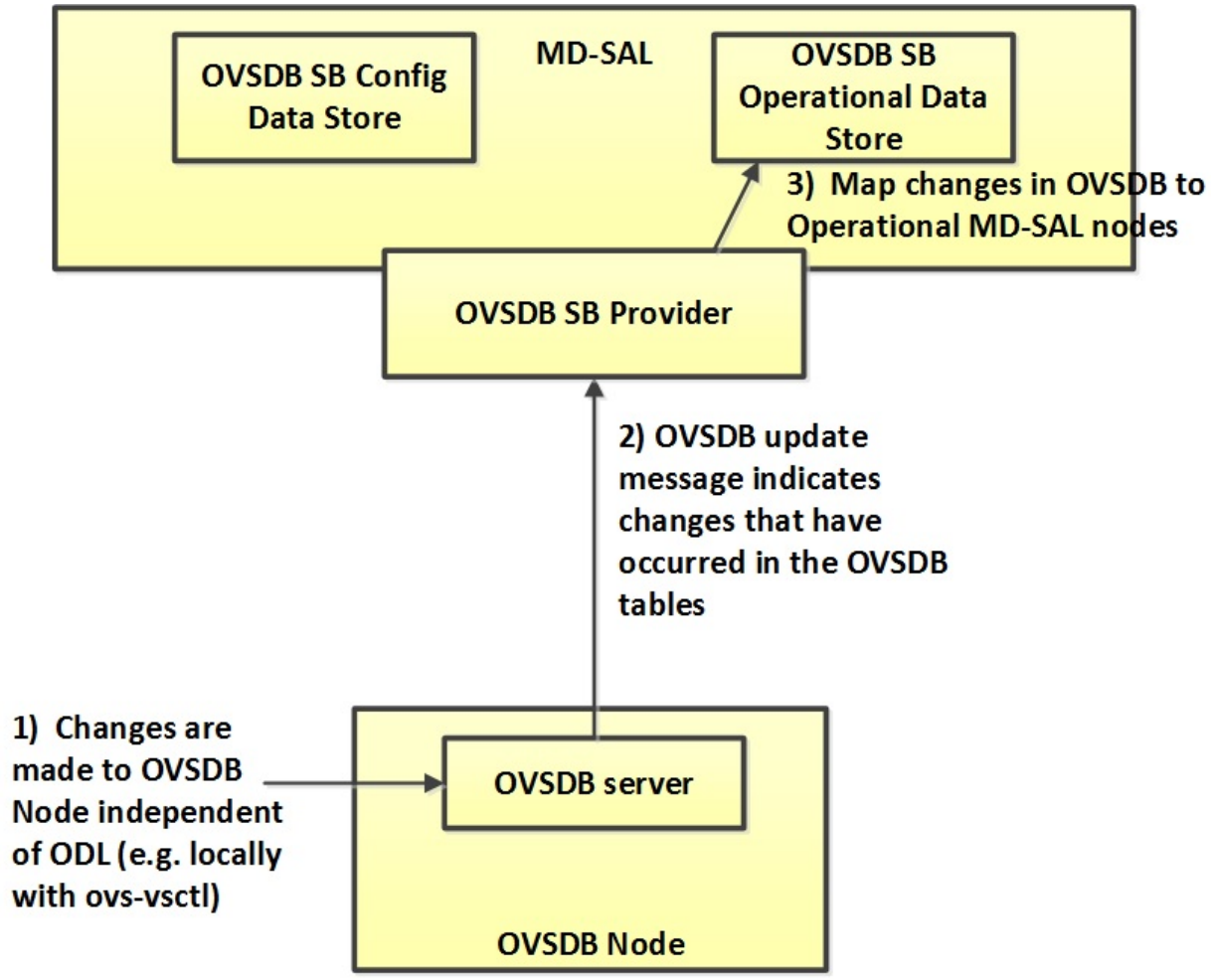


Fig. 2.49: OVSDB Changes made directly on the OVSDB node

Step 2 The OVSDB node constructs update messages to inform OpenDaylight of the changes made to its databases.

Step 3 The OVSDB Southbound provider maps the OVSDB database changes to corresponding changes in the OVSDB Southbound operational MD-SAL data store.

OVSDB Model

The OVSDB Southbound MD-SAL operates using a YANG model which is based on the abstract topology node model found in the network topology model.

The augmentations for the OVSDB Southbound MD-SAL are defined in the ovsdb.yang file.

There are three augmentations:

ovsdb-node-augmentation This augments the topology node and maps primarily to the Open_vSwitch table of the OVSDB schema. It contains the following attributes.

- **connection-info** - holds the local and remote IP address and TCP port numbers for the OpenDaylight to OVSDB node connections
- **db-version** - version of the OVSDB database
- **ovs-version** - version of OVS
- **list managed-node-entry** - a list of references to ovsdb-bridge-augmentation nodes, which are the OVS bridges managed by this OVSDB node
- **list datapath-type-entry** - a list of the datapath types supported by the OVSDB node (e.g. *system*, *netdev*) - depends on newer OVS versions
- **list interface-type-entry** - a list of the interface types supported by the OVSDB node (e.g. *internal*, *vxlan*, *gre*, *dpdk*, etc.) - depends on newer OVS verions
- **list openvswitch-external-ids** - a list of the key/value pairs in the Open_vSwitch table external_ids column
- **list openvswitch-other-config** - a list of the key/value pairs in the Open_vSwitch table other_config column

ovsdb-bridge-augmentation This augments the topology node and maps to an specific bridge in the OVSDB bridge table of the associated OVSDB node. It contains the following attributes.

- **bridge-uuid** - UUID of the OVSDB bridge
- **bridge-name** - name of the OVSDB bridge
- **bridge-openflow-node-ref** - a reference (instance-identifier) of the OpenFlow node associated with this bridge
- **list protocol-entry** - the version of OpenFlow protocol to use with the OpenFlow controller
- **list controller-entry** - a list of controller-uuid and is-connected status of the OpenFlow controllers associated with this bridge
- **datapath-id** - the datapath ID associated with this bridge on the OVSDB node
- **datapath-type** - the datapath type of this bridge
- **fail-mode** - the OVSDB fail mode setting of this bridge
- **flow-node** - a reference to the flow node corresponding to this bridge
- **managed-by** - a reference to the ovsdb-node-augmentation (OVSDB node) that is managing this bridge
- **list bridge-external-ids** - a list of the key/value pairs in the bridge table external_ids column for this bridge

- **list bridge-other-configs** - a list of the key/value pairs in the bridge table other_config column for this bridge

ovsdb-termination-point-augmentation This augments the topology termination point model. The OVSDB Southbound MD-SAL uses this model to represent both the OVSDB port and OVSDB interface for a given port/interface in the OVSDB schema. It contains the following attributes.

- **port-uuid** - UUID of an OVSDB port row
- **interface-uuid** - UUID of an OVSDB interface row
- **name** - name of the port
- **interface-type** - the interface type
- **list options** - a list of port options
- **ofport** - the OpenFlow port number of the interface
- **ofport_request** - the requested OpenFlow port number for the interface
- **vlan-tag** - the VLAN tag value
- **list trunks** - list of VLAN tag values for trunk mode
- **vlan-mode** - the VLAN mode (e.g. access, native-tagged, native-untagged, trunk)
- **list port-external-ids** - a list of the key/value pairs in the port table external_ids column for this port
- **list interface-external-ids** - a list of the key/value pairs in the interface table external_ids interface for this interface
- **list port-other-configs** - a list of the key/value pairs in the port table other_config column for this port
- **list interface-other-configs** - a list of the key/value pairs in the interface table other_config column for this interface

Examples of OVSDB Southbound MD-SAL API

Connect to an OVSDB Node

This example RESTCONF command adds an OVSDB node object to the OVSDB Southbound configuration data store and attempts to connect to the OVSDB host located at the IP address 10.11.12.1 on TCP port 6640.

```
POST http://<host>:8181/restconf/config/network-topology:network-topology/topology/
↪ovsdb:1/
Content-Type: application/json
{
  "node": [
     {
       "node-id": "ovsdb:HOST1",
       "connection-info": {
         "ovsdb:remote-ip": "10.11.12.1",
         "ovsdb:remote-port": 6640
       }
     }
  ]
}
```

Query the OVSDB Southbound Configuration MD-SAL

Following on from the previous example, if the OVSDB Southbound configuration MD-SAL is queried, the REST-CONF command and the resulting reply is similar to the following example.

```
GET http://<host>:8080/restconf/config/network-topology:network-topology/topology/
↪ovsdb:1/
Application/json data in the reply
{
  "topology": [
    {
      "topology-id": "ovsdb:1",
      "node": [
        {
          "node-id": "ovsdb:HOST1",
          "ovsdb:connection-info": {
            "remote-port": 6640,
            "remote-ip": "10.11.12.1"
          }
        }
      ]
    }
  ]
}
```

Reference Documentation

Openvswitch schema

OVSDB Hardware VTEP Developer Guide

Overview

TBD

OVSDB Hardware VTEP Architecture

TBD

PCEP Developer Guide

Overview

This section provides an overview of **feature odl-bgpcep-pcep-all** . This feature will install everything needed for PCEP (Path Computation Element Protocol) including establishing the connection, storing information about LSPs (Label Switched Paths) and displaying data in network-topology overview.

PCEP Architecture

Each feature represents a module in the BGPCEP codebase. The following diagram illustrates how the features are related.

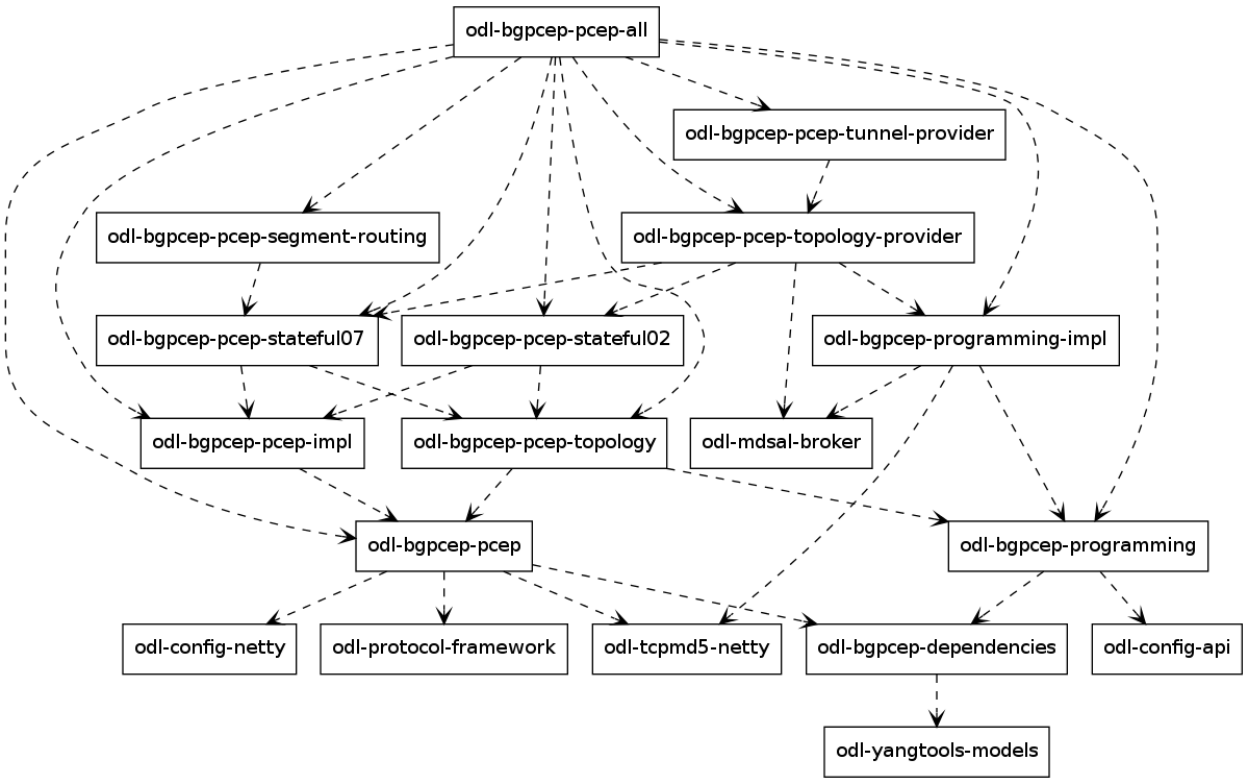


Fig. 2.50: PCEP Dependency Tree

Key APIs and Interfaces

PCEP

Session handling

32-pcep.xml defines only pcep-dispatcher the parser should be using (global-pcep-extensions), factory for creating session proposals (you can create different proposals for different PCCs (Path Computation Clients)).

```
<module>
 <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:pcep:impl">
↪prefix:pcep-dispatcher-impl</type>
 <name>global-pcep-dispatcher</name>
 <pcep-extensions>
  <type xmlns:pcepspi="urn:opendaylight:params:xml:ns:yang:controller:pcep:spi">
↪pcepspi:extensions</type>
  <name>global-pcep-extensions</name>
 </pcep-extensions>
 <pcep-session-proposal-factory>
```

```
  <type xmlns:pcep="urn:opendaylight:params:xml:ns:yang:controller:pcep">pcep:pcep-
↪session-proposal-factory</type>
  <name>global-pcep-session-proposal-factory</name>
 </pcep-session-proposal-factory>
 <boss-group>
  <type xmlns:netty="urn:opendaylight:params:xml:ns:yang:controller:netty">
↪netty:netty-threadgroup</type>
  <name>global-boss-group</name>
 </boss-group>
 <worker-group>
  <type xmlns:netty="urn:opendaylight:params:xml:ns:yang:controller:netty">
↪netty:netty-threadgroup</type>
  <name>global-worker-group</name>
 </worker-group>
</module>
```

For user configuration of PCEP, check User Guide.

Parser

The base PCEP parser includes messages and attributes from RFC5441, RFC5541, RFC5455, RFC5557 and RFC5521.

Registration

All parsers and serializers need to be registered into *Extension provider*. This *Extension provider* is configured in initial configuration of the parser-spi module (*32-pcep.xml*).

```
<module>
 <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:pcep:spi">
↪prefix:pcep-extensions-impl</type>
 <name>global-pcep-extensions</name>
 <extension>
  <type xmlns:pcepspi="urn:opendaylight:params:xml:ns:yang:controller:pcep:spi">
↪pcepspi:extension</type>
  <name>pcep-parser-base</name>
 </extension>
 <extension>
  <type xmlns:pcepspi="urn:opendaylight:params:xml:ns:yang:controller:pcep:spi">
↪pcepspi:extension</type>
  <name>pcep-parser-ietf-stateful07</name>
 </extension>
 <extension>
  <type xmlns:pcepspi="urn:opendaylight:params:xml:ns:yang:controller:pcep:spi">
↪pcepspi:extension</type>
  <name>pcep-parser-ietf-initiated00</name>
 </extension>
 <extension>
  <type xmlns:pcepspi="urn:opendaylight:params:xml:ns:yang:controller:pcep:spi">
↪pcepspi:extension</type>
  <name>pcep-parser-sync-optimizations</name>
 </extension>
</module>
```

- *pcep-parser-base* - will register parsers and serializers implemented in pcep-impl module

- *pcep-parser-ietf-stateful07* - will register parsers and serializers of draft-ietf-pce-stateful-pce-07 implementation
- *pcep-parser-ietf-initiated00* - will register parser and serializer of draft-ietf-pce-pce-initiated-lsp-00 implementation
- *pcep-parser-sync-optimizations* - will register parser and serializers of draft-ietf-pce-stateful-sync-optimizations-03 implementation

Stateful07 module is a good example of a PCEP parser extension.

Configuration of PCEP parsers specifies one implementation of *Extension provider* that will take care of registering mentioned parser extensions: SimplePCEPExtensionProviderContext. All registries are implemented in package pcep-spi.

Parsing

Parsing of PCEP elements is mostly done equally to BGP, the only exception is message parsing, that is described here.

In BGP messages, parsing of first-level elements (path-attributes) can be validated in a simple way, as the attributes should be ordered chronologically. PCEP, on the other hand, has a strict object order policy, that is described in RBNF (Routing Backus-Naur Form) in each RFC. Therefore the algorithm for parsing here is to parse all objects in order as they appear in the message. The result of parsing is a list of *PCEPObjects*, that is put through validation. *validate()* methods are present in each message parser. Depending on the complexity of the message, it can contain either a simple condition (checking the presence of a mandatory object) or a full state machine.

In addition to that, PCEP requires sending error message for each documented parsing error. This is handled by creating an empty list of messages *errors* which is then passed as argument throughout whole parsing process. If some parser encounters *PCEPDocumentedException*, it has the duty to create appropriate PCEP error message and add it to this list. In the end, when the parsing is finished, this list is examined and all messages are sent to peer.

Better understanding provides this sequence diagram:

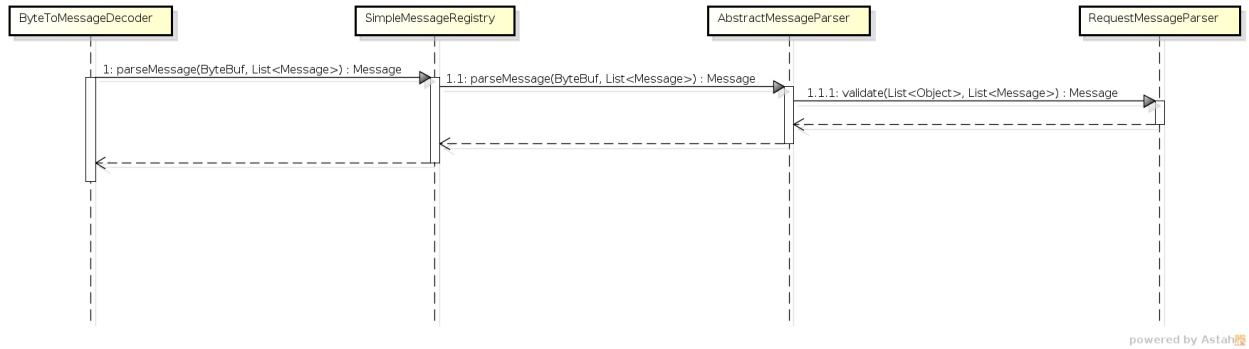


Fig. 2.51: Parsing

PCEP IETF stateful

This section summarizes module pcep-ietf-stateful07. The term *stateful* refers to draft-ietf-pce-stateful-pce and draft-ietf-pce-pce-initiated-lsp in versions draft-ietf-pce-stateful-pce-07 with draft-ietf-pce-pce-initiated-lsp-00.

We will upgrade our implementation, when the stateful draft gets promoted to RFC.

The stateful module is implemented as extensions to pcep-base-parser. The stateful draft declared new elements as well as additional fields or TLVs (type,length,value) to known objects. All new elements are defined in yang models,

that contain augmentations to elements defined in pcep-types.yang. In the case of extending known elements, the *Parser* class merely extends the base class and overrides necessary methods as shown in following diagram:

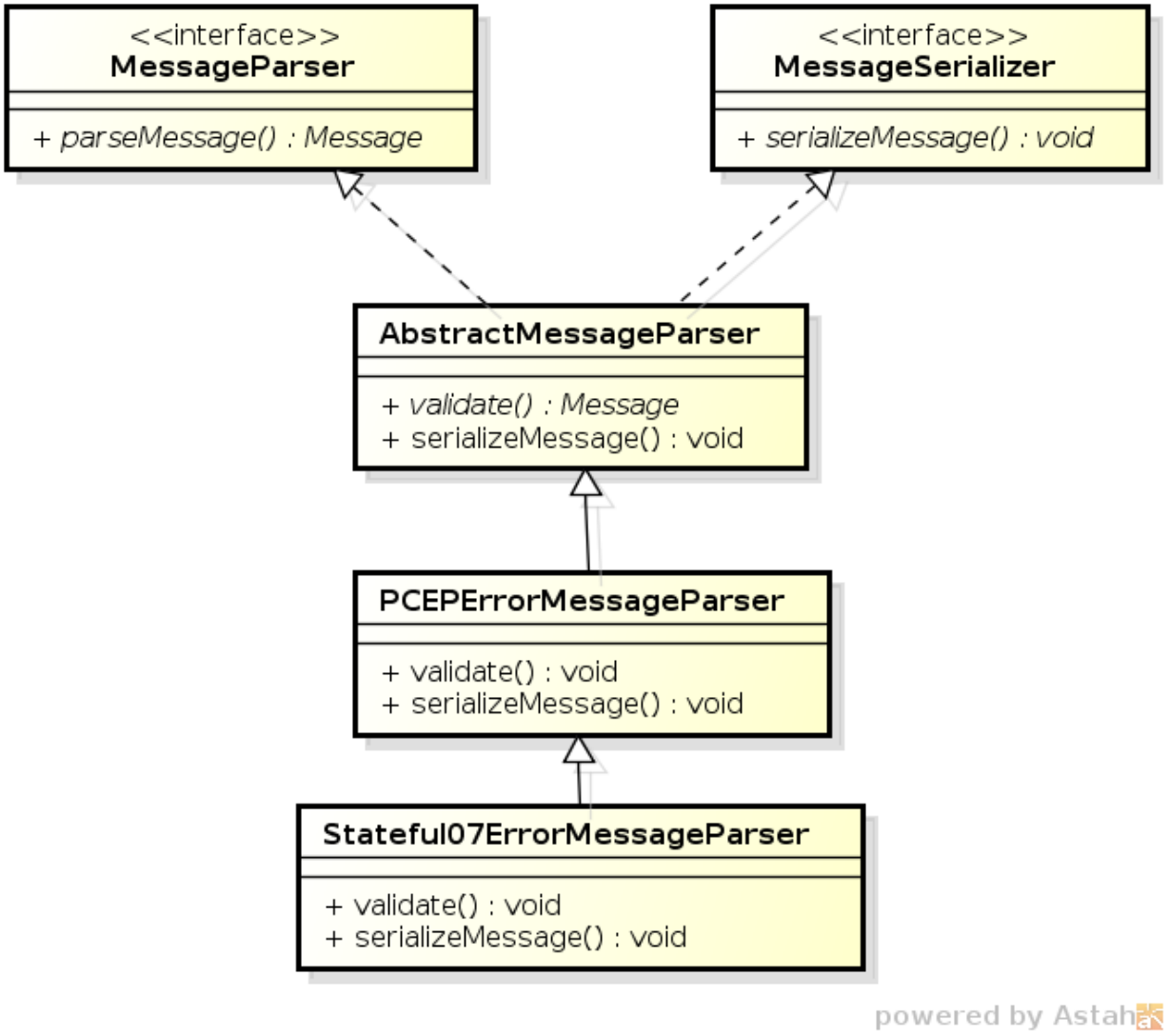


Fig. 2.52: Extending existing parsers

All parsers (including those for newly defined PCEP elements) have to be registered via the *Activator* class. This class is present in both modules.

In addition to parsers, the stateful module also introduces additional session proposal. This proposal includes new fields defined in stateful drafts for Open object.

PCEP segment routing (SR)

PCEP Segment Routing is an extension of base PCEP and pcep-ietf-stateful-07 extension. The pcep-segment-routing module implements draft-ietf-pce-segment-routing-01.

The extension brings new SR-ERO (Explicit Route Object) and SR-RRO (Reported Route Object) subobject composed of SID (Segment Identifier) and/or NAI (Node or Adjacency Identifier). The segment Routing path is carried in the

ERO and RRO object, as a list of SR-ERO/SR-RRO subobjects in an order specified by the user. The draft defines new TLV - SR-PCE-CAPABILITY TLV, carried in PCEP Open object, used to negotiate Segment Routing ability.

The yang models of subobject, SR-PCE-CAPABILITY TLV and appropriate augmentations are defined in odl-pcep-segment-routing.yang.
The pcep-segment-routing module includes parsers/serializers for new subobject (SrEroSubobjectParser) and TLV (SrPceCapabilityTlvParser).

The pcep-segment-routing module implements draft-ietf-pce-lsp-setup-type-01, too. The draft defines new TLV - Path Setup Type TLV, which value indicate path setup signaling technique. The TLV may be included in RP(Request Parameters)/SRP(Stateful PCE Request Parameters) object. For the default RSVP-TE (Resource Reservation Protocol), the TLV is omitted. For Segment Routing, PST = 1 is defined.

The Path Setup Type TLV is modeled with yang in module pcep-types.yang. A parser/serializer is implemented in PathSetupTypeTlvParser and it is overriden in segment-routing module to provide the aditional PST.

PCEP Synchronization Procedures Optimization

Optimizations of Label Switched Path State Synchronization Procedures for a Stateful PCE draft-ietf-pce-stateful-sync-optimizations-03 specifies following optimizations for state synchronization and the corresponding PCEP procedures and extensions:

- **State Synchronization Avoidance:** To skip state synchronization if the state has survived and not changed during session restart.
- **Incremental State Synchronization:** To do incremental (delta) state synchronization when possible.
- **PCE-triggered Initial Synchronization:** To let PCE control the timing of the initial state synchronization. The capability can be applied to both full and incremental state synchronization.
- **PCE-triggered Re-synchronization:** To let PCE re-synchronize the state for sanity check.

PCEP Topology

PCEP data is displayed only through one URL that is accessible from the base network-topology URL:

http://localhost:8181/restconf/operational/network-topology:network-topology/topology/pcep-topology

Each PCC will be displayed as a node:

```
<node>
 <path-computation-client>
  <ip-address>42.42.42.42</ip-address>
  <state-sync>synchronized</state-sync>
  <stateful-tlv>
   <stateful>
    <initiation>true</initiation>
    <lsp-update-capability>true</lsp-update-capability>
   </stateful>
  </stateful-tlv>
 </path-computation-client>
 <node-id>pcc://42.42.42.42</node-id>
</node>
</source>
```

If some tunnels are configured on the network, they would be displayed on the same page, within a node that initiated the tunnel:

```
<node>
 <path-computation-client>
  <state-sync>synchronized</state-sync>
  <stateful-tlv>
   <stateful>
    <initiation>true</initiation>
    <lsp-update-capability>true</lsp-update-capability>
   </stateful>
  </stateful-tlv>
  <reported-lsp>
   <name>foo</name>
   <lsp>
    <operational>down</operational>
    <sync>false</sync>
    <ignore>false</ignore>
    <plsp-id>1</plsp-id>
    <create>false</create>
    <administrative>true</administrative>
    <remove>false</remove>
    <delegate>true</delegate>
    <processing-rule>false</processing-rule>
    <tlvs>
    <lsp-identifiers>
      <ipv4>
       <ipv4-tunnel-sender-address>43.43.43.43</ipv4-tunnel-sender-address>
       <ipv4-tunnel-endpoint-address>0.0.0.0</ipv4-tunnel-endpoint-address>
       <ipv4-extended-tunnel-id>0.0.0.0</ipv4-extended-tunnel-id>
      </ipv4>
      <tunnel-id>0</tunnel-id>
      <lsp-id>0</lsp-id>
     </lsp-identifiers>
     <symbolic-path-name>
      <path-name>Zm9v</path-name>
     </symbolic-path-name>
    </tlvs>
   </lsp>
  </reported-lsp>
  <ip-address>43.43.43.43</ip-address>
 </path-computation-client>
 <node-id>pcc://43.43.43.43</node-id>
</node>
```

Note that, the <*path-name*> tag displays tunnel name in Base64 encoding.

API Reference Documentation

Javadocs are generated while creating mvn:site and they are located in target/ directory in each module.

PacketCable Developer Guide

PCMM Specification

PacketCable™ Multimedia Specification

System Overview

These components introduce a DOCSIS QoS Service Flow management using the PCMM protocol. The driver component is responsible for the PCMM/COPS/PDP functionality required to service requests from PacketCable Provider and FlowManager. Requests are transposed into PCMM Gate Control messages and transmitted via COPS to the CCAP/CMTS. This plugin adheres to the PCMM/COPS/PDP functionality defined in the CableLabs specification. PacketCable solution is an MDSAL compliant component.

PacketCable Components

The packetcable maven project is comprised of several modules.

Bundle	Description
packetcable-driver	A common module that containts the COPS stack and manages all connections to CCAPS/CMTSes.
packetcable-emulator	A basic CCAP emulator to facilitate testing the the plugin when no physical CCAP is avaible.
packetcable-policy-karaf	Generates a Karaf distribution with a config that loads all the packetcable features at runtime.
packetcable-policy-model	Contains the YANG information model.
packetcable-policy-server	Provider hosts the model processing, RESTCONF, and API implementation.

Setting Logging Levels

From the Karaf console

```
log:set <LEVEL> (<PACKAGE>|<BUNDLE>)
Example
log:set DEBUG org.opendaylight.packetcable.packetcable-policy-server
```

Tools for Testing

Postman REST client for Chrome

Install the Chrome extension

Download and import sample packetcable collection

View Rest API

1. Install the `odl-mdsal-apidocs` feature from the karaf console.
2. Open http://localhost:8181/apidoc/explorer/index.html default dev build user/pass is admin/admin
3. Navigate to the PacketCable section.

Yang-IDE

Editing yang can be done in any text editor but Yang-IDE will help prevent mistakes.

Setup and Build Yang-IDE for Eclipse

Using Wireshark to Trace PCMM

1. To start wireshark with privileges issue the following command:

```
sudo wireshark &
```

2. Select the interface to monitor.
3. Use the Filter to only display COPS messages by applying "cops" in the filter field.

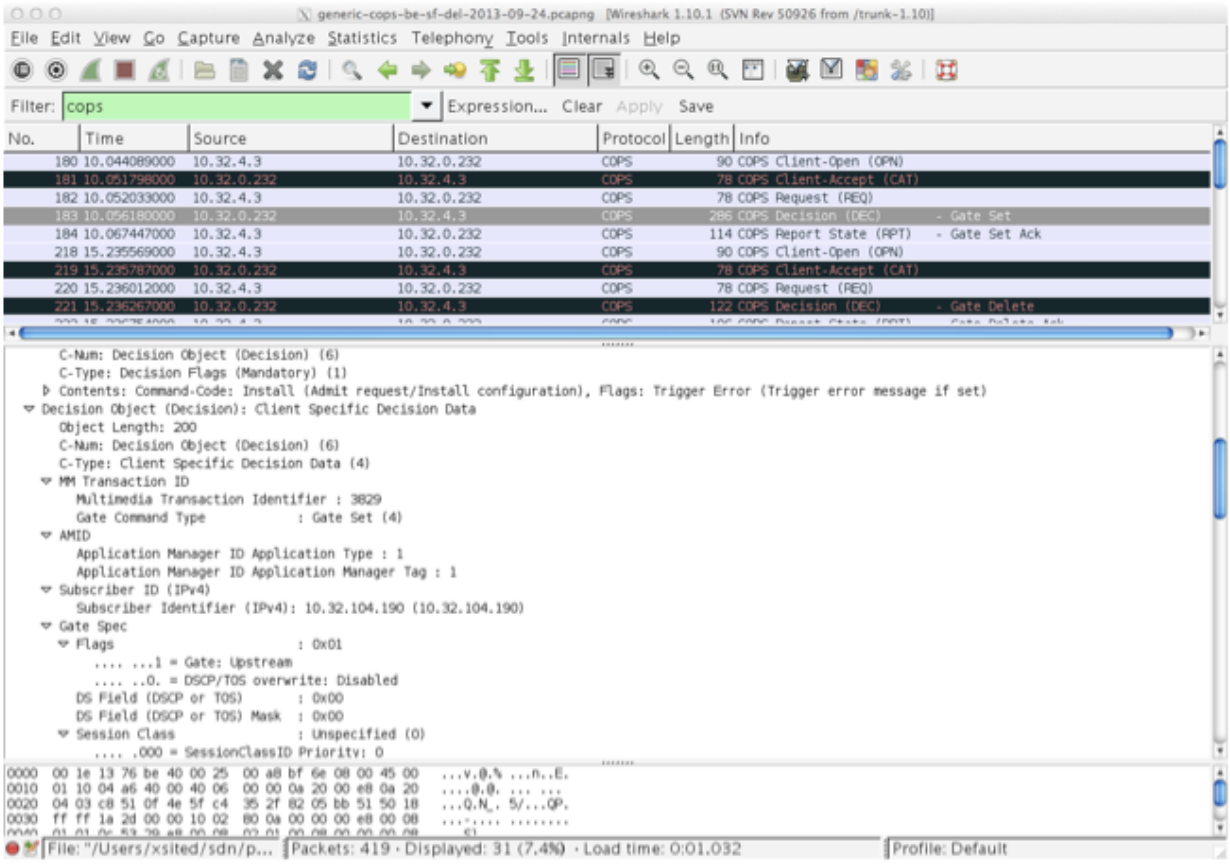


Fig. 2.53: Wireshark looking for COPS messages.

Debugging and Verifying DQoS Gate (Flows) on the CCAP/CMTS

Below are some of the most useful CCAP/CMTS commands to verify flows have been enabled on the CMTS.

Cisco

Cisco CMTS Cable Command Reference

Find the Cable Modem

```
10k2-DSG#show cable modem
                                                                                  D
MAC Address    IP Address      I/F           MAC              Prim RxPwr  Timing Num I
                                             State            Sid  (dBmv) Offset CPE P
0010.188a.faf6 0.0.0.0         C8/0/0/U0     offline          1    0.00   1482   0   N
74ae.7600.01f3 10.32.115.150   C8/0/10/U0    online           1    -0.50  1431   0   Y
0010.188a.fad8 10.32.115.142   C8/0/10/UB    w-online         2    -0.50  1507   1   Y
000e.0900.00dd 10.32.115.143   C8/0/10/UB    w-online         3    1.00   1677   0   Y
e86d.5271.304f 10.32.115.168   C8/0/10/UB    w-online         6    -0.50  1419   1   Y
```

Show PCMM Plugin Connection

```
10k2-DSG#show packetcabl ?
  cms     Gate Controllers connected to this PacketCable client
  event   Event message server information
  gate    PacketCable gate information
  global  PacketCable global information

10k2-DSG#show packetcable cms
GC-Addr        GC-Port  Client-Addr    COPS-handle  Version PSID Key PDD-Cfg


10k2-DSG#show packetcable cms
GC-Addr        GC-Port  Client-Addr    COPS-handle  Version PSID Key PDD-Cfg
10.32.0.240    54238    10.32.15.3     0x4B9C8150/1    4.0   0    0   0
```

Show COPS Messages

```
debug cops details
```

Use CM Mac Address to List Service Flows

```
10k2-DSG#show cable modem
                                                                                  D
MAC Address    IP Address      I/F           MAC              Prim RxPwr  Timing Num I
                                             State            Sid  (dBmv) Offset CPE P
0010.188a.faf6 ---             C8/0/0/UB     w-online         1    0.50   1480   1   N
74ae.7600.01f3 10.32.115.150   C8/0/10/U0    online           1    -0.50  1431   0   Y
0010.188a.fad8 10.32.115.142   C8/0/10/UB    w-online         2    -0.50  1507   1   Y
000e.0900.00dd 10.32.115.143   C8/0/10/UB    w-online         3    0.00   1677   0   Y
e86d.5271.304f 10.32.115.168   C8/0/10/UB    w-online         6    -0.50  1419   1   Y


10k2-DSG#show cable modem 000e.0900.00dd service-flow
```

```
SUMMARY:
MAC Address    IP Address      Host          MAC           Prim  Num Primary    DS
                               Interface     State         Sid   CPE Downstream RfId
000e.0900.00dd 10.32.115.143   C8/0/10/UB    w-online      3     0   Mo8/0/2:1  2353


Sfid  Dir Curr  Sid   Sched  Prio MaxSusRate  MaxBrst     MinRsvRate  Throughput
          State       Type
23    US  act   3     BE     0    0           3044        0           39
30    US  act   16    BE     0    500000      3044        0           0
24    DS  act   N/A   N/A    0    0           3044        0           17



UPSTREAM SERVICE FLOW DETAIL:

SFID  SID   Requests   Polls      Grants     Delayed    Dropped    Packets
                                             Grants     Grants
23    3     784        0          784        0          0          784
30    16    0          0          0          0          0          0


DOWNSTREAM SERVICE FLOW DETAIL:

SFID  RP_SFID QID    Flg Policer               Scheduler             FrwdIF
                         Xmits      Drops      Xmits      Drops
24    33019   131550     0          0          777        0          Wi8/0/2:2

Flags Legend:
$: Low Latency Queue (aggregated)
~: CIR Queue
```

Deleting a PCMM Gate Message from the CMTS

```
10k2-DSG#test cable dsd  000e.0900.00dd 30
```

Find service flows

All gate controllers currently connected to the PacketCable client are displayed

```
show cable modem 00:11:22:33:44:55 service flow   ????
show cable modem
```

Debug and display PCMM Gate messages

```
debug packetcable gate control
debug packetcable gate events
show packetcable gate summary
show packetcable global
show packetcable cms
```

Debug COPS messages

```
debug cops detail
debug packetcable cops
debug cable dynamic_qos trace
```

Integration Verification

Checkout the integration project and perform regression tests.

```
git clone ssh://${ODL_USERNAME}@git.opendaylight.org:29418/integration.git
git clone https:/git.opendaylight.org/gerrit/integration.git
```

1. Check and edit the integration/features/src/main/resources/features.xml and follow the directions there.
2. Check and edit the integration/features/pom.xml and add a dependency for your feature file
3. Build integration/features and debug

```
mvn clean install
```

Test your feature in the integration/distributions/extra/karaf/ distribution

```
cd integration/distributions/extra/karaf/
mvn clean install
cd target/assembly/bin
./karaf
```

service-wrapper

Install http://karaf.apache.org/manual/latest/users-guide/wrapper.html

```
opendaylight-user@root>feature:install service-wrapper
opendaylight-user@root>wrapper:install --help
DESCRIPTION
        wrapper:install

Install the container as a system service in the OS.

SYNTAX
        wrapper:install [options]

OPTIONS
        -d, --display
                The display name of the service.
                (defaults to karaf)
        --help
                Display this help message
        -s, --start-type
                Mode in which the service is installed. AUTO_START or DEMAND_START␣
↪(Default: AUTO_START)
                (defaults to AUTO_START)
        -n, --name
                The service name that will be used when installing the service.␣
↪(Default: karaf)
                (defaults to karaf)
```

```
        -D, --description
                The description of the service.
                (defaults to )

opendaylight-user@root> wrapper:install
Creating file: /home/user/odl/distribution-karaf-0.5.0-Boron/bin/karaf-wrapper
Creating file: /home/user/odl/distribution-karaf-0.5.0-Boron/bin/karaf-service
Creating file: /home/user/odl/distribution-karaf-0.5.0-Boron/etc/karaf-wrapper.conf
Creating file: /home/user/odl/distribution-karaf-0.5.0-Boron/lib/libwrapper.so
Creating file: /home/user/odl/distribution-karaf-0.5.0-Boron/lib/karaf-wrapper.jar
Creating file: /home/user/odl/distribution-karaf-0.5.0-Boron/lib/karaf-wrapper-main.
↪jar

Setup complete.  You may wish to tweak the JVM properties in the wrapper␣
↪configuration file:
/home/user/odl/distribution-karaf-0.5.0-Boron/etc/karaf-wrapper.conf
before installing and starting the service.


Ubuntu/Debian Linux system detected:
  To install the service:
    $ ln -s /home/user/odl/distribution-karaf-0.5.0-Boron/bin/karaf-service /etc/init.
↪d/

  To start the service when the machine is rebooted:
    $ update-rc.d karaf-service defaults

  To disable starting the service when the machine is rebooted:
    $ update-rc.d -f karaf-service remove

  To start the service:
    $ /etc/init.d/karaf-service start

  To stop the service:
    $ /etc/init.d/karaf-service stop

  To uninstall the service :
    $ rm /etc/init.d/karaf-service
```

Service Function Chaining

OpenDaylight Service Function Chaining (SFC) Overview

OpenDaylight Service Function Chaining (SFC) provides the ability to define an ordered list of a network services (e.g. firewalls, load balancers). These service are then "stitched" together in the network to create a service chain. This project provides the infrastructure (chaining logic, APIs) needed for ODL to provision a service chain in the network and an end-user application for defining such chains.

- ACE - Access Control Entry
- ACL - Access Control List
- SCF - Service Classifier Function
- SF - Service Function
- SFC - Service Function Chain

- SFF - Service Function Forwarder
- SFG - Service Function Group
- SFP - Service Function Path
- RSP - Rendered Service Path
- NSH - Network Service Header

SFC Classifier Control and Date plane Developer guide

Overview

Description of classifier can be found in: https://datatracker.ietf.org/doc/draft-ietf-sfc-architecture/

Classifier manages everything from starting the packet listener to creation (and removal) of appropriate ip(6)tables rules and marking received packets accordingly. Its functionality is **available only on Linux** as it leverages **NetfilterQueue**, which provides access to packets matched by an **iptables** rule. Classifier requires **root privileges** to be able to operate.

So far it is capable of processing ACL for MAC addresses, ports, IPv4 and IPv6. Supported protocols are TCP and UDP.

Classifier Architecture

Python code located in the project repository sfc-py/common/classifier.py.

Note: classifier assumes that Rendered Service Path (RSP) **already exists** in ODL when an ACL referencing it is obtained

---

1. sfc_agent receives an ACL and passes it for processing to the classifier
2. the RSP (its SFF locator) referenced by ACL is requested from ODL
3. if the RSP exists in the ODL then ACL based iptables rules for it are applied

After this process is over, every packet successfully matched to an iptables rule (i.e. successfully classified) will be NSH encapsulated and forwarded to a related SFF, which knows how to traverse the RSP.

Rules are created using appropriate iptables command. If the Access Control Entry (ACE) rule is MAC address related both iptables and IPv6 tables rules are issued. If ACE rule is IPv4 address related, only iptables rules are issued, same for IPv6.

Note: iptables **raw** table contains all created rules

Information regarding already registered RSP(s) are stored in an internal data-store, which is represented as a dictionary:

```
{rsp_id: {'name': <rsp_name>,
          'chains': {'chain_name': (<ipv>,),
                     ...
                     },
          'sff': {'ip': <ip>,
                  'port': <port>,
```

```
                    'starting-index': <starting-index>,
                    'transport-type': <transport-type>
                    },
            },
...
}
```

- `name`: name of the RSP
- `chains`: dictionary of iptables chains related to the RSP with information about IP version for which the chain exists
- `SFF`: SFF forwarding parameters
  - `ip`: SFF IP address
  - `port`: SFF port
  - `starting-index`: index given to packet at first RSP hop
  - `transport-type`: encapsulation protocol

Key APIs and Interfaces

This features exposes API to configure classifier (corresponds to service-function-classifier.yang)

API Reference Documentation

See: sfc-model/src/main/yang/service-function-classifier.yang

SFC-OVS Plug-in

Overview

SFC-OVS provides integration of SFC with Open vSwitch (OVS) devices. Integration is realized through mapping of SFC objects (like SF, SFF, Classifier, etc.) to OVS objects (like Bridge, TerminationPoint=Port/Interface). The mapping takes care of automatic instantiation (setup) of corresponding object whenever its counterpart is created. For example, when a new SFF is created, the SFC-OVS plug-in will create a new OVS bridge and when a new OVS Bridge is created, the SFC-OVS plug-in will create a new SFF.

SFC-OVS Architecture

SFC-OVS uses the OVSDB MD-SAL Southbound API for getting/writing information from/to OVS devices. The core functionality consists of two types of mapping:

1. mapping from OVS to SFC
   - OVS Bridge is mapped to SFF
   - OVS TerminationPoints are mapped to SFF DataPlane locators
2. mapping from SFC to OVS
   - SFF is mapped to OVS Bridge
   - SFF DataPlane locators are mapped to OVS TerminationPoints

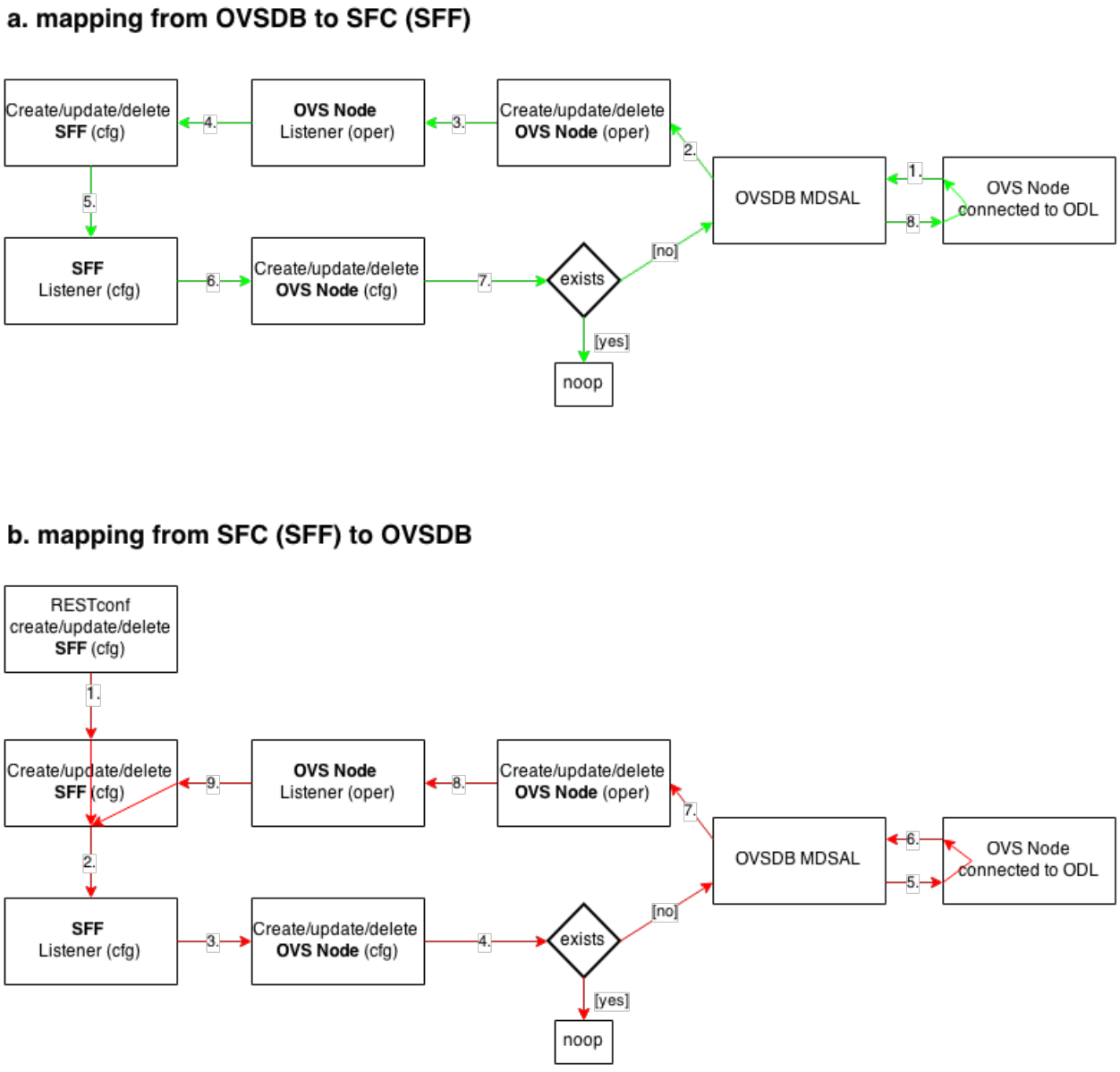


Fig. 2.54: SFC <—> OVS mapping flow diagram

Key APIs and Interfaces

- SFF to OVS mapping API (methods to convert SFF object to OVS Bridge and OVS TerminationPoints)
- OVS to SFF mapping API (methods to convert OVS Bridge and OVS TerminationPoints to SFF object)

SFC Southbound REST Plug-in

Overview

The Southbound REST Plug-in is used to send configuration from datastore down to network devices supporting a REST API (i.e. they have a configured REST URI). It supports POST/PUT/DELETE operations, which are triggered accordingly by changes in the SFC data stores.

- Access Control List (ACL)
- Service Classifier Function (SCF)
- Service Function (SF)
- Service Function Group (SFG)
- Service Function Schedule Type (SFST)
- Service Function Forwarder (SFF)
- Rendered Service Path (RSP)

Southbound REST Plug-in Architecture

1. **listeners** - used to listen on changes in the SFC data stores
2. **JSON exporters** - used to export JSON-encoded data from binding-aware data store objects
3. **tasks** - used to collect REST URIs of network devices and to send JSON-encoded data down to these devices

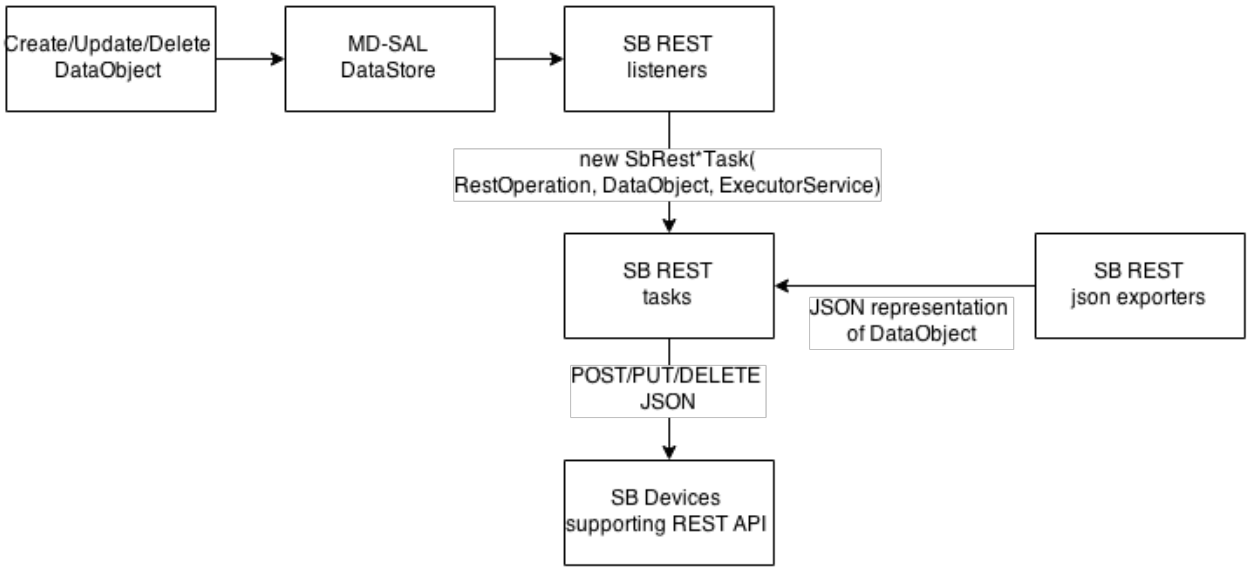


Fig. 2.55: Southbound REST Plug-in Architecture diagram

Key APIs and Interfaces

The plug-in provides Southbound REST API designated to listening REST devices. It supports POST/PUT/DELETE operations. The operation (with corresponding JSON-encoded data) is sent to unique REST URL belonging to certain data type.

- Access Control List (ACL): `http://<host>:<port>/config/ietf-acl:access-lists/access-list/`
- Service Function (SF): `http://<host>:<port>/config/service-function:service-functions/service-function/`
- Service Function Group (SFG): `http://<host>:<port>/config/service-function:service-function-groups/service-function-group/`
- Service Function Schedule Type (SFST): `http://<host>:<port>/config/service-function-scheduler-type:service-function-scheduler-types/service-function-scheduler-type/`
- Service Function Forwarder (SFF): `http://<host>:<port>/config/service-function-forwarder:service-function-forwarders/service-function-forwarder/`
- Rendered Service Path (RSP): `http://<host>:<port>/operational/rendered-service-path:rendered-service-paths/rendered-service-path/`

Therefore, network devices willing to receive REST messages must listen on these REST URLs.

Note: Service Classifier Function (SCF) URL does not exist, because SCF is considered as one of the network devices willing to receive REST messages. However, there is a listener hooked on the SCF data store, which is triggering POST/PUT/DELETE operations of ACL object, because ACL is referenced in `service-function-classifier.yang`

Service Function Load Balancing Developer Guide

Overview

SFC Load-Balancing feature implements load balancing of Service Functions, rather than a one-to-one mapping between Service Function Forwarder and Service Function.

Load Balancing Architecture

Service Function Groups (SFG) can replace Service Functions (SF) in the Rendered Path model. A Service Path can only be defined using SFGs or SFs, but not a combination of both.

Relevant objects in the YANG model are as follows:

1. Service-Function-Group-Algorithm:

```
Service-Function-Group-Algorithms {
    Service-Function-Group-Algorithm {
        String name
        String type
    }
}
```

```
Available types: ALL, SELECT, INDIRECT, FAST_FAILURE
```

2. Service-Function-Group:

```
Service-Function-Groups {
    Service-Function-Group {
        String name
        String serviceFunctionGroupAlgorithmName
        String type
        String groupId
        Service-Function-Group-Element {
            String service-function-name
            int index
        }
    }
}
```

3. ServiceFunctionHop: holds a reference to a name of SFG (or SF)

Key APIs and Interfaces

This feature enhances the existing SFC API.

REST API commands include: * For Service Function Group (SFG): read existing SFG, write new SFG, delete existing SFG, add Service Function (SF) to SFG, and delete SF from SFG * For Service Function Group Algorithm (SFG-Alg): read, write, delete

Bundle providing the REST API: sfc-sb-rest * Service Function Groups and Algorithms are defined in: sfc-sfg and sfc-sfg-alg * Relevant JAVA API: SfcProviderServiceFunctionGroupAPI, SfcProviderServiceFunctionGroupAlgAPI

Service Function Scheduling Algorithms

Overview

When creating the Rendered Service Path (RSP), the earlier release of SFC chose the first available service function from a list of service function names. Now a new API is introduced to allow developers to develop their own schedule algorithms when creating the RSP. There are four scheduling algorithms (Random, Round Robin, Load Balance and Shortest Path) are provided as examples for the API definition. This guide gives a simple introduction of how to develop service function scheduling algorithms based on the current extensible framework.

Architecture

The following figure illustrates the service function selection framework and algorithms.

The YANG Model defines the Service Function Scheduling Algorithm type identities and how they are stored in the MD-SAL data store for the scheduling algorithms.

The MD-SAL data store stores all informations for the scheduling algorithms, including their types, names, and status.

The API provides some basic APIs to manage the informations stored in the MD-SAL data store, like putting new items into it, getting all scheduling algorithms, etc.

The RESTCONF API provides APIs to manage the informations stored in the MD-SAL data store through RESTful calls.

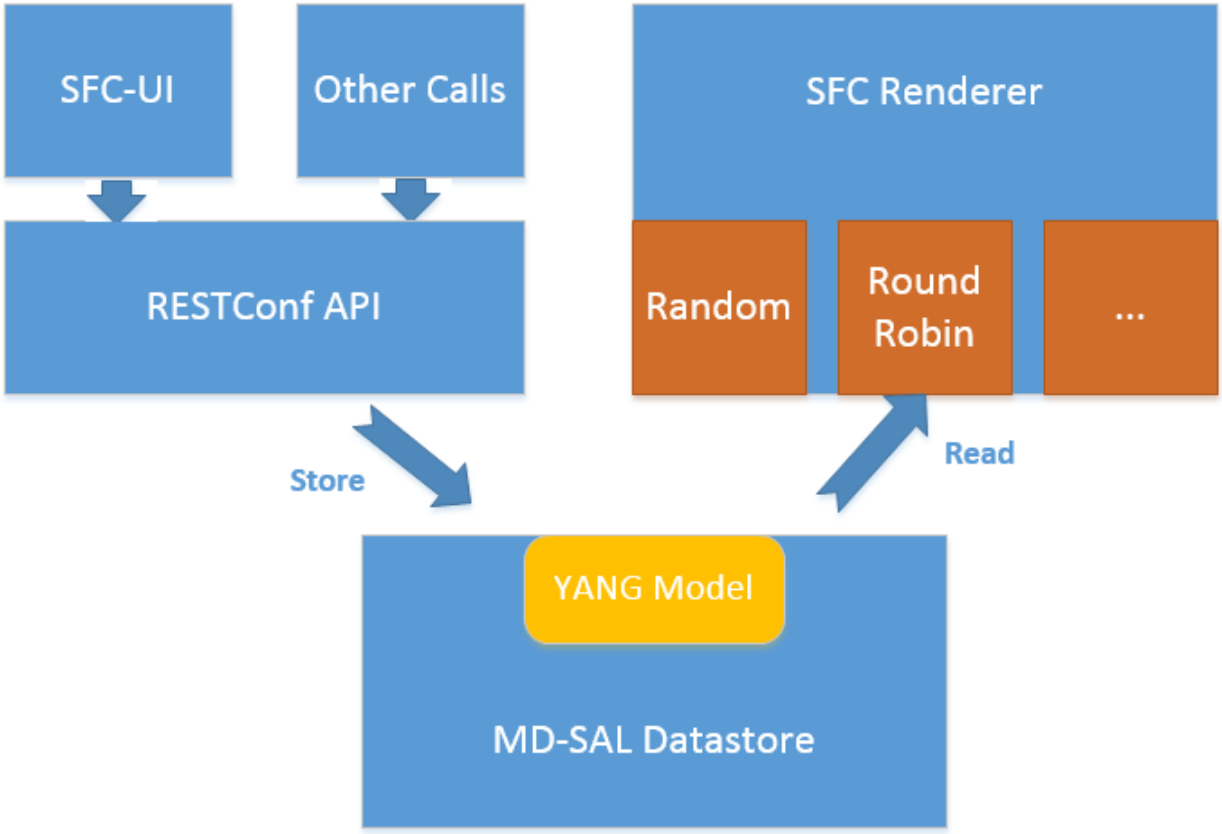


Fig. 2.56: SF Scheduling Algorithm framework Architecture

The Service Function Chain Renderer gets the enabled scheduling algorithm type, and schedules the service functions with scheduling algorithm implementation.

Key APIs and Interfaces

While developing a new Service Function Scheduling Algorithm, a new class should be added and it should extend the base schedule class SfcServiceFunctionSchedulerAPI. And the new class should implement the abstract function:

```
public List<String> scheduleServiceFuntions(ServiceFunctionChain chain, int
serviceIndex).
```

- **"ServiceFunctionChain chain"**: the chain which will be rendered
- **"int serviceIndex"**: the initial service index for this rendered service path
- **"List<String>"**: a list of service function names which scheduled by the Service Function Scheduling Algorithm.

API Reference Documentation

Please refer the API docs generated in the mdsal-apidocs.

SFC Proof of Transit Developer Guide

Overview

SFC Proof of Transit implements the in-situ OAM (iOAM) Proof of Transit verification for SFCs and other paths. The implementation is broadly divided into the North-bound (NB) and the South-bound (SB) side of the application. The NB side is primarily charged with augmenting the RSP with user-inputs for enabling the PoT on the RSP, while the SB side is dedicated to auto-generated SFC PoT parameters, periodic refresh of these parameters and delivering the parameters to the NETCONF and iOAM capable nodes (eg. VPP instances).

Architecture

The following diagram gives the high level overview of the different parts.

The Proof of Transit feature is enabled by two sub-features:

1. ODL SFC PoT: `feature:install odl-sfc-pot`
2. ODL SFC PoT NETCONF Renderer: `feature:install odl-sfc-pot-netconf-renderer`

Details

The following classes and handlers are involved.

1. The class (SfcPotRpc) sets up RPC handlers for enabling the feature.
2. There are new RPC handlers for two new RPCs (EnableSfcIoamPotRenderedPath and DisableSfcIoamPotRenderedPath) and effected via SfcPotRspProcessor class.
3. When a user configures via a POST RPC call to enable Proof of Transit on a particular SFC (via the Rendered Service Path), the configuration drives the creation of necessary augmentations to the RSP (to modify the RSP) to effect the Proof of Transit configurations.

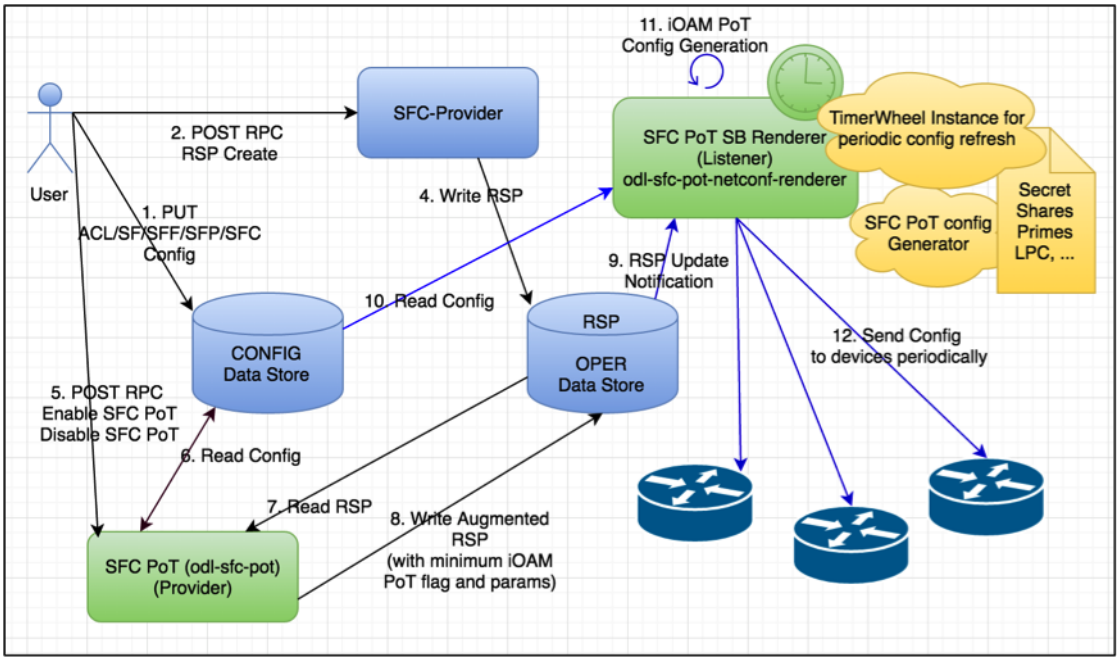


Fig. 2.57: SFC Proof of Transit Internal Architecture

4. The augmentation meta-data added to the RSP are defined in the sfc-ioam-nb-pot.yang file.

Note: There are no auto generated configuration parameters added to the RSP to avoid RSP bloat.

---

5. Adding SFC Proof of Transit meta-data to the RSP is done in the SfcPotRspProcessor class.
6. Once the RSP is updated, the RSP data listeners in the SB renderer modules (odl-sfc-pot-netconf-renderer) will listen to the RSP changes and send out configurations to the necessary network nodes that are part of the SFC.
7. The configurations are handled mainly in the SfcPotAPI, SfcPotConfigGenerator, SfcPotPolyAPI, SfcPotPolyClass and SfcPotPolyClassAPI classes.
8. There is a sfc-ioam-sb-pot.yang file that shows the format of the iOAM PoT configuration data sent to each node of the SFC.
9. A timer is started based on the "ioam-pot-refresh-period" value in the SB renderer module that handles configuration refresh periodically.
10. The SB and timer handling are done in the odl-sfc-pot-netconf-renderer module. Note: This is NOT done in the NB odl-sfc-pot module to avoid periodic updates to the RSP itself.
11. ODL creates a new profile of a set of keys and secrets at a constant rate and updates an internal data store with the configuration. The controller labels the configurations per RSP as "even" or "odd" – and the controller cycles between "even" and "odd" labeled profiles. The rate at which these profiles are communicated to the nodes is configurable and in future, could be automatic based on profile usage. Once the profile has been successfully communicated to all nodes (all Netconf transactions completed), the controller sends an "enable pot-profile" request to the ingress node.
12. The nodes are to maintain two profiles (an even and an odd pot-profile). One profile is currently active and in use, and one profile is about to get used. A flag in the packet is indicating whether the odd or even pot-profile is to be used by a node. This is to ensure that during profile change we're not disrupting the service. I.e. if the "odd" profile is active, the controller can communicate the "even" profile to all nodes and only if all the nodes

have received it, the controller will tell the ingress node to switch to the "even" profile. Given that the indicator travels within the packet, all nodes will switch to the "even" profile. The "even" profile gets active on all nodes – and nodes are ready to receive a new "odd" profile.

13. HashedTimerWheel implementation is used to support the periodic configuration refresh. The default refresh is 5 seconds to start with.
14. Depending on the last updated profile, the odd or the even profile is updated in the fresh timer pop and the configurations are sent down appropriately.
15. SfcPotTimerQueue, SfcPotTimerWheel, SfcPotTimerTask, SfcPotTimerData and SfcPotTimerThread are the classes that handle the Proof of Transit protocol profile refresh implementation.
16. The RSP data store is NOT being changed periodically and the timer and configuration refresh modules are present in the SB renderer module handler and hence there are are no scale or RSP churn issues affecting the design.

The following diagram gives the overall sequence diagram of the interactions between the different classes.

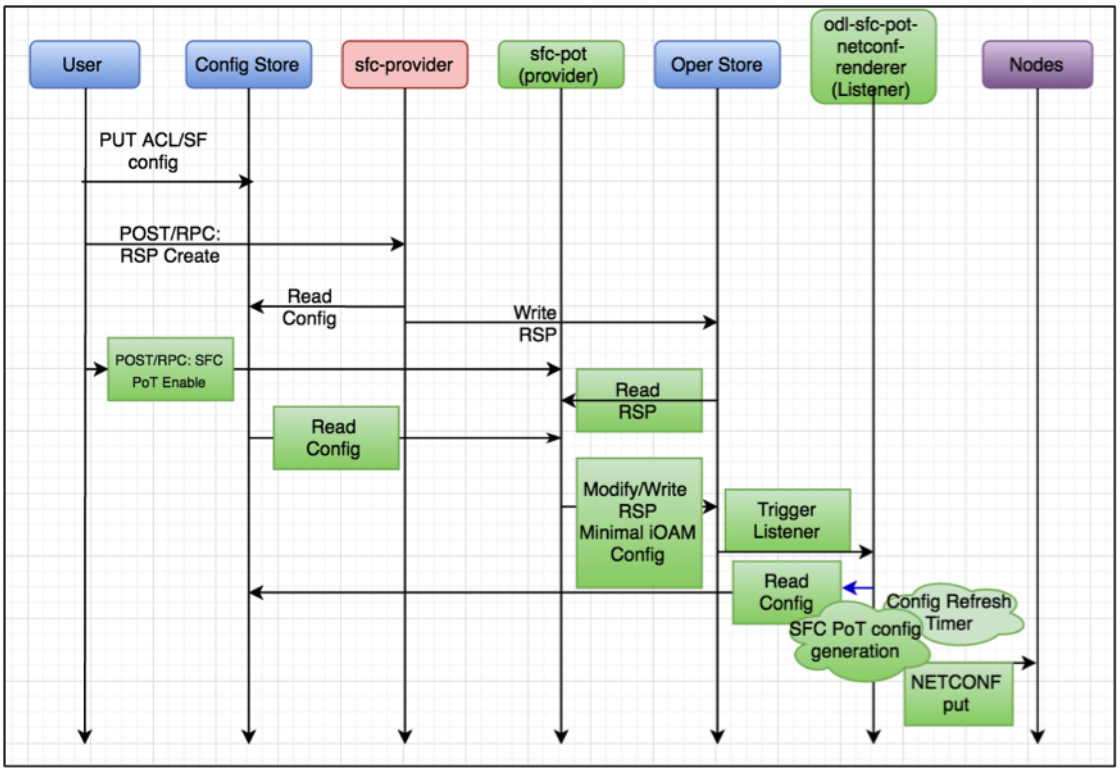


Fig. 2.58: SFC Proof of Transit Sequence Diagram

Logical Service Function Forwarder

Overview

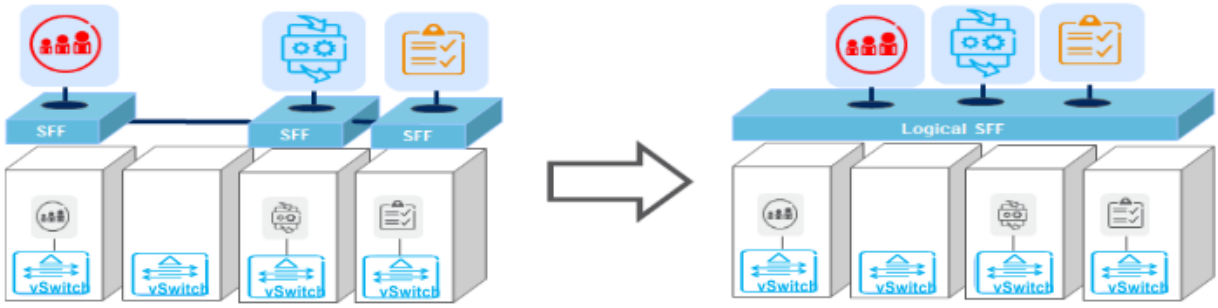
Rationale

When the current SFC is deployed in a cloud environment, it is assumed that each switch connected to a Service Function is configured as a Service Function Forwarder and each Service Function is connected to its Service Function

Forwarder depending on the Compute Node where the Virtual Machine is located. This solution allows the basic cloud use cases to be fulfilled, as for example, the ones required in OPNFV Brahmaputra, however, some advanced use cases, like the transparent migration of VMs can not be implemented. The Logical Service Function Forwarder enables the following advanced use cases:

1. Service Function mobility without service disruption
2. Service Functions load balancing and failover

As shown in the picture below, the Logical Service Function Forwarder concept extends the current SFC northbound API to provide an abstraction of the underlying Data Center infrastructure. The Data Center underlaying network can be abstracted by a single SFF. This single SFF uses the logical port UUID as data plane locator to connect SFs globally and in a location-transparent manner. SFC makes use of Genius project to track the location of the SF's logical ports.



The SFC internally distributes the necessary flow state over the relevant switches based on the internal Data Center topology and the deployment of SFs.

Changes in data model

The Logical Service Function Forwarder concept extends the current SFC northbound API to provide an abstraction of the underlying Data Center infrastructure.

The Logical SFF simplifies the configuration of the current SFC data model by reducing the number of parameters to be be configured in every SFF, since the controller will discover those parameters by interacting with the services offered by the Genius project.

The following picture shows the Logical SFF data model. The model gets simplified as most of the configuration parameters of the current SFC data model are discovered in runtime. The complete YANG model can be found here logical SFF model.

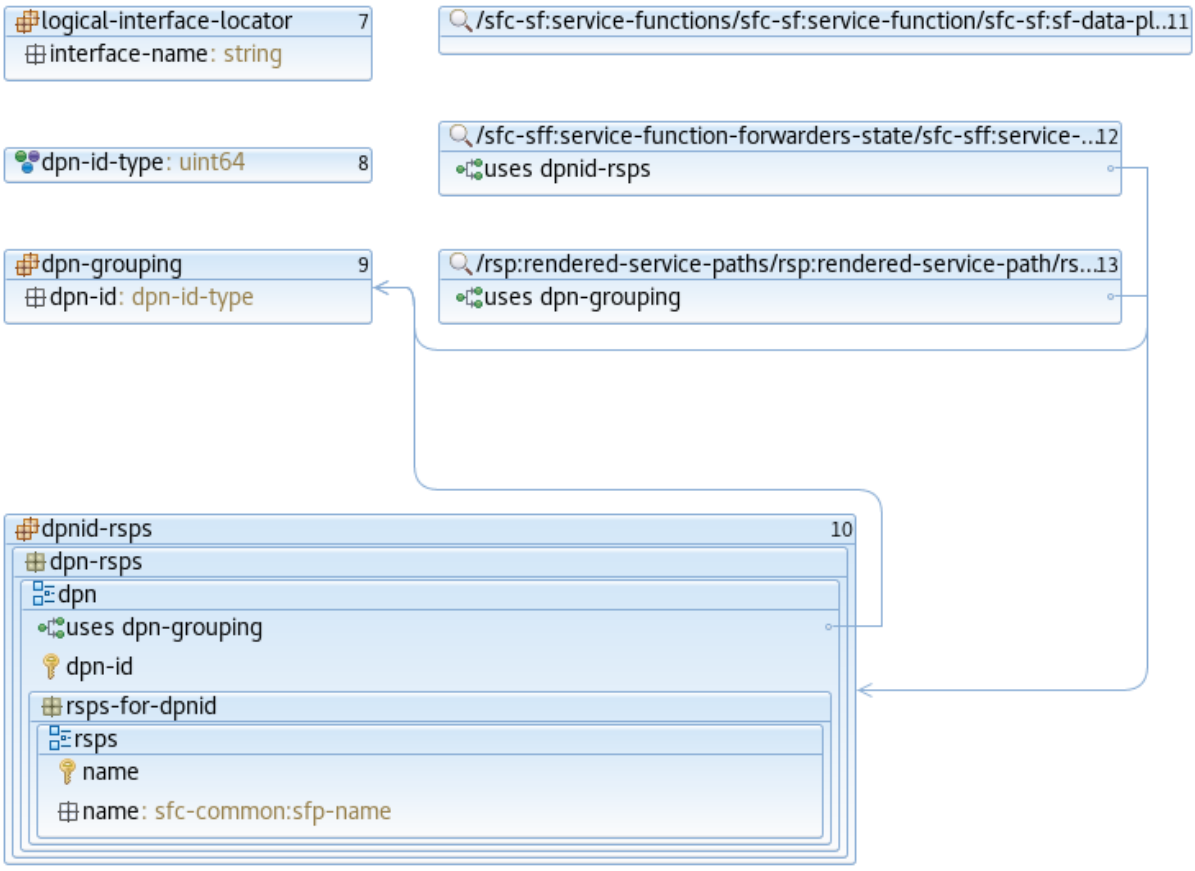
There are other minor changes in the data model; the SFC encapsulation type has been added or moved in the following files:

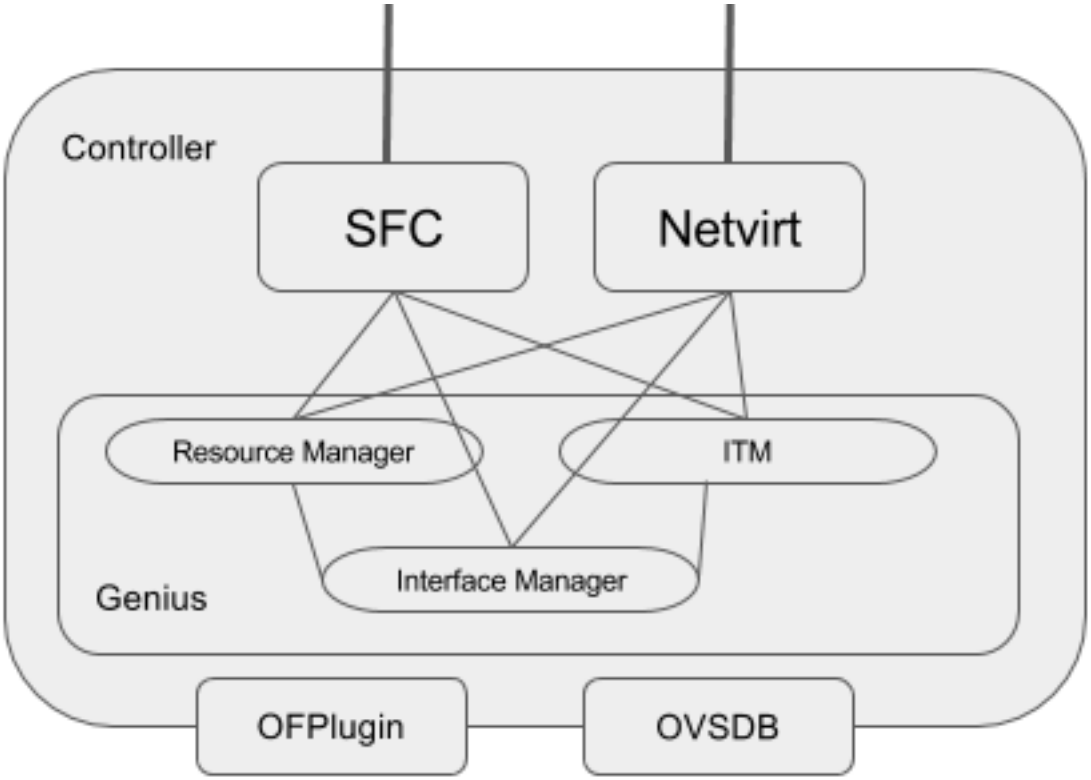
- RSP data model
- SFP data model
- Service Locator data model

Interaction with Genius

Feature *sfc-genius* functionally enables SFC integration with Genius. This allows configuring a Logical SFF and SFs attached to this Logical SFF via logical interfaces (i.e. neutron ports) that are registered with Genius.

As shown in the following picture, SFC will interact with Genius project's services to provide the Logical SFF functionality.


logical-interface-locator 7
interface-name: string
/sfc-sf:service-functions/sfc-sf:service-function/sfc-sf:sf-data-pl..11
dpn-id-type: uint64 8
/sfc-sff:service-function-forwarders-state/sfc-sff:service-...12
uses dpnid-rsps
dpn-grouping 9
dpn-id: dpn-id-type
/rsp:rendered-service-paths/rsp:rendered-service-path/rs...13
uses dpn-grouping
dpnid-rsps 10
dpn-rsps
dpn
uses dpn-grouping
dpn-id
rsps-for-dpnid
rsps
name
name: sfc-common:sfp-name



The following are the main Genius' services used by SFC:

1. Interaction with Interface Tunnel Manager (ITM)
2. Interaction with the Interface Manager
3. Interaction with Resource Manager

SFC Service registration with Genius

Genius handles the coexistence of different network services. As such, SFC service is registered with Genius performing the following actions:

SFC Service Binding As soon as a Service Function associated to the Logical SFF is involved in a Rendered Service Path, SFC service is bound to its logical interface via Genius Interface Manager. This has the effect of forwarding every incoming packet from the Service Function to the SFC pipeline of the attached switch, as long as it is not consumed by a different bound service with higher priority.

SFC Service Terminating Action As soon as SFC service is bound to the interface of a Service Function for the first time on a specific switch, a terminating service action is configured on that switch via Genius Interface Tunnel Manager. This has the effect of forwarding every incoming packet from a different switch to the SFC pipeline as long as the traffic is VXLAN encapsulated on VNI 0.

The following sequence diagrams depict how the overall process takes place:

For more information on how Genius allows different services to coexist, see the *Genius User Guide*.

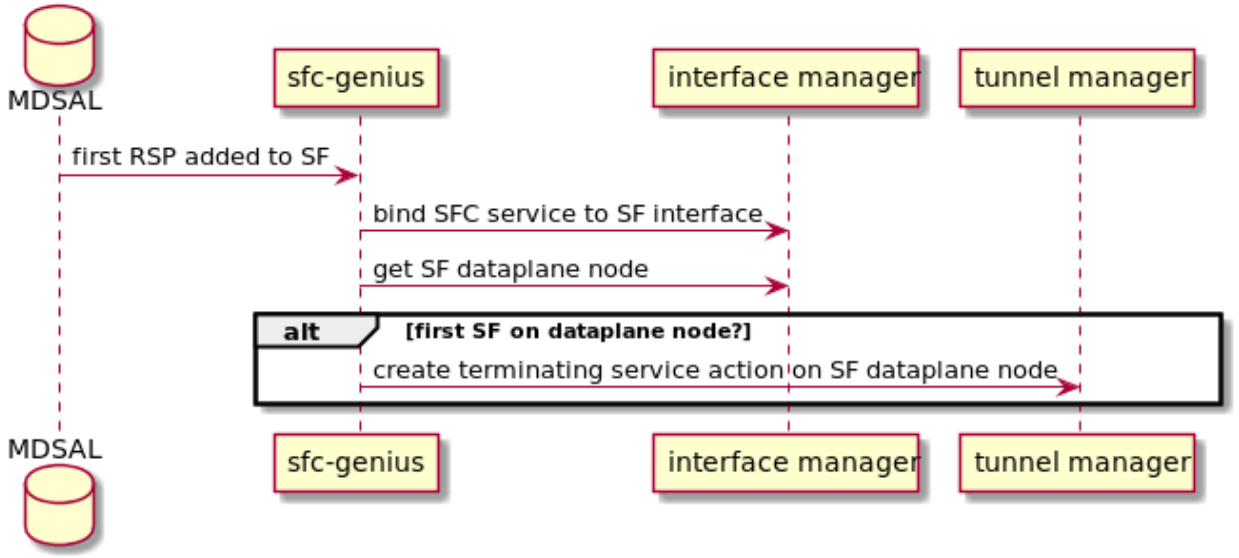


Fig. 2.59: SFC genius module interaction with Genius at RSP creation.

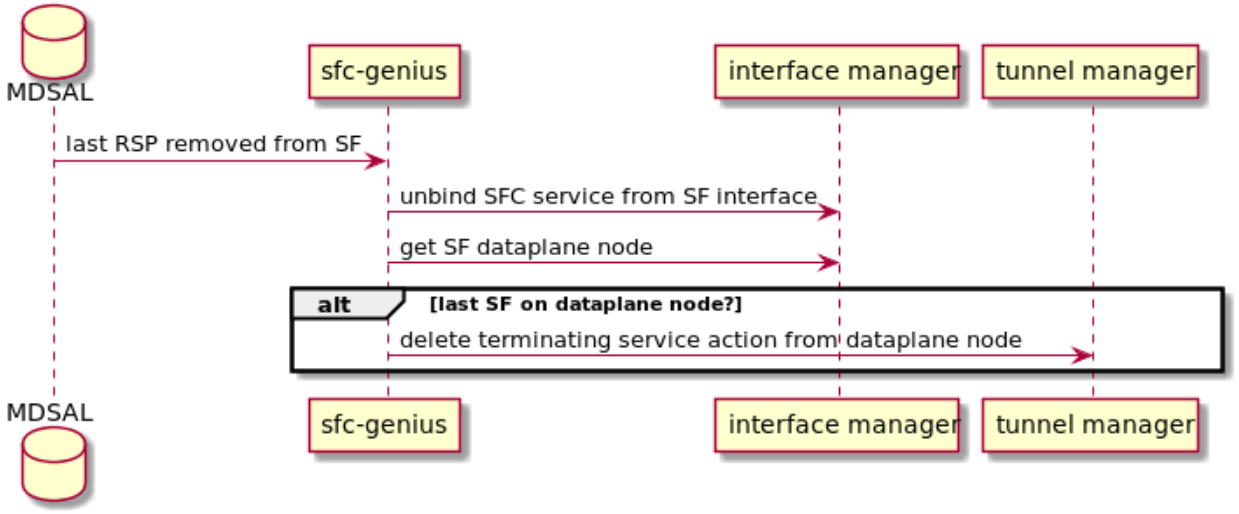


Fig. 2.60: SFC genius module interaction with Genius at RSP removal.

Path Rendering

During path rendering, Genius is queried to obtain needed information, such as:

- Location of a logical interface on the data-plane.
- Tunnel interface for a specific pair of source and destination switches.
- Egress OpenFlow actions to output packets to a specific interface.

See *RSP Rendering* section for more information.

VM migration

Upon VM migration, it's logical interface is first unregistered and then registered with Genius, possibly at a new physical location. *sfc-genius* reacts to this by re-rendering all the RSPs on which the associated SF participates, if any.

The following picture illustrates the process:

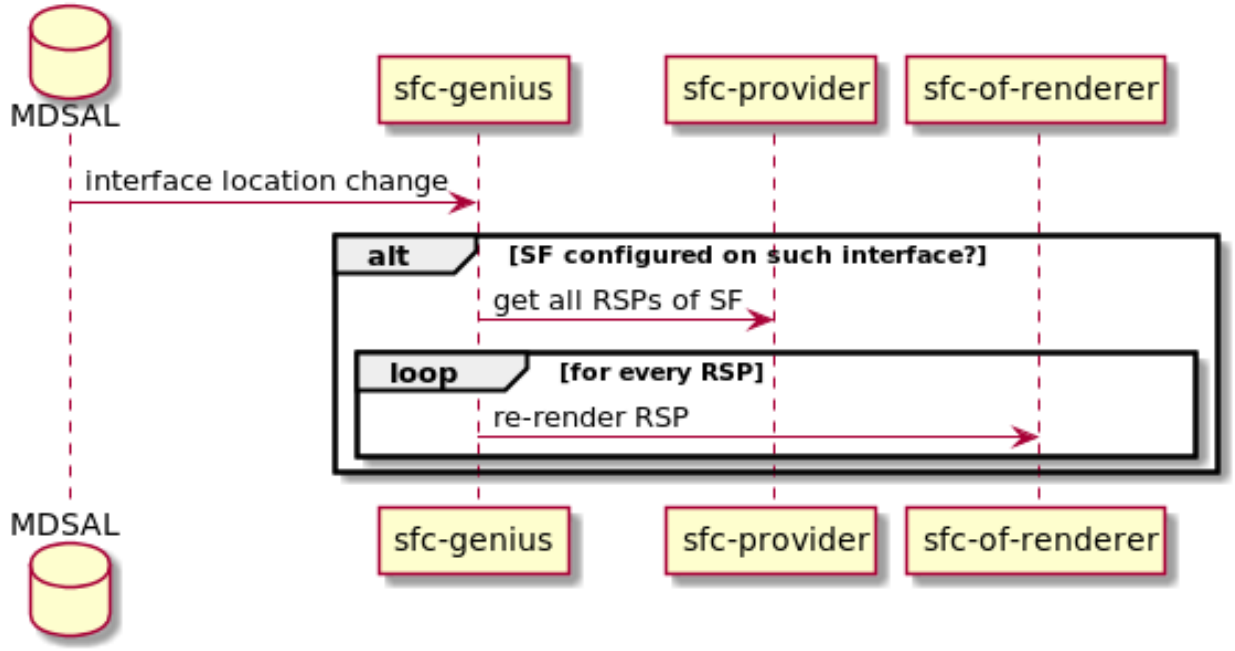


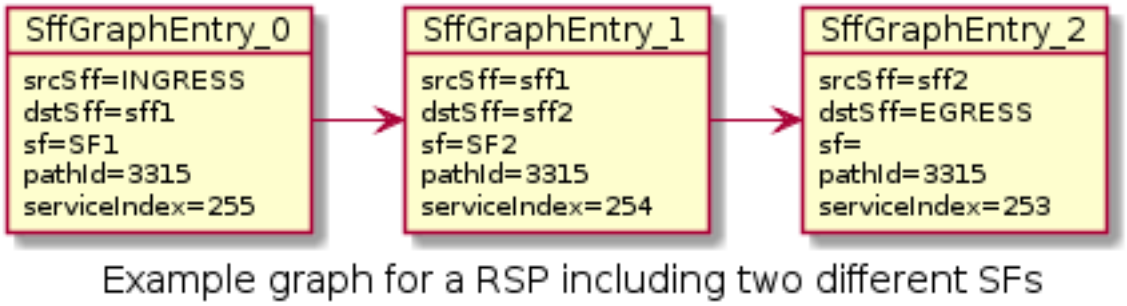
Fig. 2.61: SFC genius module at VM migration.

RSP Rendering changes for paths using the Logical SFF

1. **Construction of the auxiliary rendering graph**

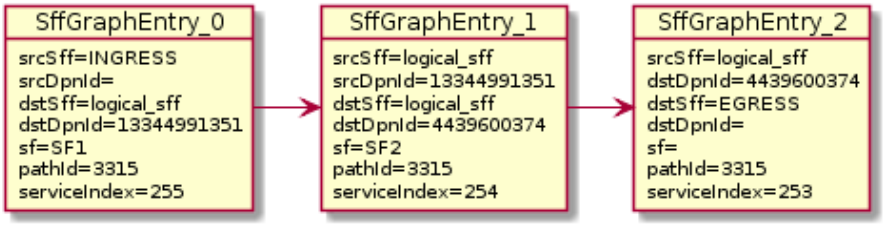
   When starting the rendering of a RSP, the SFC renderer builds an auxiliary graph with information about the required hops for traffic traversing the path. RSP processing is achieved by iteratively evaluating each of the entries in the graph, writing the required flows in the proper switch for each hop.

   It is important to note that the graph includes both traffic ingress (i.e. traffic entering into the first SF) and traffic egress (i.e. traffic leaving the chain from the last SF) as hops. Therefore, the number of entries in the graph equals the number of SFs in the chain plus one.



Example graph for a RSP including two different SFs

The process of rendering a chain when the switches involved are part of the Logical SFF also starts with the construction of the hop graph. The difference is that when the SFs used in the chain are using a logical interface, the SFC renderer will also retrieve from Genius the DPIDs for the switches, storing them in the graph. In this context, those switches are the ones in the compute nodes each SF is hosted on at the time the chain is rendered.



Example graph for a RSP including two different SFs that are configured using a Logical SFF. It can be observed that the service functions SF1, SF2 are hosted in different compute nodes

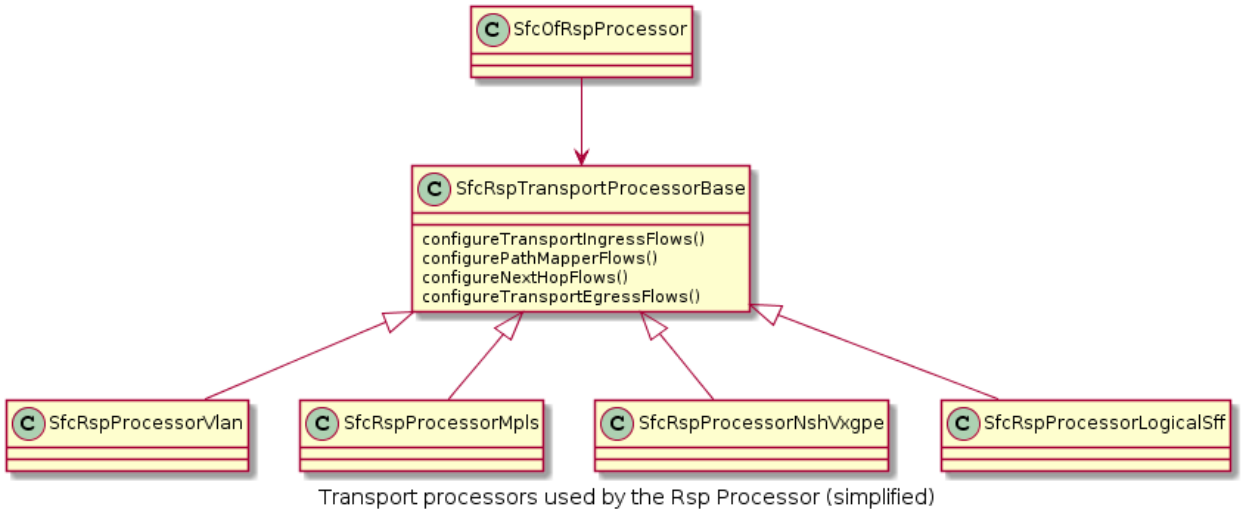
2. **New transport processor**

   Transport processors are classes which calculate and write the correct flows for a chain. Each transport processor specializes on writing the flows for a given combination of transport type and SFC encapsulation.

   A specific transport processor has been created for paths using a Logical SFF. A particularity of this transport processor is that its use is not only determined by the transport / SFC encapsulation combination, but also because the chain is using a Logical SFF. The actual condition evaluated for selecting the Logical SFF transport processor is that the SFs in the chain are using logical interface locators, and that the DPIDs for those locators can be successfully retrieved from Genius.

   The main differences between the Logical SFF transport processor and other processors are the following:

   - Instead of srcSff, dstSff fields in the hops graph (which are all equal in a path using a Logical SFF), the Logical SFF transport processor uses previously stored srcDpnId, dstDpnId fields in order to know whether an actual hop between compute nodes must be performed or not (it is possible that two consecutive SFs are collocated in the same compute node).
   - When a hop between switches really has to be performed, it relies on Genius for getting the actions to perform that hop. The retrieval of those actions involve two steps:
     - First, Genius' Overlay Tunnel Manager module is used in order to retrieve the target interface for a jump between the source and the destination DPIDs.
     - Then, egress instructions for that interface are retrieved from Genius's Interface Manager.
   - There are no next hop rules between compute nodes, only egress instructions (the transport zone tunnels have all the required routing information).



- Next hop information towards SFs uses mac adresses which are also retrieved from the Genius datastore.
- The Logical SFF transport processor performs NSH decapsulation in the last switch of the chain.

3. **Post-rendering update of the operational data model**

   When the rendering of a chain finishes successfully, the Logical SFF Transport Processor perform two operational datastore modifications in order to provide some relevant runtime information about the chain. The exposed information is the following:

   - Rendered Service Path state: when the chain uses a Logical SFF, DPIDs for the switches in the compute nodes on which the SFs participating in the chain are hosted are added to the hop information.
   - SFF state: A new list of all RSPs which use each DPID is has been added. It is updated on each RSP addition / deletion.

Classifier impacts

This section explains the changes made to the SFC classifier, enabling it to be attached to Logical SFFs.

Refer to the following image to better understand the concept, and the required steps to implement the feature.

As stated in the *SFC User Guide*, the classifier needs to be provisioned using logical interfaces as attachment points.

When that happens, MDSAL will trigger an event in the odl-sfc-scf-openflow feature (i.e. the sfc-classifier), which is responsible for installing the classifier flows in the classifier switches.

The first step of the process, is to bind the interfaces to classify in Genius, in order for the desired traffic (originating from the VMs having the provisioned attachment-points) to enter the SFC pipeline. This will make traffic reach table 82 (SFC classifier table), coming from table 0 (table managed by Genius, shared by all applications).

The next step, is deciding which flows to install in the SFC classifier table. A table-miss flow will be installed, having a MatchAny clause, whose action is to jump to Genius's egress dispatcher table. This enables traffic intended for other applications to still be processed.

The flow that allows the SFC pipeline to continue is added next, having higher match priority than the table-miss flow. This flow has two responsabilities:

1. **Push the NSH header, along with its metadata (required within the SFC pipeline)**

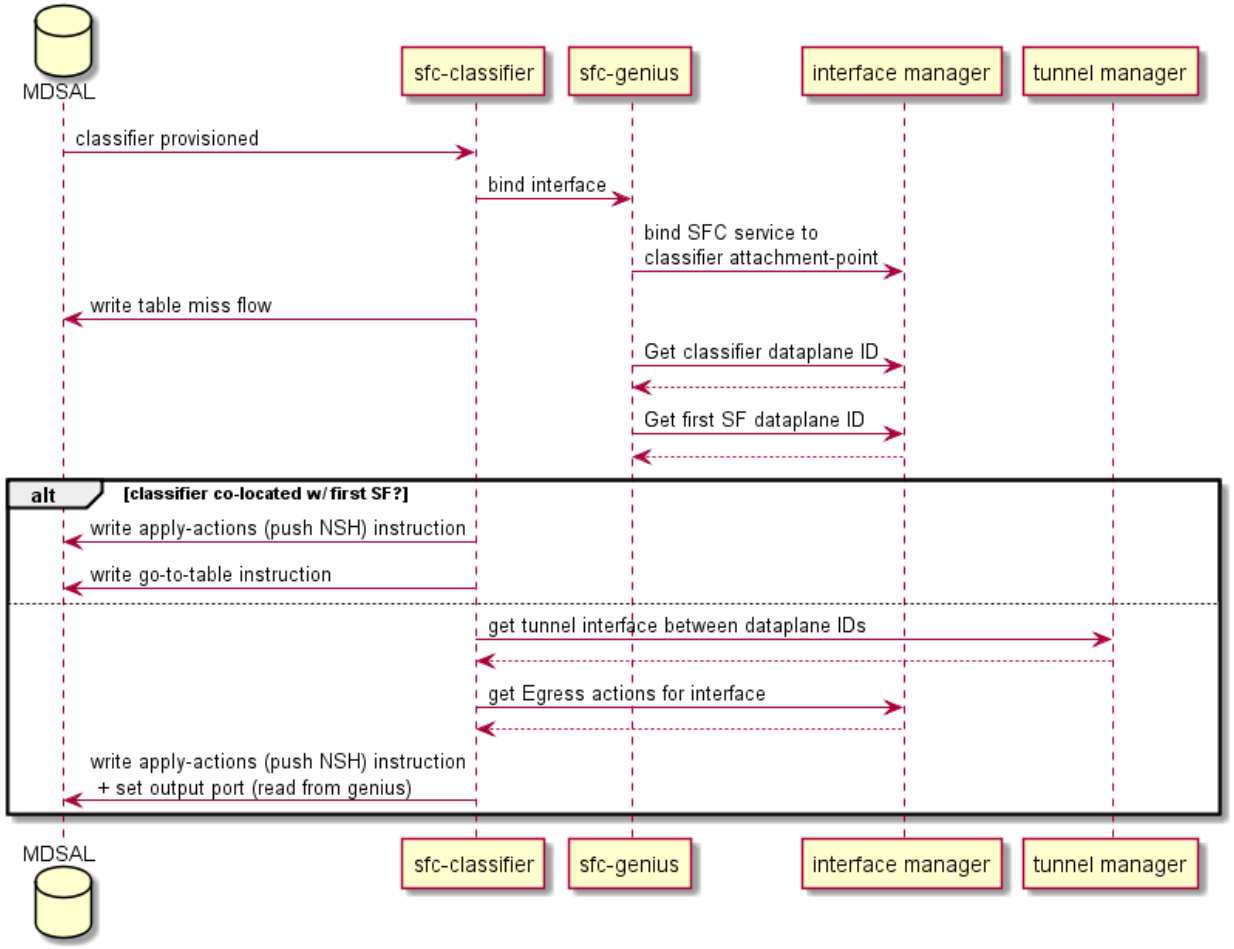


Fig. 2.62: SFC classifier integration with Genius.

Features the specified ACL matches as match criteria, and push NSH along with its metadata into the Action list.

2. **Advance the SFC pipeline**

   Forward the traffic to the first Service Function in the RSP. This steers packets into the SFC domain, and how it is done depends on whether the classifier is co-located with the first service function in the specified RSP.

   Should the classifier be co-located (i.e. in the same compute node), a new instruction is appended to the flow, telling all matches to jump to the transport ingress table.

   If not, Genius's tunnel manager service is queried to get the tunnel interface connecting the classifier node with the compute node where the first Service Function is located, and finally, Genius's interface manager service is queried asking for instructions on how to reach that tunnel interface.

   These actions are then appended to the Action list already containing push NSH and push NSH metadata Actions, and written in an Apply-Actions Instruction into the datastore.

SNMP4SDN Developer Guide

Overview

We propose a southbound plugin that can control the off-the-shelf commodity Ethernet switches for the purpose of building SDN using Ethernet switches. For Ethernet switches, forwarding table, VLAN table, and ACL are where one can install flow configuration on, and this is done via SNMP and CLI in the proposed plugin. In addition, some settings required for Ethernet switches in SDN, e.g., disabling STP and flooding, are proposed.

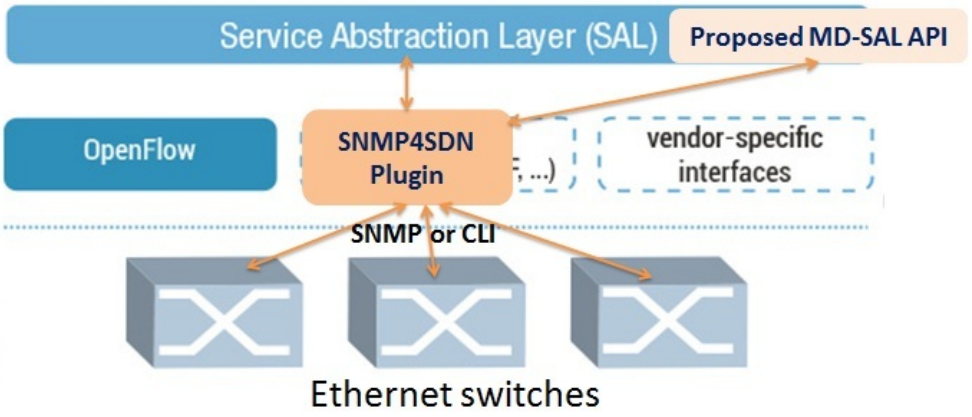


Fig. 2.63: SNMP4SDN as an OpenDaylight southbound plugin

Architecture

The modules in the plugin are depicted as the following figure.

- AclService: add/remove ACL profile and rule on the switches.
- FdbService: add/modify/remove FDB table entry on the switches.
- VlanService: add/modify/remove VLAN table entry on the switches.
- TopologyService: query and acquire the subnet topology.

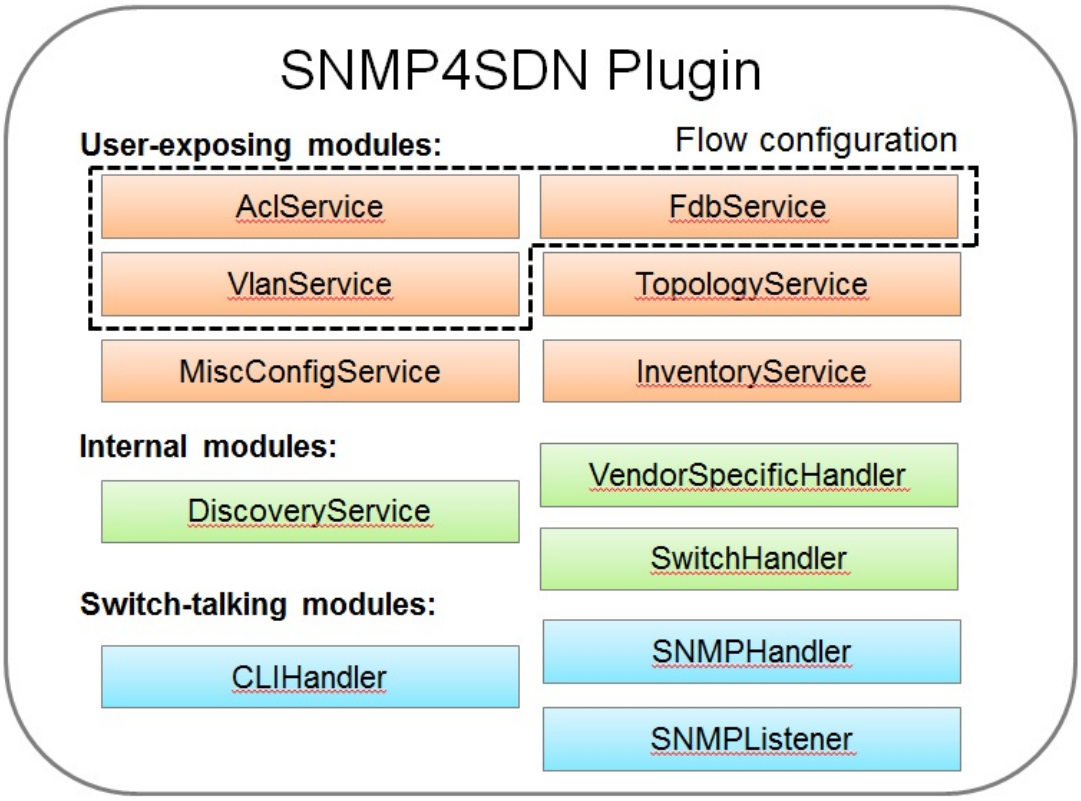


Fig. 2.64: Modules in the SNMP4SDN Plugin

- InventoryService: acquire the switches and their ports.
- DiscoveryService: probe and resolve the underlying switches as well as the port pairs connecting the switches. The probing is realized by SNMP queries. The updates from discovery will also be reflected to the TopologyService.
- MiscConfigService: do kinds of settings on switches
  - Supported STP and ARP settings such as enable/disable STP, get port's STP state, get ARP table, set ARP entry, and others
- VendorSpecificHandler: to assist the flow configuration services to call the switch-talking modules with correct parameters value and order.
- Switch-talking modules
  - For the services above, when they need to read or configure the underlying switches via SNMP or CLI, these queries are dealt with the modules SNMPHandler and CLIHandler which directly talk with the switches. The SNMPListener is to listen to snmp trap such as link up/down event or switch on/off event.

Design

In terms of the architecture of the SNMP4SDN Plugin's features, the features include flow configuration, topology discovery, and multi-vendor support. Their architectures please refer to Wiki (Developer Guide - Design).

Installation and Configuration Guide

- Please refer to the *Getting Started Guide* in https://www.opendaylight.org/downloads, find the SNMP4SDN section.
- For the latest full guide, please refer to Wiki (Installation Guide, User Guide - Configuration).

Tutorial

- For the latest full guide, please refer to Wiki (User Guide - Tutorial).

Programmatic Interface(s)

SNMP4SDN Plugin exposes APIs via MD-SAL with YANG model. The methods (RPC call) and data structures for them are listed below.

TopologyService

- RPC call
  - get-edge-list
  - get-node-list
  - get-node-connector-list
  - set-discovery-interval (given interval time in seconds)
  - rediscover

- Data structure
  - node: composed of node-id, node-type
  - node-connector: composed of node-connector-id, node-connector-type, node
  - topo-edge: composed of head-node-connector-id, head-node-connector-type, head-node-id, head-node-type, tail-node-connector-id, tail-node-connector-type, tail-node-id, tail-node-type

VlanService

- RPC call
  - add-vlan (given node ID, VLAN ID, VLAN name)
  - add-vlan-and-set-ports (given node ID, VLAN ID, VLAN name, tagged ports, untagged ports)
  - set-vlan-ports (given node ID, VLAN ID, tagged ports, untagged ports)
  - delete-vlan (given node ID, VLAN ID)
  - get-vlan-table (given node ID)

AclService

- RPC call
  - create-acl-profile (given node ID, acl-profile-index, acl-profile)
  - del-acl-profile (given node ID, acl-profile-index)
  - set-acl-rule (given node ID, acl-index, acl-rule)
  - del-acl-rule (given node ID, acl-index)
  - clear-acl-table (given node ID)
- Data structure
  - acl-profile-index: composed of profile-id, profile name
  - acl-profile: composed of acl-layer, vlan-mask, src-ip-mask, dst-ip-mask
  - acl-layer: IP or ETHERNET
  - acl-index: composed of acl-profile-index, acl-rule-index
  - acl-rule-index: composed of rule-id, rule-name
  - acl-rule: composed of port-list, acl-layer, acl-field, acl-action
  - acl-field: composed of vlan-id, src-ip, dst-ip
  - acl-action: PERMIT or DENY

FdbService

- RPC call
  - set-fdb-entry (given fdb-entry)
  - del-fdb-entry (given node-id, vlan-id, dest-mac-adddr)

  - get-fdb-entry (given node-id, vlan-id, dest-mac-adddr)
  - get-fdb-table (given node-id)
- Data structure
  - fdb-entry: composed of node-id, vlan-id, dest-mac-addr, port, fdb-entry-type
  - fdb-entry-type: OTHER/INVALID/LEARNED/SELF/MGMT

MiscConfigService

- RPC call
  - set-stp-port-state (given node-id, port, is_nable)
  - get-stp-port-state (given node-id, port)
  - get-stp-port-root (given node-id, port)
  - enable-stp (given node-id)
  - disable-stp (given node-id)
  - delete-arp-entry (given node-id, ip-address)
  - set-arp-entry (given node-id, arp-entry)
  - get-arp-entry (given node-id, ip-address)
  - get-arp-table (given node-id)
- Data structure
  - stp-port-state: DISABLE/BLOCKING/LISTENING/LEARNING/FORWARDING/BROKEN
  - arp-entry: composed of ip-address and mac-address

SwitchDbService

- RPC call
  - reload-db (The following 4 RPC implemention is TBD)
  - add-switch-entry
  - delete-switch-entry
  - clear-db
  - update-db
- Data structure
  - switch-info: compose of node-ip, node-mac, community, cli-user-name, cli-password, model

Help

- SNMP4SDN Wiki
- SNMP4SDN Mailing List (user, developer)
- Latest troubleshooting in Wiki

SXP Developer Guide

Overview

SXP (Scalable-Group Tag eXchange Protocol) project is an effort to enhance OpenDaylight platform with IP-SGT (IP Address to Source Group Tag) bindings that can be learned from connected SXP-aware network nodes. The current implementation supports SXP protocol version 4 according to the Smith, Kandula - SXP IETF draft and grouping of peers and creating filters based on ACL/Prefix-list syntax for filtering outbound and inbound IP-SGT bindings. All protocol legacy versions 1-3 are supported as well. Additionally, version 4 adds bidirectional connection type as an extension of a unidirectional one.

SXP Architecture

The SXP Server manages all connected clients in separate threads and a common SXP protocol agreement is used between connected peers. Each SXP network peer is modelled with its pertaining class, e.g., SXP Server represents the SXP Speaker, SXP Listener the Client. The server program creates the ServerSocket object on a specified port and waits until a client starts up and requests connect on the IP address and port of the server. The client program opens a Socket that is connected to the server running on the specified host IP address and port.

The SXP Listener maintains connection with its speaker peer. From an opened channel pipeline, all incoming SXP messages are processed by various handlers. Message must be decoded, parsed and validated.

The SXP Speaker is a counterpart to the SXP Listener. It maintains a connection with its listener peer and sends composed messages.

The SXP Binding Handler extracts the IP-SGT binding from a message and pulls it into the SXP-Database. If an error is detected during the IP-SGT extraction, an appropriate error code and sub-code is selected and an error message is sent back to the connected peer. All transitive messages are routed directly to the output queue of SXP Binding Dispatcher.

The SXP Binding Dispatcher represents a selector that will decides how many data from the SXP-database will be sent and when. It is responsible for message content composition based on maximum message length.

The SXP Binding Filters handles filtering of outgoing and incoming IP-SGT bindings according to BGP filtering using ACL and Prefix List syntax for specifying filter or based on Peer-sequence length.

The SXP Domains feature provides isolation of SXP peers and bindings learned between them, also exchange of Bindings is possible across SXP-Domains by ACL, Prefix List or Peer-Sequence filters

Key APIs and Interfaces

As this project is fairly small, it provides only few features that install and provide all APIs and implementations for this project.

- sxp-route
- sxp-controller
- sxp-api
- spx-core

sxp-route

Performs managing of SXP devices in cluster environment

sxp-controller

RPC request handling

sxp-api

Contains data holders and entities

spx-core

Main logic and core features

API Reference Documentation

RESTCONF Interface and Dynamic Tree Specification and Architecture

Topology Processing Framework Developer Guide

Overview

The Topology Processing Framework allows developers to aggregate and filter topologies according to defined correlations. It also provides functionality, which you can use to make your own topology model by automating the translation from one model to another. For example to translate from the opendaylight-inventory model to only using the network-topology model.

Architecture

Chapter Overview

In this chapter we describe the architecture of the Topology Processing Framework. In the first part, we provide information about available features and basic class relationships. In the second part, we describe our model specific approach, which is used to provide support for different models.

Basic Architecture

The Topology Processing Framework consists of several Karaf features:

- odl-topoprocessing-framework
- odl-topoprocessing-inventory
- odl-topoprocessing-network-topology
- odl-topoprocessing-i2rs
- odl-topoprocessing-inventory-rendering

The feature odl-topoprocessing-framework contains the topoprocessing-api, topoprocessing-spi and topoprocessing-impl bundles. This feature is the core of the Topology Processing Framework and is required by all others features.

- topoprocessing-api - contains correlation definitions and definitions required for rendering

- topoprocessing-spi - entry point for topoprocessing service (start and close)
- topoprocessing-impl - contains base implementations of handlers, listeners, aggregators and filtrators

TopoProcessingProvider is the entry point for Topology Processing Framework. It requires a DataBroker instance. The DataBroker is needed for listener registration. There is also the TopologyRequestListener which listens on aggregated topology requests (placed into the configuration datastore) and UnderlayTopologyListeners which listen on underlay topology data changes (made in operational datastore). The TopologyRequestHandler saves toporequest data and provides a method for translating a path to the specified leaf. When a change in the topology occurs, the registered UnderlayTopologyListener processes this information for further aggregation and/or filtration. Finally, after an overlay topology is created, it is passed to the TopologyWriter, which writes this topology into operational datastore.

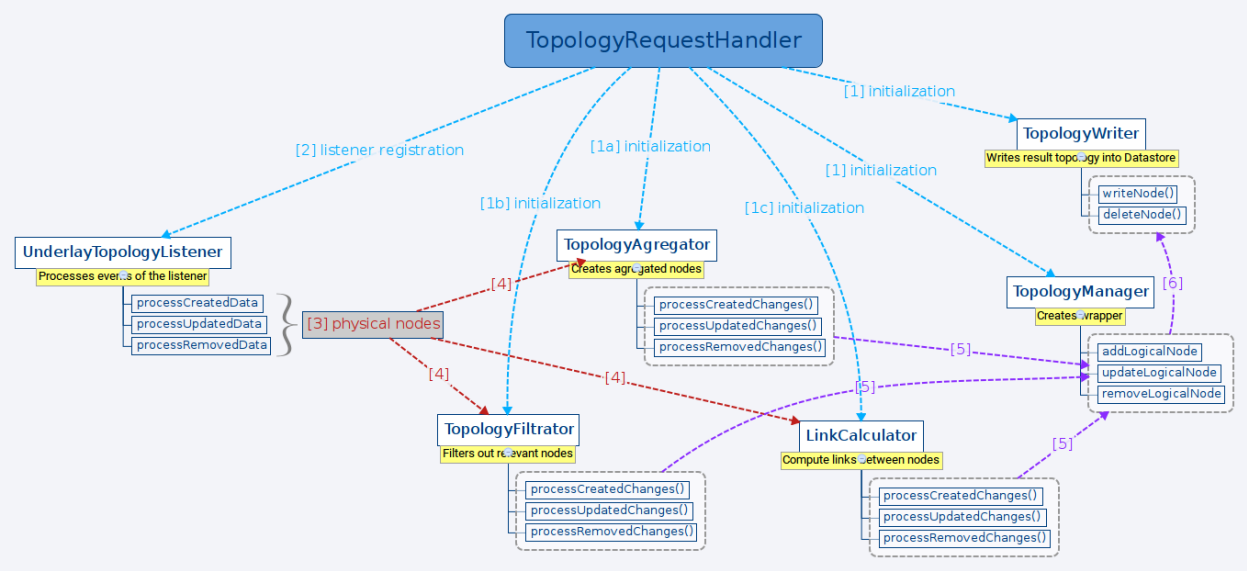


Fig. 2.65: Class relationship

[1] TopologyRequestHandler instantiates TopologyWriter and TopologyManager. Then, according to the request, initializes either TopologyAggregator, TopologyFiltrator or LinkCalculator.

[2] It creates as many instances of UnderlayTopologyListener as there are underlay topologies.

[3] PhysicalNodes are created for relevant incoming nodes (those having node ID).

[4a] It performs aggregation and creates logical nodes.

[4b] It performs filtration and creates logical nodes.

[4c] It performs link computation and creates links between logical nodes.

[5] Logical nodes are put into wrapper.

[6] The wrapper is translated into the appropriate format and written into datastore.

Model Specific Approach

The Topology Processing Framework consists of several modules and Karaf features, which provide support for different input models. Currently we support the network-topology, opendaylight-inventory and i2rs models. For each of these input models, the Topology Processing Framework has one module and one Karaf feature.

How it works

User point of view:

When you start the odl-topoprocessing-framework feature, the Topology Processing Framework starts without knowledge how to work with any input models. In order to allow the Topology Processing Framework to process some kind of input model, you must install one (or more) model specific features. Installing these features will also start odl-topoprocessing-framework feature if it is not already running. These features inject appropriate logic into the odl-topoprocessing-framework feature. From that point, the Topology Processing Framework is able to process different kinds of input models, specifically those that you install features for.

Developer point of view:

The topoprocessing-impl module contains (among other things) classes and interfaces, which are common for every model specific topoprocessing module. These classes and interfaces are implemented and extended by classes in particular model specific modules. Model specific modules also depend on the TopoProcessingProvider class in the topoprocessing-spi module. This dependency is injected during installation of model specific features in Karaf. When a model specific feature is started, it calls the registerAdapters(adapters) method of the injected TopoProcessingProvider object. After this step, the Topology Processing Framework is able to use registered model adapters to work with input models.

To achieve the described functionality we created a ModelAdapter interface. It represents installed feature and provides methods for creating crucial structures specific to each model.

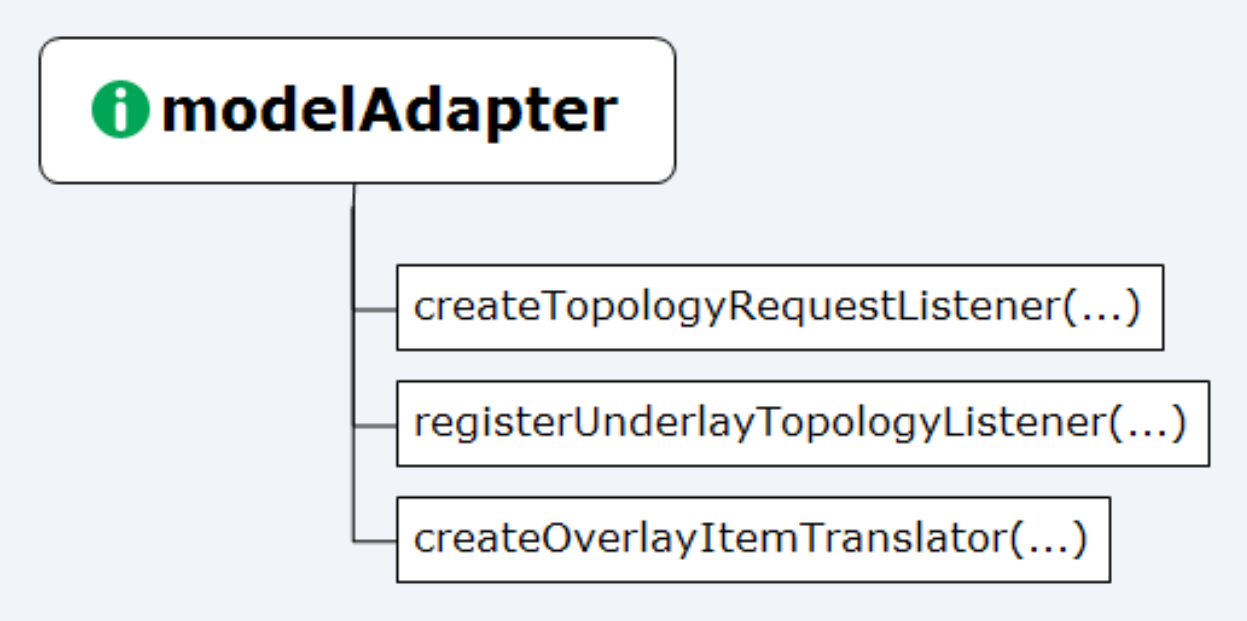


Fig. 2.66: ModelAdapter interface

Model Specific Features

- odl-topoprocessing-network-topology - this feature contains logic to work with network-topology model
- odl-topoprocessing-inventory - this feature contains logic to work with opendaylight-inventory model
- odl-topoprocessing-i2rs - this feature contains logic to work with i2rs model

Inventory Model Support

The opendaylight-inventory model contains only nodes, termination points, information regarding these structures. This model co-operates with network-topology model, where other topology related information is stored. This means that we have to handle two input models at once. To support the inventory model, InventoryListener and NotificationInterConnector classes were introduced. Please see the flow diagrams below.

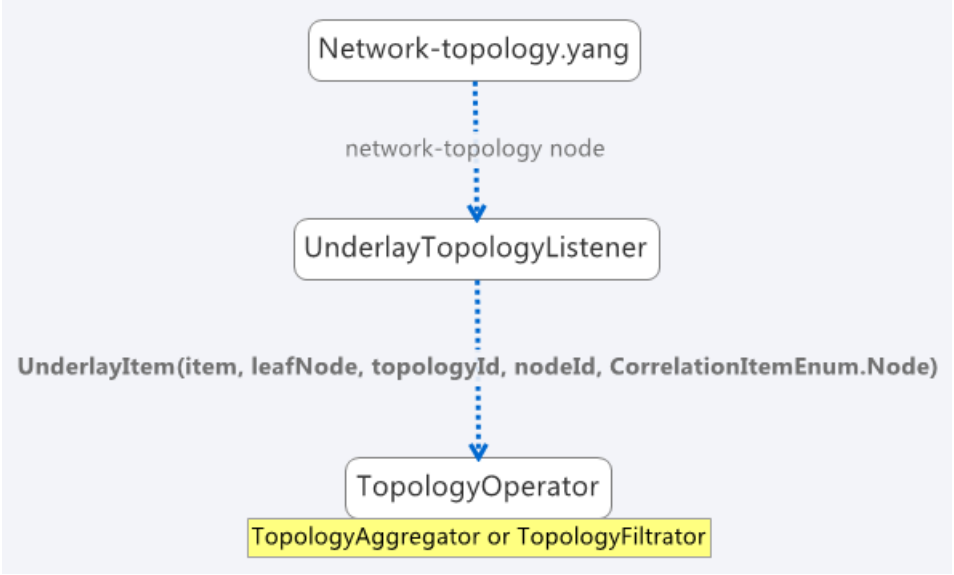


Fig. 2.67: Network topology model

Here we can see the InventoryListener and NotificationInterConnector classes. InventoryListener listens on data changes in the inventory model and passes these changes wrapped as an UnderlayItem for further processing to NotificationInterConnector. It doesn't contain node information - it contains a leafNode (node based on which aggregation occurs) instead. The node information is stored in the topology model, where UnderlayTopologyListener is registered as usual. This listener delivers the missing information.

Then the NotificationInterConnector combines the two notifications into a complete UnderlayItem (no null values) and delivers this UnderlayItem for further processing (to next TopologyOperator).

Aggregation and Filtration

Chapter Overview

The Topology Processing Framework allows the creation of aggregated topologies and filtered views over existing topologies. Currently, aggregation and filtration is supported for topologies that follow network-topology, opendaylight-inventory or i2rs model. When a request to create an aggregated or filtered topology is received, the framework creates one listener per underlay topology. Whenever any specified underlay topology is changed, the appropriate listener is triggered with the change and the change is processed. Two types of correlations (functionalities) are currently supported:

- Aggregation
  - Unification
  - Equality
- Filtration

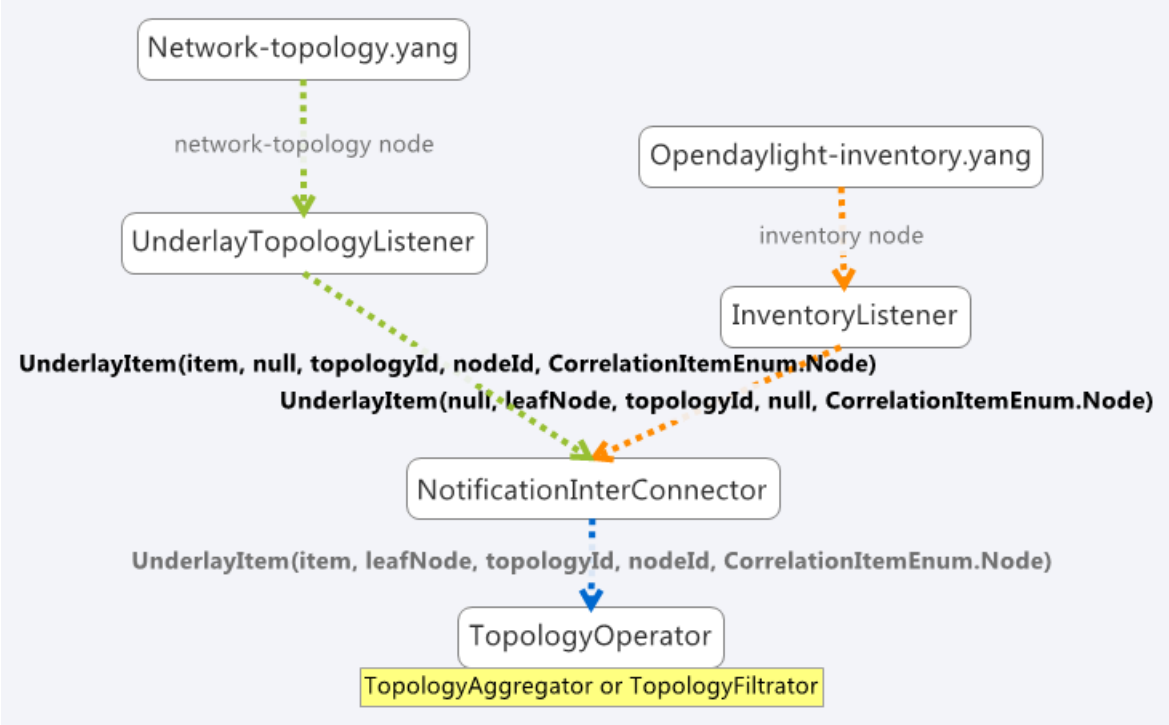


Fig. 2.68: Inventory model

Terminology

We use the term underlay item (physical node) for items (nodes, links, termination-points) from underlay and overlay item (logical node) for items from overlay topologies regardless of whether those are actually physical network elements.

Aggregation

Aggregation is an operation which creates an aggregated item from two or more items in the underlay topology if the aggregation condition is fulfilled. Requests for aggregated topologies must specify a list of underlay topologies over which the overlay (aggregated) topology will be created and a target field in the underlay item that the framework will check for equality.

Create Overlay Node

First, each new underlay item is inserted into the proper topology store. Once the item is stored, the framework compares it (using the target field value) with all stored underlay items from underlay topologies. If there is a target-field match, a new overlay item is created containing pointers to all *equal* underlay items. The newly created overlay item is also given new references to its supporting underlay items.

Equality case:

If an item doesn't fulfill the equality condition with any other items, processing finishes after adding the item into topology store. It will stay there for future use, ready to create an aggregated item with a new underlay item, with which it would satisfy the equality condition.

Unification case:

An overlay item is created for all underlay items, even those which don't fulfill the equality condition with any other items. This means that an overlay item is created for every underlay item, but for items which satisfy the equality condition, an aggregated item is created.

Update Node

Processing of updated underlay items depends on whether the target field has been modified. If yes, then:

- if the underlay item belonged to some overlay item, it is removed from that item. Next, if the aggregation condition on the target field is satisfied, the item is inserted into another overlay item. If the condition isn't met then:
  - in equality case - the item will not be present in overlay topology.
  - in unification case - the item will create an overlay item with a single underlay item and this will be written into overlay topology.
- if the item didn't belong to some overlay item, it is checked again for aggregation with other underlay items.

Remove Node

The underlay item is removed from the corresponding topology store, from it's overlay item (if it belongs to one) and this way it is also removed from overlay topology.

Equality case:

If there is only one underlay item left in the overlay item, the overlay item is removed.

Unification case:

The overlay item is removed once it refers to no underlay item.

Filtration

Filtration is an operation which results in creation of overlay topology containing only items fulfilling conditions set in the topoprocessing request.

Create Underlay Item

If a newly created underlay item passes all filtrators and their conditions, then it is stored in topology store and a creation notification is delivered into topology manager. No operation otherwise.

Update Underlay Item

First, the updated item is checked for presence in topology store:

- if it is present in topology store:
  - if it meets the filtering conditions, then processUpdatedData notification is triggered
  - else processRemovedData notification is triggered
- if item isn't present in topology store
  - if item meets filtering conditions, then processCreatedData notification is triggered

- else it is ignored

Remove Underlay Item

If an underlay node is supporting some overlay node, the overlay node is simply removed.

Default Filtrator Types

There are seven types of default filtrators defined in the framework:

- IPv4-address filtrator - checks if specified field meets IPv4 address + mask criteria
- IPv6-address filtrator - checks if specified field meets IPv6 address + mask criteria
- Specific number filtrator - checks for specific number
- Specific string filtrator - checks for specific string
- Range number filtrator - checks if specified field is higher than provided minimum (inclusive) and lower than provided maximum (inclusive)
- Range string filtrator - checks if specified field is alphabetically greater than provided minimum (inclusive) and alphabetically lower than provided maximum (inclusive)
- Script filtrator - allows a user or application to implement their own filtrator

Register Custom Filtrator

There might be some use case that cannot be achieved with the default filtrators. In these cases, the framework offers the possibility for a user or application to register a custom filtrator.

Pre-Filtration / Filtration & Aggregation

This feature was introduced in order to lower memory and performance demands. It is a combination of the filtration and aggregation operations. First, uninteresting items are filtered out and then aggregation is performed only on items that passed filtration. This way the framework saves on compute time. The PreAggregationFiltrator and TopologyAggregator share the same TopoStoreProvider (and thus topology store) which results in lower memory demands (as underlay items are stored only in one topology store - they aren't stored twice).

Link Computation

Chapter Overview

While processing the topology request, we create overlay nodes with lists of supporting underlay nodes. Because these overlay nodes have completely new identifiers, we lose link information. To regain this link information, we provide Link Computation functionality. Its main purpose is to create new overlay links based on the links from the underlay topologies and underlay items from overlay items. The required information for Link Computation is provided via the Link Computation model in (topology-link-computation.yang).

Link Computation Functionality

Let us consider two topologies with following components:

Topology 1:

- Node: `node:1:1`
- Node: `node:1:2`
- Node: `node:1:3`
- Link: `link:1:1` (from `node:1:1` to `node:1:2`)
- Link: `link:1:2` (from `node:1:3` to `node:1:2`)

Topology 2:

- Node: `node:2:1`
- Node: `node:2:2`
- Node: `node:2:3`
- Link: `link:2:1` (from `node:2:1` to `node:2:3`)

Now let's say that we applied some operations over these topologies that results into aggregating together

- `node:1:1` and `node:2:3` (`node:1`)
- `node:1:2` and `node:2:2` (`node:2`)
- `node:1:3` and `node:2:1` (`node:3`)

At this point we can no longer use available links in new topology because of the node ID change, so we must create new overlay links with source and destination node set to new nodes IDs. It means that `link:1:1` from topology 1 will create new link `link:1`. Since original source (`node:1:1`) is already aggregated under `node:1`, it will become source node for `link:1`. Using same method the destination will be `node:2`. And the final output will be three links:

- `link:1`, from `node:1` to `node:2`
- `link:2`, from `node:3` to `node:2`
- `link:3`, from `node:3` to `node:1`

In-Depth Look

The main logic behind Link Computation is executed in the LinkCalculator operator. The required information is passed to LinkCalculator through the LinkComputation section of the topology request. This section is defined in the topology-link-computation.yang file. The main logic also covers cases when some underlay nodes may not pass through other topology operators.

Link Computation Model

There are three essential pieces of information for link computations. All of them are provided within the LinkComputation section. These pieces are:

- output model
- overlay topology with new nodes
- underlay topologies with original links

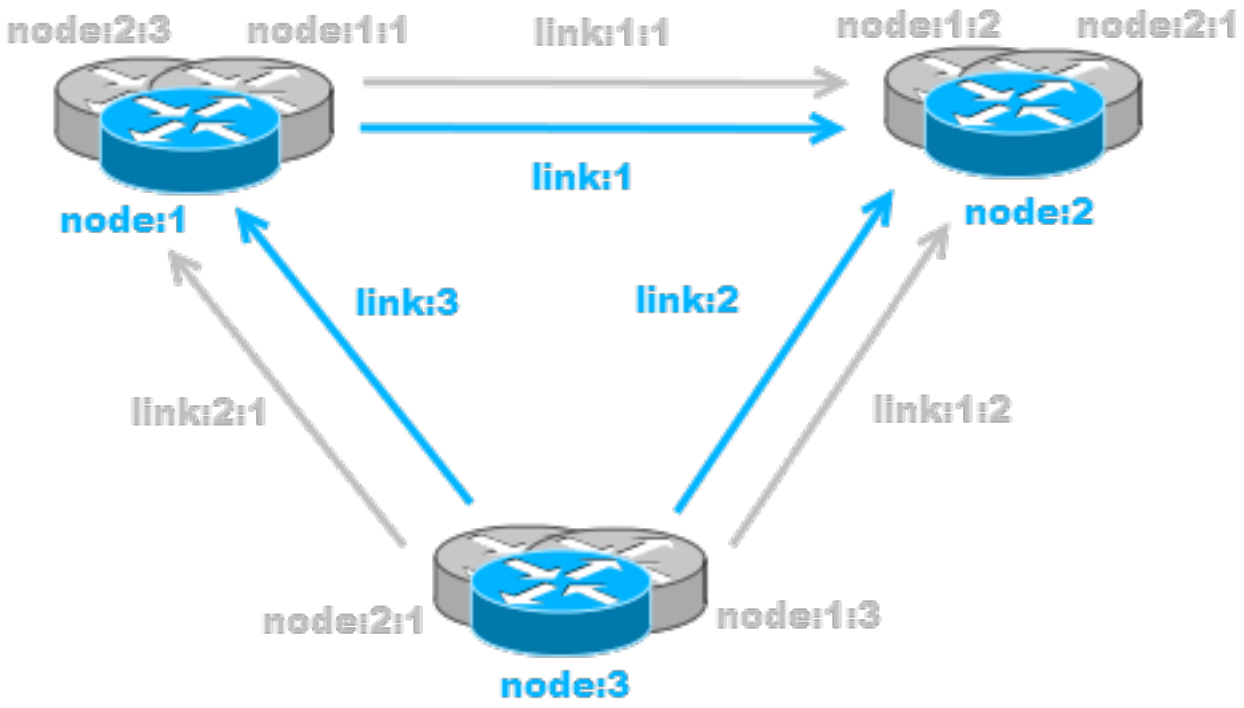


Fig. 2.69: Overlay topology with computed links

This whole section is augmented into `network-topology:topology`. By placing this section out of correlations section, it allows us to send link computation request separately from topology operations request.

Main Logic

Taking into consideration that some of the underlay nodes may not transform into overlay nodes (e.g. they are filtered out), we created two possible states for links:

- matched - a link is considered as matched when both original source and destination node were transformed to overlay nodes
- waiting - a link is considered as waiting if original source, destination or both nodes are missing from the overlay topology

All links in waiting the state are stored in waitingLinks list, already matched links are stored in matchedLinks list and overlay nodes are stored in the storedOverlayNodes list. All processing is based only on information in these lists. Processing created, updated and removed underlay items is slightly different and described in next sections separately.

Processing Created Items

Created items can be either nodes or links, depending on the type of listener from which they came. In the case of a link, it is immediately added to waitingLinks and calculation for possible overlay link creations (calculatePossibleLink) is started. The flow diagram for this process is shown in the following picture:

Searching for the source and destination nodes in the calculatePossibleLink method runs over each node in storedOverlayNodes and the IDs of each supporting node is compared against IDs from the underlay link's source and destination nodes. If there are any nodes missing, the link remains in the waiting state. If both the source and destination nodes are found, the corresponding overlay nodes is recorded as the new source and destination. The link is then removed from waitingLinks and a new CalculatedLink is added to the matched links. At the end, the new link (if it exists) is written into the datastore.

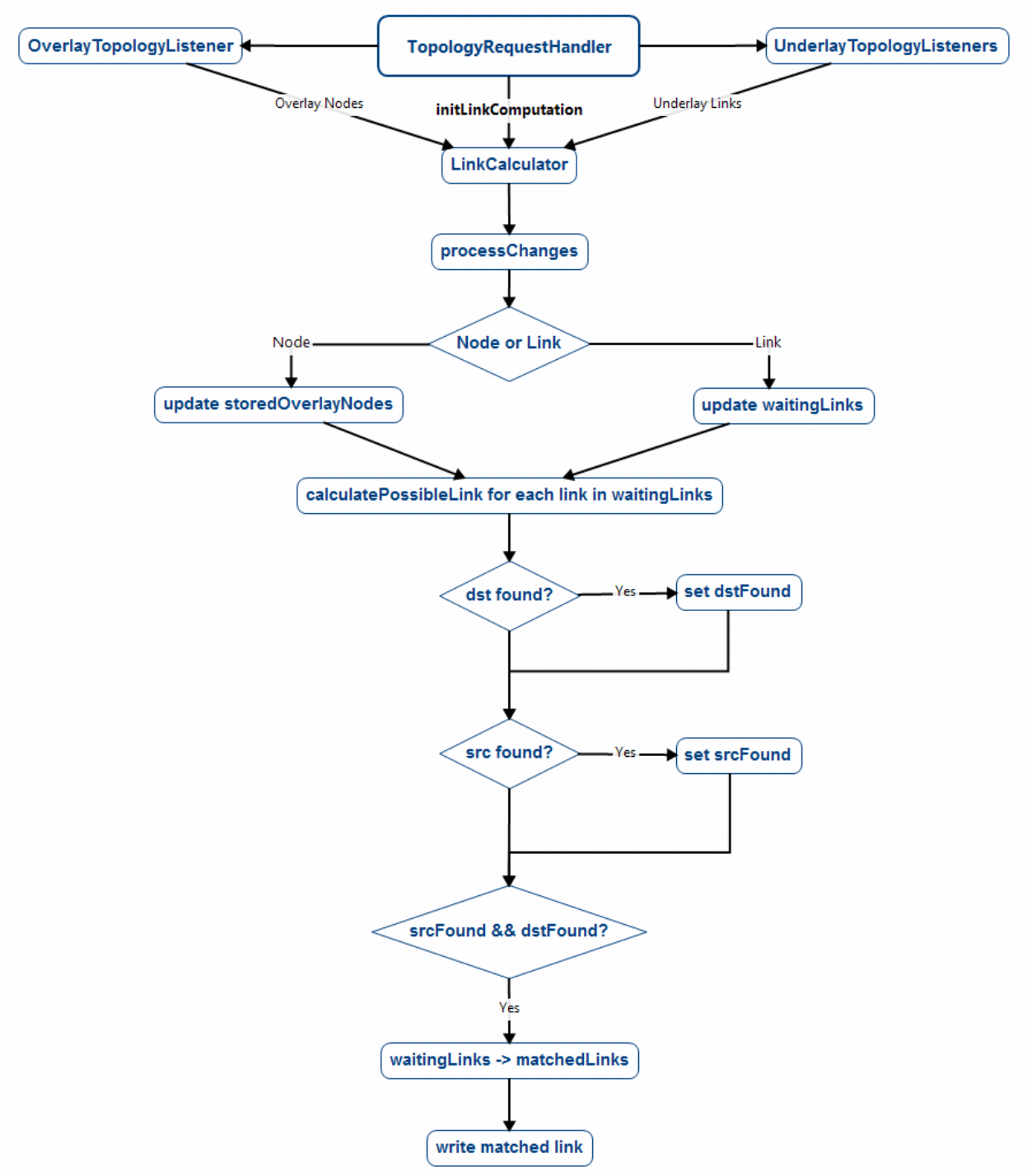


Fig. 2.70: Flow diagram of processing created items

If the created item is an overlayNode, this is added to storedOverlayNodes and we call calculatePossibleLink for every link in waitingLinks.

Processing Updated Items

The difference from processing created items is that we have three possible types of updated items: overlay nodes, waiting underlay links, and matched underlay links.

- In the case of a change in a matched link, this must be recalculated and based on the result it will either be matched with new source and destination or will be returned to waiting links. If the link is moved back to a waiting state, it must also be removed from the datastore.
- In the case of change in a waiting link, it is passed to the calculation process and based on the result will either remain in waiting state or be promoted to the matched state.
- In the case of a change in an overlay node, storedOverlayNodes must be updated properly and all links must be recalculated in case of changes.

Processing Removed items

Same as for processing updated item. There can be three types of removed items:

- In case of waiting link removal, the link is just removed from waitingLinks
- In case of matched link removal, the link is removed from matchingLinks and datastore
- In case of overlay node removal, the node must be removed form storedOverlayNodes and all matching links must be recalculated

Wrapper, RPC Republishing, Writing Mechanism

Chapter Overview

During the process of aggregation and filtration, overlay items (so called logical nodes) were created from underlay items (physical nodes). In the topology manager, overlay items are put into a wrapper. A wrapper is identified with unique ID and contains list of logical nodes. Wrappers are used to deal with transitivity of underlay items - which permits grouping of overlay items (into wrappers).

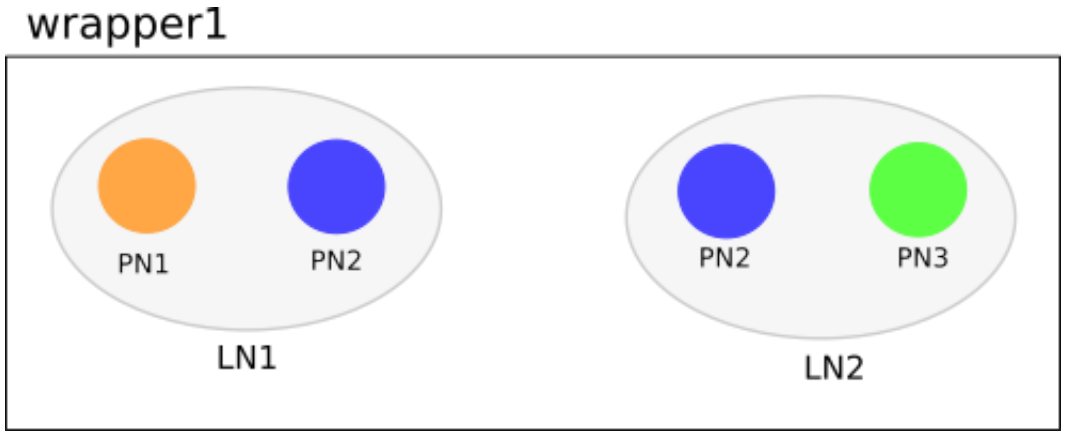


Fig. 2.71: Wrapper

PN1, PN2, PN3 = physical nodes

LN1, LN2 = logical nodes

RPC Republishing

All RPCs registered to handle underlay items are re-registered under their corresponding wrapper ID. RPCs of underlay items (belonging to an overlay item) are gathered, and registered under ID of their wrapper.

RPC Call

When RPC is called on overlay item, this call is delegated to it's underlay items, this means that the RPC is called on all underlay items of this overlay item.

Writing Mechanism

When a wrapper (containing overlay item(s) with it's underlay item(s)) is ready to be written into data store, it has to be converted into DOM format. After this translation is done, the result is written into datastore. Physical nodes are stored as supporting-nodes. In order to use resources responsibly, writing operation is divided into two steps. First, a set of threads registers prepared operations (deletes and puts) and one thread makes actual write operation in batch.

Topology Rendering Guide - Inventory Rendering

Chapter Overview

In the most recent OpenDaylight release, the opendaylight-inventory model is marked as deprecated. To facilitate migration from it to the network-topology model, there were requests to render (translate) data from inventory model (whether augmented or not) to another model for further processing. The Topology Processing Framework was extended to provide this functionality by implementing several rendering-specific classes. This chapter is a step-by-step guide on how to implement your own topology rendering using our inventory rendering as an example.

Use case

For the purpose of this guide we are going to render the following augmented fields from the OpenFlow model:

- from inventory node:
  - manufacturer
  - hardware
  - software
  - serial-number
  - description
  - ip-address
- from inventory node-connector:
  - name
  - hardware-address
  - current-speed
  - maximum-speed

We also want to preserve the node ID and termination-point ID from opendaylight-topology-inventory model, which is network-topology part of the inventory model.

Implementation

There are two ways to implement support for your specific topology rendering:

- add a module to your project that depends on the Topology Processing Framework
- add a module to the Topology Processing Framework itself

Regardless, a successful implementation must complete all of the following steps.

Step1 - Target Model Creation

Because the network-topology node does not have fields to store all desired data, it is necessary to create new model to render this extra data in to. For this guide we created the inventory-rendering model. The picture below shows how data will be rendered and stored.

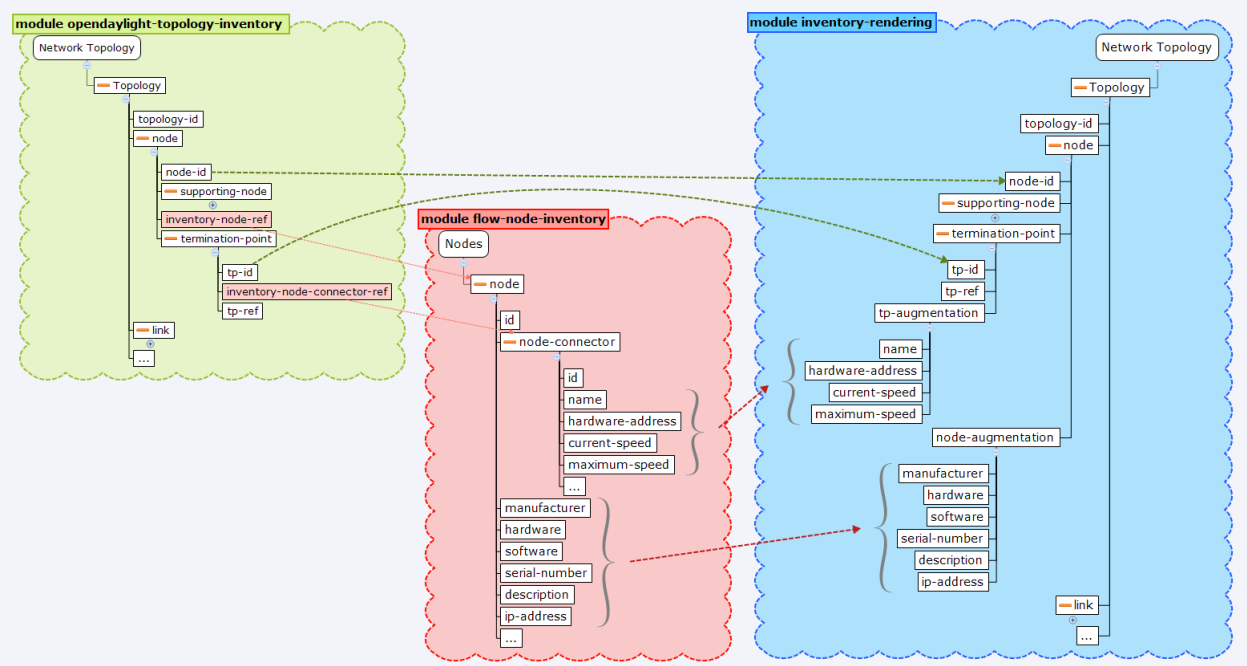


Fig. 2.72: Rendering to the inventory-rendering model

Important: When implementing your version of the topology-rendering model in the Topology Processing Framework, the source file of the model (.yang) must be saved in /topoprocessing-api/src/main/yang folder so corresponding structures can be generated during build and can be accessed from every module through dependencies.

When the target model is created you have to add an identifier through which you can set your new model as output model. To do that you have to add another identity item to topology-correlation.yang file. For our inventory-rendering model identity looks like this:

After that you will be able to set inventory-rendering-model as output model in XML.

Step2 - Module and Feature Creation

Important: This and following steps are based on the *model specific approach* in the Topology Processing Framework. We highly recommend that you familiarize yourself with this approach in advance.

To create a base module and add it as a feature to Karaf in the Topology Processing Framework we made the changes in following commit. Changes in other projects will likely be similar.

File	Changes
pom.xml	add new module to topoprocessing
features.xml	add feature to topoprocessing
features/pom.xml	add dependencies needed by features
topoprocessing-artifacts/pom.xml	add artifact
topoprocessing-config/pom.xml	add configuration file
81-topoprocessing-inventory-renderin g-config.xml	configuration file for new module
topoprocessing-inventory-rendering/p om.xml	main pom for new module
TopoProcessingProviderIR.java	contains startup method which register new model adapter
TopoProcessingProviderIRMod-ule.java	generated class which contains createInstance method. You should call your startup method from here.
TopoProcessingProviderIRModule-Factor y.java	generated class. You will probably not need to edit this file
log4j.xml	configuration file for logger topoprocessing-inventory-rendering-p rovider-impl.yang

Step3 - Module Adapters Creation

There are seven mandatory interfaces or abstract classes that needs to be implemented in each module. They are:

- TopoProcessingProvider - provides module registration
- ModelAdapter - provides model specific instances
- TopologyRequestListener - listens on changes in the configuration datastore
- TopologyRequestHandler - processes configuration datastore changes
- UnderlayTopologyListener - listens for changes in the specific model
- LinkTransaltor and NodeTranslator - used by OverlayItemTranslator to create NormalizedNodes from OverlayItems

The name convention we used was to add an abbreviation for the specific model to the beginning of implementing class name (e.g. the IRModelAdapter refers to class which implements ModelAdapter in module Inventory Rendering). In the case of the provider class, we put the abbreviation at the end.

Important:

- In the next sections, we use the terms TopologyRequestListener, TopologyRequestHandler, etc. without a prepended or appended abbreviation because the steps apply regardless of which specific model you are targeting.

- If you want to implement rendering from inventory to network-topology, you can just copy-paste our module and additional changes will be required only in the output part.

Provider part

This part is the starting point of the whole module. It is responsible for creating and registering TopologyRequestListeners. It is necessary to create three classes which will import:

- **TopoProcessingProviderModule** - is a generated class from topoprocessing-inventory-rendering-provider-impl.yang (created in previous step, file will appear after first build). Its method `createInstance()` is called at the feature start and must be modified to create an instance of TopoProcessingProvider and call its `startup(TopoProcessingProvider topoProvider)` function.
- **TopoProcessingProvider** - in `startup(TopoProcessingProvider topoProvider)` function provides ModelAdapter registration to TopoProcessingProviderImpl.
- **ModelAdapter** - provides creation of corresponding module specific classes.

Input part

This includes the creation of the classes responsible for input data processing. In this case, we had to create five classes implementing:

- **TopologyRequestListener** and **TopologyRequestHandler** - when notified about a change in the configuration datastore, verify if the change contains a topology request (has correlations in it) and creates UnderlayTopologyListeners if needed. The implementation of these classes will differ according to the model in which are correlations saved (network-topology or i2rs). In the case of using network-topology, as the input model, you can use our classes IRTopologyRequestListener and IRTopologyRequestHandler.
- **UnderlayTopologyListener** - registers underlay listeners according to input model. In our case (listening in the inventory model), we created listeners for the network-topology model and inventory model, and set the NotificationInterConnector as the first operator and set the IRRenderingOperator as the second operator (after NotificationInterConnector). Same as for TopologyRequestListener/Handler, if you are rendering from the inventory model, you can use our class IRUnderlayTopologyListener.
- **InventoryListener** - a new implementation of this class is required only for inventory input model. This is because the InventoryListener from topoprocessing-impl requires pathIdentifier which is absent in the case of rendering.
- **TopologyOperator** - replaces classic topoprocessing operator. While the classic operator provides specific operations on topology, the rendering operator just wraps each received UnderlayItem to OverlayItem and sends them to write.

Important: For purposes of topology rendering from inventory to network-topology, there are misused fields in UnderlayItem as follows:

- item - contains node from network-topology part of inventory
- leafItem - contains node from inventory

In case of implementing UnderlayTopologyListener or InventoryListener you have to carefully adjust UnderlayItem creation to these terms.

Output part

The output part of topology rendering is responsible for translating received overlay items to normalized nodes. In the case of inventory rendering, this is where node information from inventory are combined with node information from network-topology. This combined information is stored in our inventory-rendering model normalized node and passed to the writer.

The output part consists of two translators implementing the NodeTranslator and LinkTranslator interfaces.

NodeTranslator implementation - The NodeTranslator interface has one `translate(OverlayItemWrapper wrapper)` method. For our purposes, there is one important thing in wrapper - the list of OverlayItems which have one or more common UnderlayItems. Regardless of this list, in the case of rendering it will always contains only one OverlayItem. This item has list of UnderlayItems, but again in case of rendering there will be only one UnderlayItem item in this list. In NodeTranslator, the OverlayItem and corresponding UnderlayItem represent nodes from the translating model.

The UnderlayItem has several attributes. How you will use these attributes in your rendering is up to you, as you create this item in your topology operator. For example, as mentioned above, in our inventory rendering example is an inventory node normalized node stored in the UnderlayItem leafNode attribute, and we also store node-id from network-topology model in UnderlayItem itemId attribute. You can now use these attributes to build a normalized node for your new model. How to read and create normalized nodes is out of scope of this document.

LinkTranslator implementation - The LinkTranslator interface also has one `translate(OverlayItemWrapper wrapper)` method. In our inventory rendering this method returns `null`, because the inventory model doesn't have links. But if you also need links, this is the place where you should translate it into a normalized node for your model. In LinkTranslator, the OverlayItem and corresponding UnderlayItem represent links from the translating model. As in NodeTranslator, there will be only one OverlayItem and one UnderlayItem in the corresponding lists.

Testing

If you want to test topoprocessing with some manually created underlay topologies (like in this guide), than you have to tell Topoprocessing to listen for underlay topologies on Configuration datastore instead of Operational.

You can do this in this config file
`<topoprocessing_directory>/topoprocessing-config/src/main/resources/80-topoprocessing-config.xml`.
Here you have to change
`<datastore-type>OPERATIONAL</datastore-type>`
to
`<datastore-type>CONFIGURATION</datastore-type>`.

Also you have to add dependency required to test "inventory" topologies.

In `<topoprocessing_directory>/features/pom.xml`
add `<openflowplugin.version>latest_snapshot</openflowplugin.version>` to properties section
and add this dependency to dependencies section

```
<dependency>
        <groupId>org.opendaylight.openflowplugin</groupId>
        <artifactId>features-openflowplugin</artifactId>
        <version>${openflowplugin.version}</version>
        <classifier>features</classifier><type>xml</type>
</dependency>
```

`latest_snapshot` in `<openflowplugin.version>` replace with latest snapshot, which can be found here.

And in `<topoprocessing_directory>/features/src/main/resources/features.xml` add `<repository>mvn:org.opendaylight.openflowplugin/features-openflowplugin/${openflowplugin.version}/xml/features</repository>` to repositories section.

Now after you rebuild project and start Karaf, you can install necessary features.

You can install all with one command:

```
feature:install odl-restconf-noauth odl-topoprocessing-inventory-rendering
odl-openflowplugin-southbound odl-openflowplugin-nsf-model
```

Now you can send messages to REST from any REST client (e.g. Postman in Chrome). Messages have to have following headers:

Header	Value
Content-Type:	application/xml
Accept:	application/xml
username:	admin
password:	admin

Firstly send topology request to http://localhost:8181/restconf/config/network-topology:network-topology/topology/render:1 with method PUT. Example of simple rendering request:

```
<topology xmlns="urn:TBD:params:xml:ns:yang:network-topology">
  <topology-id>render:1</topology-id>
    <correlations xmlns="urn:opendaylight:topology:correlation" >
      <output-model>inventory-rendering-model</output-model>
      <correlation>
         <correlation-id>1</correlation-id>
          <type>rendering-only</type>
          <correlation-item>node</correlation-item>
          <rendering>
            <underlay-topology>und-topo:1</underlay-topology>
        </rendering>
      </correlation>
    </correlations>
</topology>
```

This request says that we want create topology with name render:1 and this topology should be stored in the inventory-rendering-model and it should be created from topology flow:1 by node rendering.

Next we send the network-topology part of topology flow:1. So to the URL http://localhost:8181/restconf/config/network-topology:network-topology/topology/und-topo:1 we PUT:

```
<topology xmlns="urn:TBD:params:xml:ns:yang:network-topology"
          xmlns:it="urn:opendaylight:model:topology:inventory"
          xmlns:i="urn:opendaylight:inventory">
    <topology-id>und-topo:1</topology-id>
    <node>
        <node-id>openflow:1</node-id>
        <it:inventory-node-ref>
    /i:nodes/i:node[i:id="openflow:1"]
        </it:inventory-node-ref>
        <termination-point>
```

```
            <tp-id>tp:1</tp-id>
            <it:inventory-node-connector-ref>
                /i:nodes/i:node[i:id="openflow:1"]/i:node-connector[i:id="openflow:1:1
↪"]
            </it:inventory-node-connector-ref>
        </termination-point>
    </node>
</topology>
```

And the last input will be inventory part of topology. To the URL http://localhost:8181/restconf/config/opendaylight-inventory:nodes we PUT:

```
<nodes
    xmlns="urn:opendaylight:inventory">
    <node>
        <id>openflow:1</id>
        <node-connector>
            <id>openflow:1:1</id>
            <port-number
                xmlns="urn:opendaylight:flow:inventory">1
            </port-number>
            <current-speed
                xmlns="urn:opendaylight:flow:inventory">10000000
            </current-speed>
            <name
                xmlns="urn:opendaylight:flow:inventory">s1-eth1
            </name>
            <supported
                xmlns="urn:opendaylight:flow:inventory">
            </supported>
            <current-feature
                xmlns="urn:opendaylight:flow:inventory">copper ten-gb-fd
            </current-feature>
            <configuration
                xmlns="urn:opendaylight:flow:inventory">
            </configuration>
            <peer-features
                xmlns="urn:opendaylight:flow:inventory">
            </peer-features>
            <maximum-speed
                xmlns="urn:opendaylight:flow:inventory">0
            </maximum-speed>
            <advertised-features
                xmlns="urn:opendaylight:flow:inventory">
            </advertised-features>
            <hardware-address
                xmlns="urn:opendaylight:flow:inventory">0E:DC:8C:63:EC:D1
            </hardware-address>
            <state
                xmlns="urn:opendaylight:flow:inventory">
                <link-down>false</link-down>
                <blocked>false</blocked>
                <live>false</live>
            </state>
            <flow-capable-node-connector-statistics
                xmlns="urn:opendaylight:port:statistics">
                <receive-errors>0</receive-errors>
                <receive-frame-error>0</receive-frame-error>
```

```
                <receive-over-run-error>0</receive-over-run-error>
                <receive-crc-error>0</receive-crc-error>
                <bytes>
                    <transmitted>595</transmitted>
                    <received>378</received>
                </bytes>
                <receive-drops>0</receive-drops>
                <duration>
                    <second>28</second>
                    <nanosecond>410000000</nanosecond>
                </duration>
                <transmit-errors>0</transmit-errors>
                <collision-count>0</collision-count>
                <packets>
                    <transmitted>7</transmitted>
                    <received>5</received>
                </packets>
                <transmit-drops>0</transmit-drops>
            </flow-capable-node-connector-statistics>
        </node-connector>
        <node-connector>
            <id>openflow:1:LOCAL</id>
            <port-number
                xmlns="urn:opendaylight:flow:inventory">4294967294
            </port-number>
            <current-speed
                xmlns="urn:opendaylight:flow:inventory">0
            </current-speed>
            <name
                xmlns="urn:opendaylight:flow:inventory">s1
            </name>
            <supported
                xmlns="urn:opendaylight:flow:inventory">
            </supported>
            <current-feature
                xmlns="urn:opendaylight:flow:inventory">
            </current-feature>
            <configuration
                xmlns="urn:opendaylight:flow:inventory">
            </configuration>
            <peer-features
                xmlns="urn:opendaylight:flow:inventory">
            </peer-features>
            <maximum-speed
                xmlns="urn:opendaylight:flow:inventory">0
            </maximum-speed>
            <advertised-features
                xmlns="urn:opendaylight:flow:inventory">
            </advertised-features>
            <hardware-address
                xmlns="urn:opendaylight:flow:inventory">BA:63:87:0C:76:41
            </hardware-address>
            <state
                xmlns="urn:opendaylight:flow:inventory">
                <link-down>false</link-down>
                <blocked>false</blocked>
                <live>false</live>
            </state>
```

```
            <flow-capable-node-connector-statistics
                xmlns="urn:opendaylight:port:statistics">
                <receive-errors>0</receive-errors>
                <receive-frame-error>0</receive-frame-error>
                <receive-over-run-error>0</receive-over-run-error>
                <receive-crc-error>0</receive-crc-error>
                <bytes>
                    <transmitted>576</transmitted>
                    <received>468</received>
                </bytes>
                <receive-drops>0</receive-drops>
                <duration>
                    <second>28</second>
                    <nanosecond>426000000</nanosecond>
                </duration>
                <transmit-errors>0</transmit-errors>
                <collision-count>0</collision-count>
                <packets>
                    <transmitted>6</transmitted>
                    <received>6</received>
                </packets>
                <transmit-drops>0</transmit-drops>
            </flow-capable-node-connector-statistics>
        </node-connector>
        <serial-number
            xmlns="urn:opendaylight:flow:inventory">None
        </serial-number>
        <manufacturer
            xmlns="urn:opendaylight:flow:inventory">Nicira, Inc.
        </manufacturer>
        <hardware
            xmlns="urn:opendaylight:flow:inventory">Open vSwitch
        </hardware>
        <software
            xmlns="urn:opendaylight:flow:inventory">2.1.3
        </software>
        <description
            xmlns="urn:opendaylight:flow:inventory">None
        </description>
        <ip-address
            xmlns="urn:opendaylight:flow:inventory">10.20.30.40
    </ip-address>
        <meter-features
            xmlns="urn:opendaylight:meter:statistics">
            <max_bands>0</max_bands>
            <max_color>0</max_color>
            <max_meter>0</max_meter>
        </meter-features>
        <group-features
            xmlns="urn:opendaylight:group:statistics">
            <group-capabilities-supported
                xmlns:x="urn:opendaylight:group:types">x:chaining
            </group-capabilities-supported>
            <group-capabilities-supported
                xmlns:x="urn:opendaylight:group:types">x:select-weight
            </group-capabilities-supported>
            <group-capabilities-supported
                xmlns:x="urn:opendaylight:group:types">x:select-liveness
```

```
            </group-capabilities-supported>
            <max-groups>4294967040</max-groups>
            <actions>67082241</actions>
            <actions>0</actions>
        </group-features>
    </node>
</nodes>
```

After this, the expected result from a GET request to http://127.0.0.1:8181/restconf/operational/network-topology:network-topology is:

```
<network-topology
    xmlns="urn:TBD:params:xml:ns:yang:network-topology">
    <topology>
        <topology-id>render:1</topology-id>
        <node>
            <node-id>openflow:1</node-id>
            <node-augmentation
                xmlns="urn:opendaylight:topology:inventory:rendering">
                <ip-address>10.20.30.40</ip-address>
                <serial-number>None</serial-number>
                <manufacturer>Nicira, Inc.</manufacturer>
                <description>None</description>
                <hardware>Open vSwitch</hardware>
                <software>2.1.3</software>
            </node-augmentation>
            <termination-point>
                <tp-id>openflow:1:1</tp-id>
                <tp-augmentation
                    xmlns="urn:opendaylight:topology:inventory:rendering">
                    <hardware-address>0E:DC:8C:63:EC:D1</hardware-address>
                    <current-speed>10000000</current-speed>
                    <maximum-speed>0</maximum-speed>
                    <name>s1-eth1</name>
                </tp-augmentation>
            </termination-point>
            <termination-point>
                <tp-id>openflow:1:LOCAL</tp-id>
                <tp-augmentation
                    xmlns="urn:opendaylight:topology:inventory:rendering">
                    <hardware-address>BA:63:87:0C:76:41</hardware-address>
                    <current-speed>0</current-speed>
                    <maximum-speed>0</maximum-speed>
                    <name>s1</name>
                </tp-augmentation>
            </termination-point>
        </node>
    </topology>
</network-topology>
```

Use Cases

You can find use case examples on this wiki page.

Key APIs and Interfaces

The basic provider class is TopoProcessingProvider which provides startup and shutdown methods. Otherwise, the framework communicates via requests and outputs stored in the MD-SAL datastores.

API Reference Documentation

You can find API examples on this wiki page.

TTP Model Developer Guide

Overview

Table Type Patterns are a specification developed by the Open Networking Foundation to enable the description and negotiation of subsets of the OpenFlow protocol. This is particularly useful for hardware switches that support OpenFlow as it enables the to describe what features they do (and thus also what features they do not) support. More details can be found in the full specification listed on the OpenFlow specifications page.

TTP Model Architecture

The TTP Model provides a YANG-modeled type for a TTP and allows them to be associated with a master list of known TTPs, as well as active and supported TTPs with nodes in the MD-SAL inventory model.

Key APIs and Interfaces

The key API provided by the TTP Model feature is the ability to store a set of TTPs in the MD-SAL as well as associate zero or one active TTPs and zero or more supported TTPs along with a given node in the MD-SAL inventory model.

API Reference Documentation

RESTCONF

See the generated RESTCONF API documentation at: http://localhost:8181/apidoc/explorer/index.html

Look for the onf-ttp module to expand and see the various RESTCONF APIs.

Java Bindings

As stated above there are 3 locations where a Table Type Pattern can be placed into the MD-SAL Data Store. They correspond to 3 different REST API URIs:

1. `restconf/config/onf-ttp:opendaylight-ttps/onf-ttp:table-type-patterns/`
2. `restconf/config/opendaylight-inventory:nodes/node/{id}/ttp-inventory-node:active_ttp/`
3. `restconf/config/opendaylight-inventory:nodes/node/{id}/ttp-inventory-node:supported_ttps/`

Note: Typically, these URIs are running on the machine the controller is on at port 8181. If you are on the same machine they can thus be accessed at `http://localhost:8181/<uri>`

Using the TTP Model RESTCONF APIs

Setting REST HTTP Headers

Authentication

The REST API calls require authentication by default. The default method is to use basic auth with a user name and password of 'admin'.

Content-Type and Accept

RESTCONF supports both xml and json. This example focuses on JSON, but xml can be used just as easily. When doing a PUT or POST be sure to specify the appropriate `Conetnt-Type` header: either `application/json` or `application/xml`.

When doing a GET be sure to specify the appropriate `Accept` header: again, either `application/json` or `application/xml`.

Content

The contents of a PUT or POST should be a OpenDaylight Table Type Pattern. An example of one is provided below. The example can also be found at parser/sample-TTP-from-tests.ttp in the TTP git repository.

Sample Table Type Pattern (json).

```
{
    "table-type-patterns": {
        "table-type-pattern": [
            {
                "security": {
                    "doc": [
                        "This TTP is not published for use by ONF. It is an example␣
↪and for",
                        "illustrative purposes only.",
                        "If this TTP were published for use it would include",
                        "guidance as to any security considerations in this doc␣
↪member."
                    ]
                },
                "NDM_metadata": {
                    "authority": "org.opennetworking.fawg",
                    "OF_protocol_version": "1.3.3",
                    "version": "1.0.0",
                    "type": "TTPv1",
                    "doc": [
                        "Example of a TTP supporting L2 (unicast, multicast,␣
↪flooding), L3 (unicast only),",
                        "and an ACL table."
```

```
                    ],
                    "name": "L2-L3-ACLs"
                },
                "identifiers": [
                    {
                        "doc": [
                            "The VLAN ID of a locally attached L2 subnet on a Router."
                        ],
                        "var": "<subnet_VID>"
                    },
                    {
                        "doc": [
                            "An OpenFlow group identifier (integer) identifying a␣
↪group table entry",
                            "of the type indicated by the variable name"
                        ],
                        "var": "<<group_entry_types/name>>"
                    }
                ],
                "features": [
                    {
                        "doc": [
                            "Flow entry notification Extension - notification of␣
↪changes in flow entries"
                        ],
                        "feature": "ext187"
                    },
                    {
                        "doc": [
                            "Group notifications Extension - notification of changes␣
↪in group or meter entries"
                        ],
                        "feature": "ext235"
                    }
                ],
                "meter_table": {
                    "meter_types": [
                        {
                            "name": "ControllerMeterType",
                            "bands": [
                                {
                                    "type": "DROP",
                                    "rate": "1000..10000",
                                    "burst": "50..200"
                                }
                            ]
                        },
                        {
                            "name": "TrafficMeter",
                            "bands": [
                                {
                                    "type": "DSCP_REMARK",
                                    "rate": "10000..500000",
                                    "burst": "50..500"
                                },
                                {
                                    "type": "DROP",
                                    "rate": "10000..500000",
```

```
                                "burst": "50..500"
                            }
                        ]
                    }
                ],
                "built_in_meters": [
                    {
                        "name": "ControllerMeter",
                        "meter_id": 1,
                        "type": "ControllerMeterType",
                        "bands": [
                            {
                                "rate": 2000,
                                "burst": 75
                            }
                        ]
                    },
                    {
                        "name": "AllArpMeter",
                        "meter_id": 2,
                        "type": "ControllerMeterType",
                        "bands": [
                            {
                                "rate": 1000,
                                "burst": 50
                            }
                        ]
                    }
                ]
            },
            "table_map": [
                {
                    "name": "ControlFrame",
                    "number": 0
                },
                {
                    "name": "IngressVLAN",
                    "number": 10
                },
                {
                    "name": "MacLearning",
                    "number": 20
                },
                {
                    "name": "ACL",
                    "number": 30
                },
                {
                    "name": "L2",
                    "number": 40
                },
                {
                    "name": "ProtoFilter",
                    "number": 50
                },
                {
                    "name": "IPv4",
                    "number": 60
```

```
                    },
                    {
                        "name": "IPv6",
                        "number": 80
                    }
                ],
                "parameters": [
                    {
                        "doc": [
                            "documentation"
                        ],
                        "name": "Showing-curt-how-this-works",
                        "type": "type1"
                    }
                ],
                "flow_tables": [
                    {
                        "doc": [
                            "Filters L2 control reserved destination addresses and",
                            "may forward control packets to the controller.",
                            "Directs all other packets to the Ingress VLAN table."
                        ],
                        "name": "ControlFrame",
                        "flow_mod_types": [
                            {
                                "doc": [
                                    "This match/action pair allows for flow_mods that␣
↪match on either",
                                    "ETH_TYPE or ETH_DST (or both) and send the␣
↪packet to the",
                                    "controller, subject to metering."
                                ],
                                "name": "Frame-To-Controller",
                                "match_set": [
                                    {
                                        "field": "ETH_TYPE",
                                        "match_type": "all_or_exact"
                                    },
                                    {
                                        "field": "ETH_DST",
                                        "match_type": "exact"
                                    }
                                ],
                                "instruction_set": [
                                    {
                                        "doc": [
                                            "This meter may be used to limit the rate␣
↪of PACKET_IN frames",
                                            "sent to the controller"
                                        ],
                                        "instruction": "METER",
                                        "meter_name": "ControllerMeter"
                                    },
                                    {
                                        "instruction": "APPLY_ACTIONS",
                                        "actions": [
                                            {
                                                "action": "OUTPUT",
```

```
                                                "port": "CONTROLLER"
                                            }
                                        ]
                                    }
                                ]
                            }
                        ],
                        "built_in_flow_mods": [
                            {
                                "doc": [
                                    "Mandatory filtering of control frames with C-
↪VLAN Bridge reserved DA."
                                ],
                                "name": "Control-Frame-Filter",
                                "priority": "1",
                                "match_set": [
                                    {
                                        "field": "ETH_DST",
                                        "mask": "0xfffffffffff0",
                                        "value": "0x0180C2000000"
                                    }
                                ]
                            },
                            {
                                "doc": [
                                    "Mandatory miss flow_mod, sends packets to␣
↪IngressVLAN table."
                                ],
                                "name": "Non-Control-Frame",
                                "priority": "0",
                                "instruction_set": [
                                    {
                                        "instruction": "GOTO_TABLE",
                                        "table": "IngressVLAN"
                                    }
                                ]
                            }
                        ]
                    }
                ],
                "group_entry_types": [
                    {
                        "doc": [
                            "Output to a port, removing VLAN tag if needed.",
                            "Entry per port, plus entry per untagged VID per port."
                        ],
                        "name": "EgressPort",
                        "group_type": "INDIRECT",
                        "bucket_types": [
                            {
                                "name": "OutputTagged",
                                "action_set": [
                                    {
                                        "action": "OUTPUT",
                                        "port": "<port_no>"
                                    }
                                ]
                            },
```

```
                            {
                                "name": "OutputUntagged",
                                "action_set": [
                                    {
                                        "action": "POP_VLAN"
                                    },
                                    {
                                        "action": "OUTPUT",
                                        "port": "<port_no>"
                                    }
                                ]
                            },
                            {
                                "opt_tag": "VID-X",
                                "name": "OutputVIDTranslate",
                                "action_set": [
                                    {
                                        "action": "SET_FIELD",
                                        "field": "VLAN_VID",
                                        "value": "<local_vid>"
                                    },
                                    {
                                        "action": "OUTPUT",
                                        "port": "<port_no>"
                                    }
                                ]
                            }
                        ]
                    }
                ],
                "flow_paths": [
                    {
                        "doc": [
                            "This object contains just a few examples of flow paths,␣
↪it is not",
                            "a comprehensive list of the flow paths required for this␣
↪TTP.  It is",
                            "intended that the flow paths array could include either␣
↪a list of",
                            "required flow paths or a list of specific flow paths␣
↪that are not",
                            "required (whichever is more concise or more useful."
                        ],
                        "name": "L2-2",
                        "path": [
                            "Non-Control-Frame",
                            "IV-pass",
                            "Known-MAC",
                            "ACLskip",
                            "L2-Unicast",
                            "EgressPort"
                        ]
                    },
                    {
                        "name": "L2-3",
                        "path": [
                            "Non-Control-Frame",
                            "IV-pass",
```

```
                "Known-MAC",
                "ACLskip",
                "L2-Multicast",
                "L2Mcast",
                "[EgressPort]"
            ]
        },
        {
            "name": "L2-4",
            "path": [
                "Non-Control-Frame",
                "IV-pass",
                "Known-MAC",
                "ACL-skip",
                "VID-flood",
                "VIDflood",
                "[EgressPort]"
            ]
        },
        {
            "name": "L2-5",
            "path": [
                "Non-Control-Frame",
                "IV-pass",
                "Known-MAC",
                "ACLskip",
                "L2-Drop"
            ]
        },
        {
            "name": "v4-1",
            "path": [
                "Non-Control-Frame",
                "IV-pass",
                "Known-MAC",
                "ACLskip",
                "L2-Router-MAC",
                "IPv4",
                "v4-Unicast",
                "NextHop",
                "EgressPort"
            ]
        },
        {
            "name": "v4-2",
            "path": [
                "Non-Control-Frame",
                "IV-pass",
                "Known-MAC",
                "ACLskip",
                "L2-Router-MAC",
                "IPv4",
                "v4-Unicast-ECMP",
                "L3ECMP",
                "NextHop",
                "EgressPort"
            ]
        }
```

```
                    ]
                }
            ]
        }
}
```

Making a REST Call

In this example we'll do a PUT to install the sample TTP from above into OpenDaylight and then retrieve it both as json and as xml. We'll use the Postman - REST Client for Chrome in the examples, but any method of accessing REST should work.

First, we'll fill in the basic information:

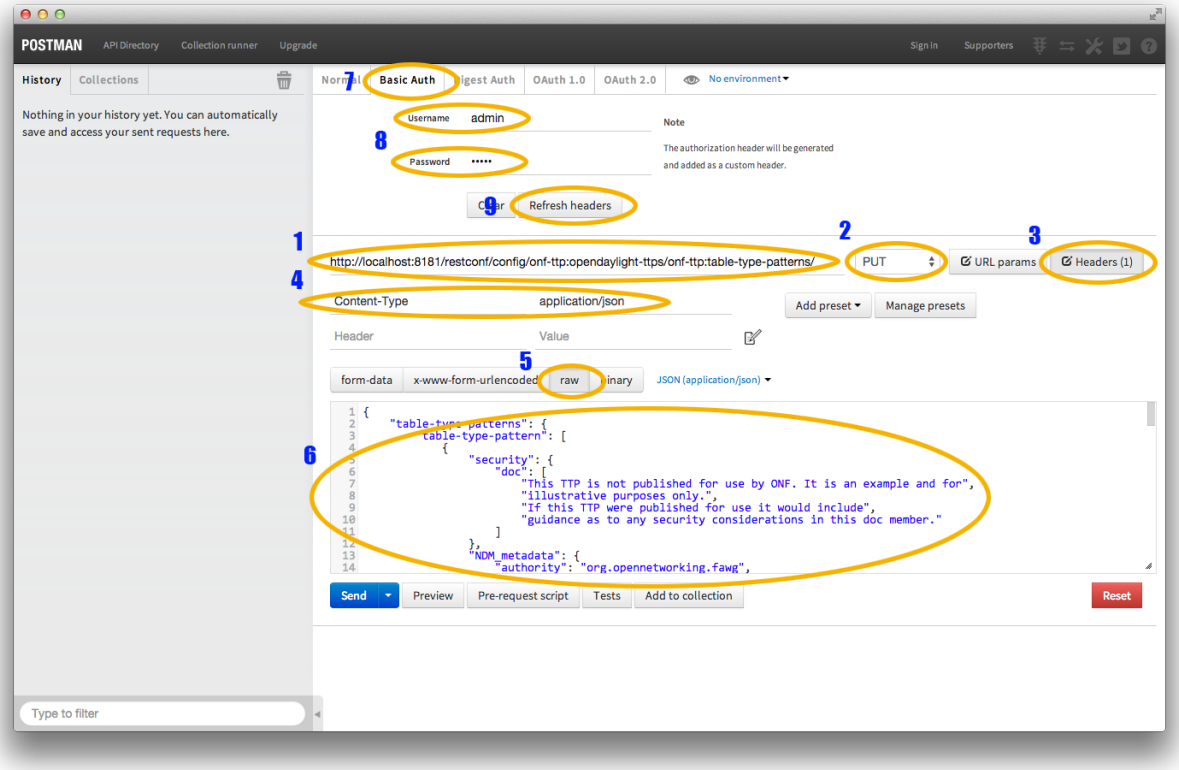


Fig. 2.73: Filling in URL, content, Content-Type and basic auth

1. Set the URL to `http://localhost:8181/restconf/config/onf-ttp:opendaylight-ttps/onf-ttp:table-type-patterns/`
2. Set the action to `PUT`
3. Click Headers and
4. Set a header for `Content-Type` to `application/json`
5. Make sure the content is set to raw and
6. Copy the sample TTP from above into the content

7. Click the Basic Auth tab and
8. Set the username and password to admin
9. Click Refresh headers

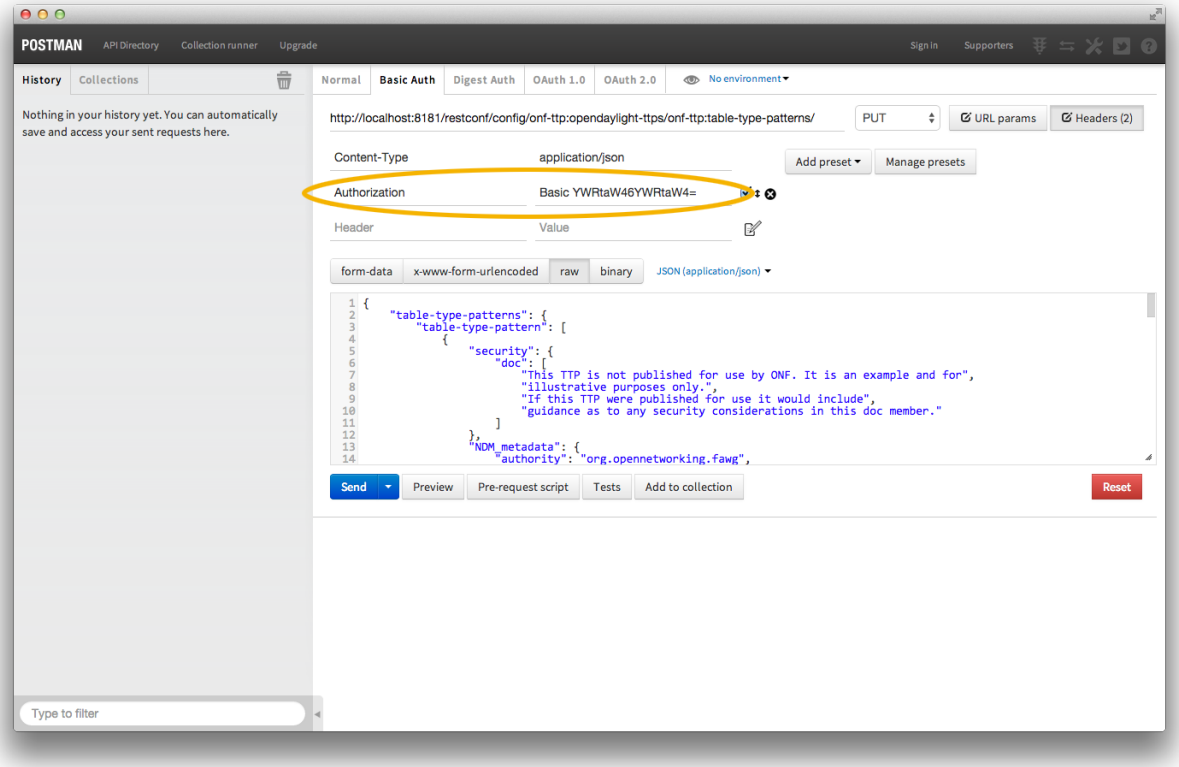


Fig. 2.74: Refreshing basic auth headers

After clicking Refresh headers, we can see that a new header (`Authorization`) has been created and this will allow us to authenticate to make the REST call.

At this point, clicking send should result in a Status response of `200 OK` indicating we've successfully PUT the TTP into OpenDaylight.

We can now retrieve the TTP by:

1. Changing the action to `GET`
2. Setting an `Accept` header to `application/json` and
3. Pressing send

The same process can retrieve the content as xml by setting the `Accept` header to `application/xml`.

TTP CLI Tools Developer Guide

Overview

Table Type Patterns are a specification developed by the Open Networking Foundation to enable the description and negotiation of subsets of the OpenFlow protocol. This is particularly useful for hardware switches that support Open-

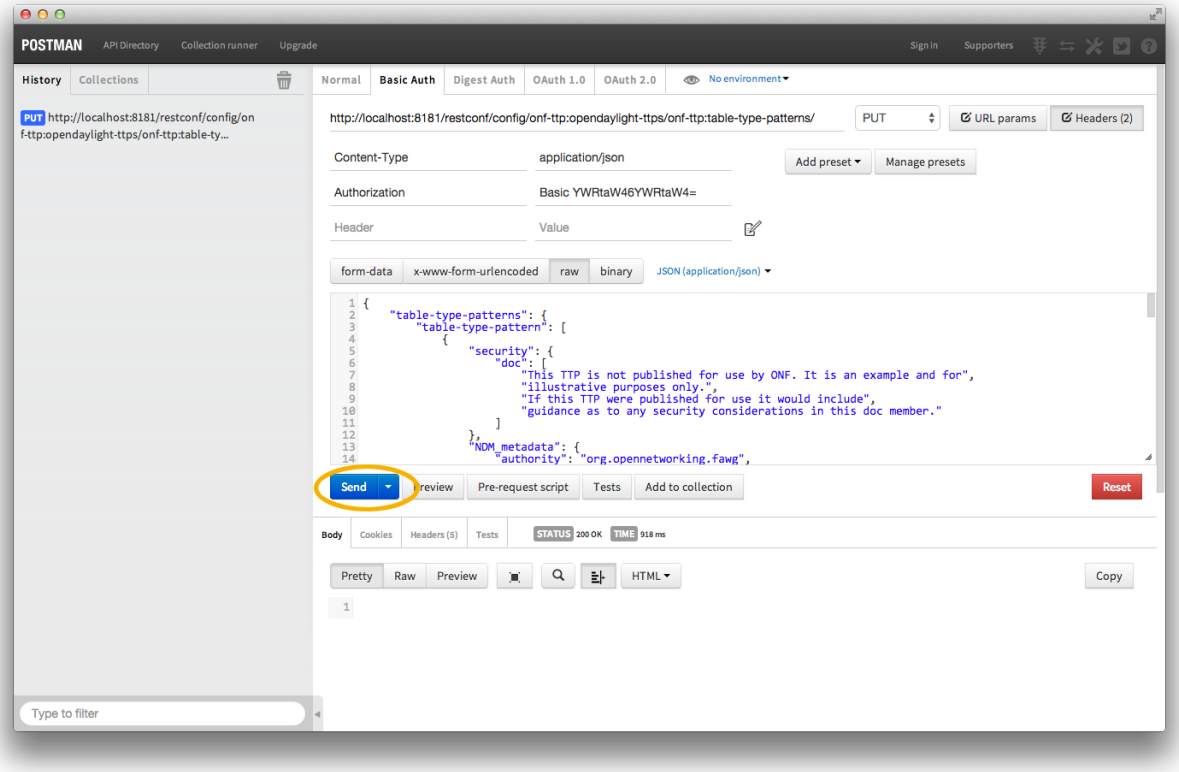


Fig. 2.75: PUTting a TTP

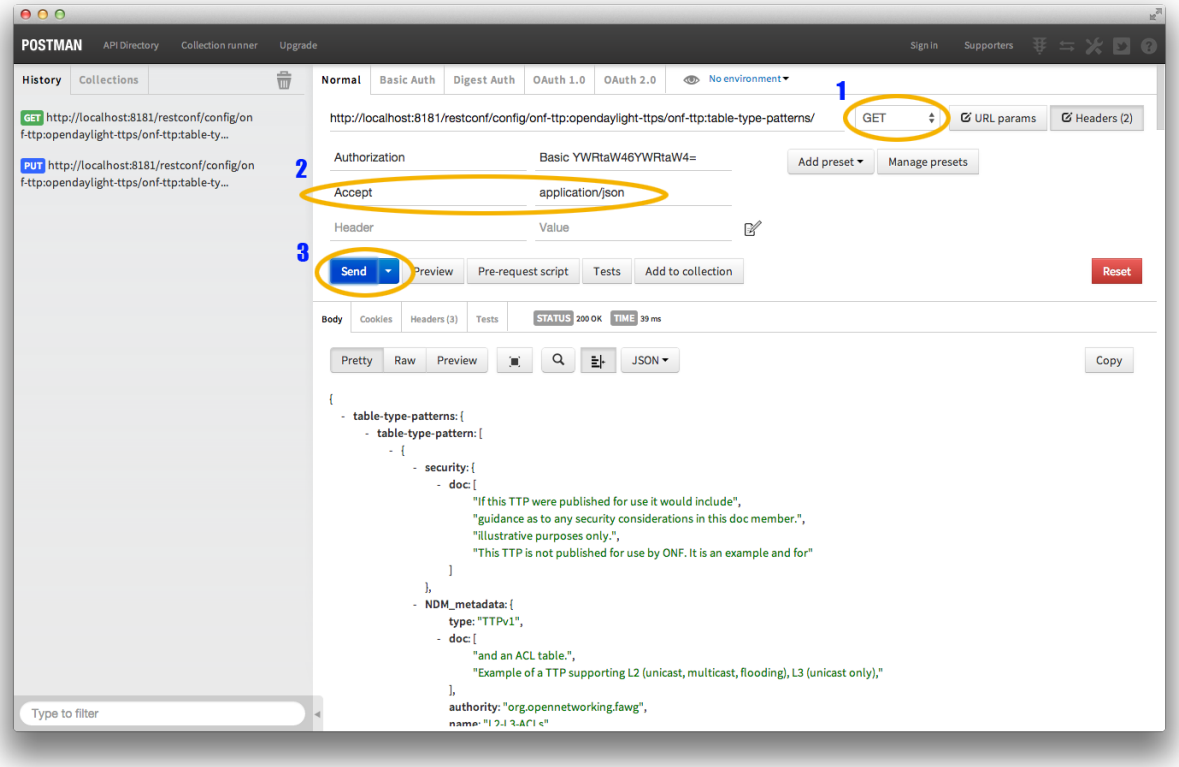


Fig. 2.76: Retrieving the TTP as json via a GET

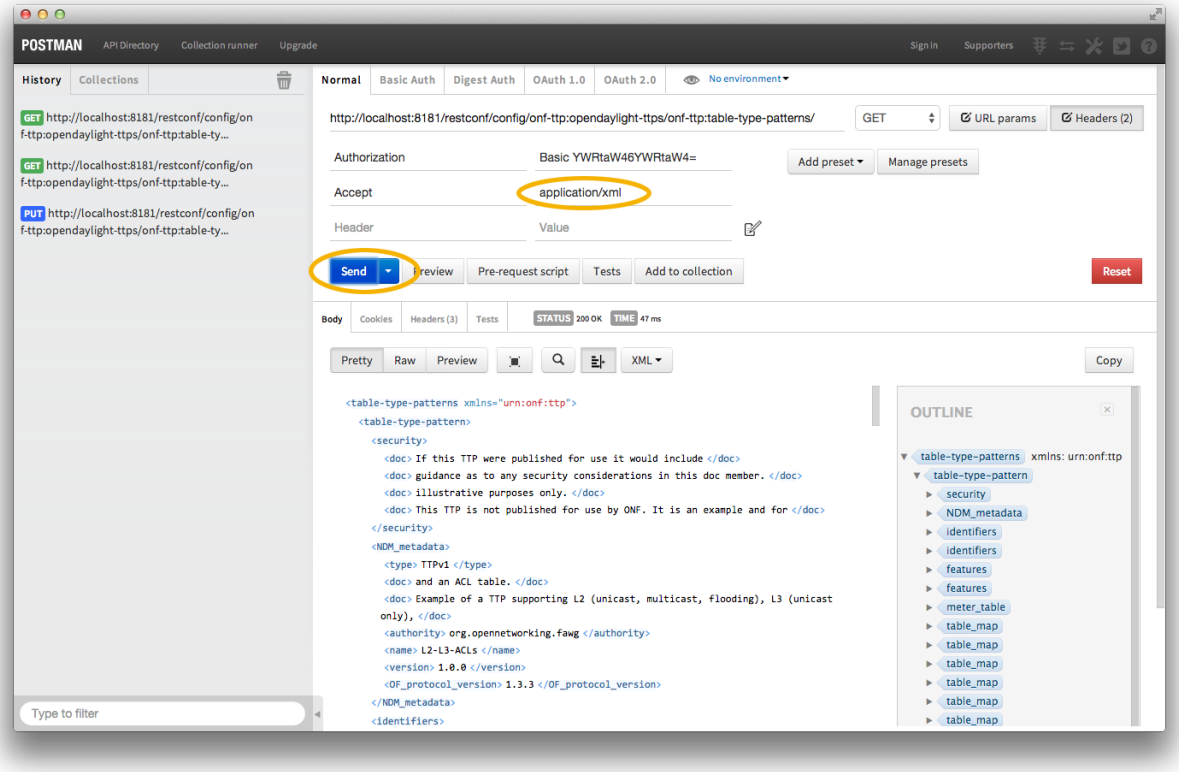


Fig. 2.77: Retrieving the TTP as xml via a GET

Flow as it enables the to describe what features they do (and thus also what features they do not) support. More details can be found in the full specification listed on the OpenFlow specifications page.

The TTP CLI Tools provide a way for people interested in TTPs to read in, validate, output, and manipulate TTPs as a self-contained, executable jar file.

TTP CLI Tools Architecture

The TTP CLI Tools use the TTP Model and the YANG Tools/RESTCONF codecs to translate between the Data Transfer Objects (DTOs) and JSON/XML.

Command Line Options

This will cover the various options for the CLI Tools. For now, there are no options and it merely outputs fixed data using the codecs.

User Network Interface Manager Plug-in (Unimgr) Developer Guide

Overview

The User Network Interface (UNI) Manager project within OpenDaylight provides data models and APIs that enable software applications and service orchestrators to configure and provision connectivity services; in particular, Carrier Ethernet services as defined by MEF Forum, in physical and virtual network elements.

Unimgr Architecture

Unimgr provides support for both service orchestration, via the Legato API, and network resource provisioning, via the Presto API. These APIs, and the interfaces they provide, are defined by YANG models developed within MEF in collaboration with ONF and IETF. An application/user can interact with Unimgr at either layer. For the Carbon release, the YANG models are as follows:

Key APIs and Interfaces

Legato YANG models: https://git.opendaylight.org/gerrit/gitweb?p=unimgr.git;a=tree;f=legato-api/src/main/yang;hb=refs/heads/stable/carbon

Presto YANG models: https://git.opendaylight.org/gerrit/gitweb?p=unimgr.git;a=tree;f=presto-api/src/main/yang;hb=refs/heads/stable/carbon

Legato API Tree

module: mef-services

```
+--rw mef-services
   +--rw mef-service* [svc-id]
      +--rw evc
      |  +--rw unis
      |  |  +--rw uni* [uni-id]
      |  |     +--rw evc-uni-ce-vlans
```

```
      |  |     |  +--rw evc-uni-ce-vlan* [vid]
      |  |     |     +--rw vid    -> /mef-interfaces:mef-interfaces/unis/uni[mef-
↪interfaces:uni-id = current()/../../../uni-id]/ce-vlans/ce-vlan/vid
      |  |     +--rw ingress-bwp-flows-per-cos!
      |  |     |  +--rw coupling-enabled?   boolean
      |  |     |  +--rw bwp-flow-per-cos* [cos-name]
      |  |     |     +--rw cos-name      -> /mef-global:mef-global/profiles/cos-names/
↪cos-name/name
      |  |     |     +--rw bw-profile    -> /mef-interfaces:mef-interfaces/unis/
↪uni[mef-interfaces:uni-id = current()/../../../uni-id]/ingress-envelopes/envelope/
↪env-id
      |  |     +--rw egress-bwp-flows-per-eec!
      |  |     |  +--rw coupling-enabled?   boolean
      |  |     |  +--rw bwp-flow-per-eec* [eec-name]
      |  |     |     +--rw eec-name      -> /mef-global:mef-global/profiles/eec-names/
↪eec-name/name
      |  |     |     +--rw bw-profile    -> /mef-interfaces:mef-interfaces/unis/
↪uni[mef-interfaces:uni-id = current()/../../../uni-id]/egress-envelopes/envelope/
↪env-id
      |  |     +--rw status
      |  |     |  +--ro oper-state-enabled?   boolean
      |  |     |  +--ro available-status?     mef-types:svc-endpoint-availability-type
      |  |     +--rw uni-id                        -> /mef-interfaces:mef-interfaces/
↪unis/uni/uni-id
      |  |     +--rw role                          mef-types:evc-uni-role-type
      |  |     +--rw admin-state-enabled?          boolean
      |  |     +--rw color-id?                     mef-types:cos-color-identifier-
↪type
      |  |     +--rw data-svc-frm-cos?             -> /mef-global:mef-global/
↪profiles/cos/cos-profile/id
      |  |     +--rw l2cp-svc-frm-cos?             -> /mef-global:mef-global/
↪profiles/l2cp-cos/l2cp-profile/id
      |  |     +--rw soam-svc-frm-cos?             -> /mef-global:mef-global/
↪profiles/cos/cos-profile/id
      |  |     +--rw data-svc-frm-eec?             -> /mef-global:mef-global/
↪profiles/eec/eec-profile/id
      |  |     +--rw l2cp-svc-frm-eec?             -> /mef-global:mef-global/
↪profiles/l2cp-eec/l2cp-profile/id
      |  |     +--rw soam-svc-frm-eec?             -> /mef-global:mef-global/
↪profiles/eec/eec-profile/id
      |  |     +--rw ingress-bw-profile-per-evc?   -> /mef-interfaces:mef-interfaces/
↪unis/uni[mef-interfaces:uni-id = current()/../uni-id]/ingress-envelopes/envelope/
↪env-id
      |  |     +--rw egress-bw-profile-per-evc?    -> /mef-interfaces:mef-interfaces/
↪unis/uni[mef-interfaces:uni-id = current()/../uni-id]/egress-envelopes/envelope/env-
↪id
      |  |     +--rw src-mac-addr-limit-enabled?   boolean
      |  |     +--rw src-mac-addr-limit?           uint32
      |  |     +--rw src-mac-addr-limit-interval?  yang:timeticks
      |  |     +--rw test-meg-enabled?             boolean
      |  |     +--rw test-meg?                     mef-types:identifier45
      |  |     +--rw subscriber-meg-mip-enabled?   boolean
      |  |     +--rw subscriber-meg-mip?           mef-types:identifier45
      |  +--rw status
      |  |  +--ro oper-state-enabled?   boolean
      |  |  +--ro available-status?     mef-types:virt-cx-availability-type
      |  +--rw sls-inclusions-by-cos
      |  |  +--rw sls-inclusion-by-cos* [cos-name]
```

```
      |  |     +--rw cos-name    -> /mef-global:mef-global/profiles/cos-names/cos-
↪name/name
      |  +--rw sls-uni-inclusions!
      |  |  +--rw sls-uni-inclusion-set* [pm-type pm-id uni-id1 uni-id2]
      |  |     +--rw pm-type     -> /mef-global:mef-global/slss/sls[mef-global:sls-id␣
↪= current()/../../../evc-performance-sls]/perf-objs/perf-obj/pm-type
      |  |     +--rw pm-id       -> /mef-global:mef-global/slss/sls[mef-global:sls-id␣
↪= current()/../../../evc-performance-sls]/perf-objs/perf-obj[mef-global:pm-type =␣
↪current()/../pm-type]/pm-id
      |  |     +--rw uni-id1     -> ../../../unis/uni/uni-id
      |  |     +--rw uni-id2     -> ../../../unis/uni/uni-id
      |  +--rw sls-uni-exclusions!
      |  |  +--rw sls-uni-exclusion-set* [pm-type pm-id uni-id1 uni-id2]
      |  |     +--rw pm-type     -> /mef-global:mef-global/slss/sls[mef-global:sls-id␣
↪= current()/../../../evc-performance-sls]/perf-objs/perf-obj/pm-type
      |  |     +--rw pm-id       -> /mef-global:mef-global/slss/sls[mef-global:sls-id␣
↪= current()/../../../evc-performance-sls]/perf-objs/perf-obj[mef-global:pm-type =␣
↪current()/../pm-type]/pm-id
      |  |     +--rw uni-id1     -> ../../../unis/uni/uni-id
      |  |     +--rw uni-id2     -> ../../../unis/uni/uni-id
      |  +--rw evc-id                          mef-types:evc-id-type
      |  +--ro evc-status?                     mef-types:evc-status-type
      |  +--rw evc-type                        mef-types:evc-type
      |  +--rw admin-state-enabled?            boolean
      |  +--rw elastic-enabled?                boolean
      |  +--rw elastic-service?                mef-types:identifier45
      |  +--rw max-uni-count?                  uint32
      |  +--rw preserve-ce-vlan-id?            boolean
      |  +--rw cos-preserve-ce-vlan-id?        boolean
      |  +--rw evc-performance-sls?            -> /mef-global:mef-global/slss/sls/sls-id
      |  +--rw unicast-svc-frm-delivery?       mef-types:data-svc-frame-delivery-type
      |  +--rw multicast-svc-frm-delivery?     mef-types:data-svc-frame-delivery-type
      |  +--rw broadcast-svc-frm-delivery?     mef-types:data-svc-frame-delivery-type
      |  +--rw evc-meg-id?                     mef-types:identifier45
      |  +--rw max-svc-frame-size?             mef-types:max-svc-frame-size-type
      +--rw svc-id         mef-types:retail-svc-id-type
      +--rw sp-id?         -> /mef-global:mef-global/svc-providers/svc-provider/sp-id
      +--rw svc-type?      mef-types:mef-service-type
      +--rw user-label?    mef-types:identifier45
      +--rw svc-entity?    mef-types:service-entity-type
```

module: mef-global

```
+--rw mef-global
   +--rw svc-providers!
   |  +--rw svc-provider* [sp-id]
   |     +--rw sp-id    mef-types:svc-provider-type
   +--rw cens!
   |  +--rw cen* [cen-id]
   |     +--rw cen-id    mef-types:cen-type
   |     +--rw sp-id?    -> /mef-global/svc-providers/svc-provider/sp-id
   +--rw slss!
   |  +--rw sls* [sls-id]
   |     +--rw perf-objs
   |     |  +--rw pm-time-interval                    uint64
   |     |  +--rw pm-time-interval-increment          uint64
   |     |  +--rw unavail-flr-threshold-pp            mef-types:simple-percent
   |     |  +--rw consecutive-small-time-intervals    uint64
```

```
   |      |  +--rw perf-obj* [pm-type pm-id]
   |      |     +--rw pm-type                                  mef-types:performance-
↪metric-type
   |      |     +--rw pm-id                                    mef-types:identifier45
   |      |     +--rw cos-name                                 -> /mef-global/profiles/
↪cos-names/cos-name/name
   |      |     +--rw fd-pp                                    mef-types:simple-percent
   |      |     +--rw fd-range-pp                              mef-types:simple-percent
   |      |     +--rw fd-perf-obj                              uint64
   |      |     +--rw fd-range-perf-obj                        uint64
   |      |     +--rw fd-mean-perf-obj                         uint64
   |      |     +--rw ifdv-pp                                  mef-types:simple-percent
   |      |     +--rw ifdv-pair-interval                       mef-types:simple-percent
   |      |     +--rw ifdv-perf-obj                            uint64
   |      |     +--rw flr-perf-obj                             uint64
   |      |     +--rw avail-pp                                 mef-types:simple-percent
   |      |     +--rw hli-perf-obj                             uint64
   |      |     +--rw chli-consecutive-small-time-intervals    uint64
   |      |     +--rw chli-perf-obj                            uint64
   |      |     +--rw min-uni-pairs-avail                      uint64
   |      |     +--rw gp-avail-pp                              mef-types:simple-percent
   |      +--rw sls-id       mef-types:cen-type
   |      +--rw sp-id?       -> /mef-global/svc-providers/svc-provider/sp-id
   +--rw subscribers!
   |  +--rw subscriber* [sub-id]
   |     +--rw sub-id    mef-types:subscriber-type
   |     +--rw sp-id?    -> /mef-global/svc-providers/svc-provider/sp-id
   |     +--rw cen-id?   -> /mef-global/cens/cen/cen-id
   +--rw profiles!
      +--rw cos-names
      |  +--rw cos-name* [name]
      |     +--rw name    mef-types:identifier45
      +--rw eec-names
      |  +--rw eec-name* [name]
      |     +--rw name    mef-types:identifier45
      +--rw ingress-bwp-flows
      |  +--rw bwp-flow* [bw-profile]
      |     +--rw bw-profile          mef-types:identifier45
      |     +--rw user-label?         mef-types:identifier45
      |     +--rw cir?                mef-types:bwp-cir-type
      |     +--rw cir-max?            mef-types:bwp-cir-type
      |     +--rw cbs?                mef-types:bwp-cbs-type
      |     +--rw eir?                mef-types:bwp-eir-type
      |     +--rw eir-max?            mef-types:bwp-eir-type
      |     +--rw ebs?                mef-types:bwp-ebs-type
      |     +--rw coupling-enabled?   boolean
      |     +--rw color-mode?         mef-types:bwp-color-mode-type
      |     +--rw coupling-flag?      mef-types:bwp-coupling-flag-type
      +--rw egress-bwp-flows
      |  +--rw bwp-flow* [bw-profile]
      |     +--rw bw-profile          mef-types:identifier45
      |     +--rw user-label?         mef-types:identifier45
      |     +--rw cir?                mef-types:bwp-cir-type
      |     +--rw cir-max?            mef-types:bwp-cir-type
      |     +--rw cbs?                mef-types:bwp-cbs-type
      |     +--rw eir?                mef-types:bwp-eir-type
      |     +--rw eir-max?            mef-types:bwp-eir-type
      |     +--rw ebs?                mef-types:bwp-ebs-type
```

```
      |     +--rw coupling-enabled?   boolean
      |     +--rw color-mode?         mef-types:bwp-color-mode-type
      |     +--rw coupling-flag?      mef-types:bwp-coupling-flag-type
      +--rw l2cp-cos
      |  +--rw l2cp-profile* [id]
      |     +--rw l2cps
      |     |  +--rw l2cp* [dest-mac-addr peering-proto-name]
      |     |     +--rw dest-mac-addr         yang:mac-address
      |     |     +--rw peering-proto-name    mef-types:identifier45
      |     |     +--rw protocol?             mef-types:l2cp-peering-protocol-type
      |     |     +--rw protocol-id?          yang:hex-string
      |     |     +--rw cos-name?             -> /mef-global/profiles/cos-names/cos-
↪name/name
      |     |     +--rw handling?             mef-types:l2cp-handling-type
      |     |     +--rw subtype*              yang:hex-string
      |     +--rw id           mef-types:identifier45
      |     +--rw user-label?   mef-types:identifier45
      +--rw l2cp-eec
      |  +--rw l2cp-profile* [id]
      |     +--rw l2cps
      |     |  +--rw l2cp* [dest-mac-addr peering-proto-name]
      |     |     +--rw dest-mac-addr         yang:mac-address
      |     |     +--rw peering-proto-name    mef-types:identifier45
      |     |     +--rw protocol?             mef-types:l2cp-peering-protocol-type
      |     |     +--rw protocol-id?          yang:hex-string
      |     |     +--rw eec-name?             -> /mef-global/profiles/eec-names/eec-
↪name/name
      |     |     +--rw handling?             mef-types:l2cp-handling-type
      |     |     +--rw subtype*              yang:hex-string
      |     +--rw id           mef-types:identifier45
      |     +--rw user-label?   mef-types:identifier45
      +--rw l2cp-peering
      |  +--rw l2cp-profile* [id]
      |     +--rw l2cps
      |     |  +--rw l2cp* [dest-mac-addr peering-proto-name]
      |     |     +--rw dest-mac-addr         yang:mac-address
      |     |     +--rw peering-proto-name    mef-types:identifier45
      |     |     +--rw protocol?             mef-types:l2cp-peering-protocol-type
      |     |     +--rw protocol-id?          yang:hex-string
      |     |     +--rw subtype*              yang:hex-string
      |     +--rw id           mef-types:identifier45
      |     +--rw user-label?   mef-types:identifier45
      +--rw elmi
      |  +--rw elmi-profile* [id]
      |     +--rw id                            mef-types:identifier45
      |     +--rw user-label?                   mef-types:identifier45
      |     +--rw polling-counter?              mef-types:elmi-polling-counter-type
      |     +--rw status-error-threshold?       mef-types:elmi-status-error-threshold-
↪type
      |     +--rw polling-timer?                mef-types:elmi-polling-timer-type
      |     +--rw polling-verification-timer?   mef-types:elmi-polling-verification-
↪timer-type
      +--rw eec
      |  +--rw eec-profile* [id]
      |     +--rw id          mef-types:identifier45
      |     +--rw (eec-id)?
      |        +--:(pcp)
      |        |  +--rw eec-pcp!
```

```
      |         |     +--rw default-pcp-eec-name?   -> /mef-global/profiles/eec-names/
↪eec-name/name
      |         |     +--rw default-pcp-color?      mef-types:cos-color-type
      |         |     +--rw pcp* [pcp-value]
      |         |        +--rw pcp-value        mef-types:ieee8021p-priority-type
      |         |        +--rw discard-value?   boolean
      |         |        +--rw eec-name?        -> /mef-global/profiles/eec-names/eec-
↪name/name
      |         |        +--rw color?           mef-types:cos-color-type
      |         +--:(dscp)
      |            +--rw eec-dscp!
      |               +--rw default-ipv4-eec-name?   -> /mef-global/profiles/eec-names/
↪eec-name/name
      |               +--rw default-ipv4-color?      mef-types:cos-color-type
      |               +--rw default-ipv6-eec-name?   -> /mef-global/profiles/eec-names/
↪eec-name/name
      |               +--rw default-ipv6-color?      mef-types:cos-color-type
      |               +--rw ipv4-dscp* [dscp-value]
      |               |  +--rw dscp-value       inet:dscp
      |               |  +--rw discard-value?   boolean
      |               |  +--rw eec-name?        -> /mef-global/profiles/eec-names/eec-
↪name/name
      |               |  +--rw color?           mef-types:cos-color-type
      |               +--rw ipv6-dscp* [dscp-value]
      |                  +--rw dscp-value       inet:dscp
      |                  +--rw discard-value?   boolean
      |                  +--rw eec-name?        -> /mef-global/profiles/eec-names/eec-
↪name/name
      |                  +--rw color?           mef-types:cos-color-type
      +--rw cos
         +--rw cos-profile* [id]
            +--rw id          mef-types:identifier45
            +--rw (cos-id)?
               +--:(evc)
               |  +--rw cos-evc!
               |     +--rw default-evc-cos-name?   -> /mef-global/profiles/cos-names/
↪cos-name/name
               |     +--rw default-evc-color?      mef-types:cos-color-type
               +--:(pcp)
               |  +--rw cos-pcp!
               |     +--rw default-pcp-cos-name?   -> /mef-global/profiles/cos-names/
↪cos-name/name
               |     +--rw default-pcp-color?      mef-types:cos-color-type
               |     +--rw pcp* [pcp-value]
               |        +--rw pcp-value        mef-types:ieee8021p-priority-type
               |        +--rw discard-value?   boolean
               |        +--rw cos-name?        -> /mef-global/profiles/cos-names/cos-
↪name/name
               |        +--rw color?           mef-types:cos-color-type
               +--:(dscp)
                  +--rw cos-dscp!
                     +--rw default-ipv4-cos-name?   -> /mef-global/profiles/cos-names/
↪cos-name/name
                     +--rw default-ipv4-color?      mef-types:cos-color-type
                     +--rw default-ipv6-cos-name?   -> /mef-global/profiles/cos-names/
↪cos-name/name
                     +--rw default-ipv6-color?      mef-types:cos-color-type
                     +--rw ipv4-dscp* [dscp-value]
```

```
                     |  +--rw dscp-value       inet:dscp
                     |  +--rw discard-value?   boolean
                     |  +--rw cos-name?        -> /mef-global/profiles/cos-names/cos-
↪name/name
                     |  +--rw color?           mef-types:cos-color-type
                     +--rw ipv6-dscp* [dscp-value]
                        +--rw dscp-value       inet:dscp
                        +--rw discard-value?   boolean
                        +--rw cos-name?        -> /mef-global/profiles/cos-names/cos-
↪name/name
                        +--rw color?           mef-types:cos-color-type
```

Presto API Tree

module: onf-core-network-module

```
+--rw forwarding-constructs
   +--rw forwarding-construct* [uuid]
      +--rw uuid                    string
      +--rw layerProtocolName?      onf-cnt:LayerProtocolName
      +--rw lowerLevelFc*           -> /forwarding-constructs/forwarding-construct/uuid
      +--rw fcRoute* [uuid]
      |  +--rw uuid    string
      |  +--rw fc*     -> /forwarding-constructs/forwarding-construct/uuid
      +--rw fcPort* [topology node tp]
      |  +--rw topology           nt:topology-ref
      |  +--rw node               nt:node-ref
      |  +--rw tp                 nt:tp-ref
      |  +--rw role?              onf-cnt:PortRole
      |  +--rw fcPortDirection?   onf-cnt:PortDirection
      +--rw fcSpec
      |  +--rw uuid?                      string
      |  +--rw fcPortSpec* [uuid]
      |  |  +--rw uuid                 string
      |  |  +--rw ingressFcPortSet* [topology node tp]
      |  |  |  +--rw topology    nt:topology-ref
      |  |  |  +--rw node        nt:node-ref
      |  |  |  +--rw tp          nt:tp-ref
      |  |  +--rw egressFcPortSet* [topology node tp]
      |  |  |  +--rw topology    nt:topology-ref
      |  |  |  +--rw node        nt:node-ref
      |  |  |  +--rw tp          nt:tp-ref
      |  |  +--rw role?               string
      |  +--rw nrp:nrp-ce-fcspec-attrs
      |     +--rw nrp:connectionType?           nrp-types:NRP_ConnectionType
      |     +--rw nrp:unicastFrameDelivery?     nrp-types:NRP_ServiceFrameDelivery
      |     +--rw nrp:multicastFrameDelivery?   nrp-types:NRP_ServiceFrameDelivery
      |     +--rw nrp:broadcastFrameDelivery?   nrp-types:NRP_ServiceFrameDelivery
      |     +--rw nrp:vcMaxServiceFrame?        nrp-types:NRP_PositiveInteger
      |     +--rw nrp:vcId?                     nrp-types:NRP_PositiveInteger
      +--rw forwardingDirection?   onf-cnt:ForwardingDirection
```

augment /nt:network-topology/nt:topology/nt:node/nt:termination-point:

```
+--rw ltp-attrs
   +--rw lpList* [uuid]
```

```
   |  +--rw uuid                          string
   |  +--rw layerProtocolName?            onf-cnt:LayerProtocolName
   |  +--rw lpSpec
   |  |  +--rw adapterSpec
   |  |  |  +--rw nrp:nrp-conn-adapt-spec-attrs
   |  |  |  |  +--rw nrp:sourceMacAddressLimit
   |  |  |  |  |  +--rw nrp:enabled?        boolean
   |  |  |  |  |  +--rw nrp:limit?          NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:timeInterval?   NRP_NaturalNumber
   |  |  |  |  +--rw nrp:CeExternalInterface
   |  |  |  |  |  +--rw nrp:physicalLayer?             nrp-types:NRP_PhysicalLayer
   |  |  |  |  |  +--rw nrp:syncMode* [linkId]
   |  |  |  |  |  |  +--rw nrp:linkId             string
   |  |  |  |  |  |  +--rw nrp:syncModeEnabled?   boolean
   |  |  |  |  |  +--rw nrp:numberOfLinks?             nrp-types:NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:resiliency?                nrp-types:NRP_
↪InterfaceResiliency
   |  |  |  |  |  +--rw nrp:portConvsIdToAggLinkMap
   |  |  |  |  |  |  +--rw nrp:conversationId?   NRP_NaturalNumber
   |  |  |  |  |  |  +--rw nrp:linkId?           NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:maxFrameSize?              nrp-types:NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:linkOamEnabled?            boolean
   |  |  |  |  |  +--rw nrp:tokenShareEnabled?         boolean
   |  |  |  |  |  +--rw nrp:serviceProviderUniId?      string
   |  |  |  |  +--rw nrp:coloridentifier
   |  |  |  |  |  +--rw (identifier)?
   |  |  |  |  |     +--:(sap-color-id)
   |  |  |  |  |     |  +--rw nrp:serviceAccessPointColorId
   |  |  |  |  |     |     +--rw nrp:color?   nrp-types:NRP_FrameColor
   |  |  |  |  |     +--:(pcp-color-id)
   |  |  |  |  |     |  +--rw nrp:pcpColorId
   |  |  |  |  |     |     +--rw nrp:vlanTag?    nrp-types:NRP_VlanTag
   |  |  |  |  |     |     +--rw nrp:pcpValue*   nrp-types:NRP_NaturalNumber
   |  |  |  |  |     |     +--rw nrp:color?      nrp-types:NRP_FrameColor
   |  |  |  |  |     +--:(dei-color-id)
   |  |  |  |  |     |  +--rw nrp:deiColorId
   |  |  |  |  |     |     +--rw nrp:vlanTag?    nrp-types:NRP_VlanTag
   |  |  |  |  |     |     +--rw nrp:deiValue*   nrp-types:NRP_NaturalNumber
   |  |  |  |  |     |     +--rw nrp:color?      nrp-types:NRP_FrameColor
   |  |  |  |  |     +--:(desp-color-id)
   |  |  |  |  |        +--rw nrp:despColorId
   |  |  |  |  |           +--rw nrp:ipVersion?   nrp-types:NRP_IpVersion
   |  |  |  |  |           +--rw nrp:dscpValue*   nrp-types:NRP_NaturalNumber
   |  |  |  |  |           +--rw nrp:color?       nrp-types:NRP_FrameColor
   |  |  |  |  +--rw nrp:ingressBwpFlow
   |  |  |  |  |  +--rw nrp:bwpFlowIndex?         nrp-types:NRP_PositiveInteger
   |  |  |  |  |  +--rw nrp:cir?                  nrp-types:NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:cirMax?               nrp-types:NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:cbs?                  nrp-types:NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:eir?                  nrp-types:NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:eirMax?               nrp-types:NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:ebs?                  nrp-types:NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:couplingFlag?         nrp-types:NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:colorMode?            nrp-types:NRP_ColorMode
   |  |  |  |  |  +--rw nrp:rank?                 nrp-types:NRP_PositiveInteger
   |  |  |  |  |  +--rw nrp:tokenRequestOffset?   nrp-types:NRP_NaturalNumber
   |  |  |  |  +--rw nrp:egressBwpFlow
   |  |  |  |  |  +--rw nrp:bwpFlowIndex?         nrp-types:NRP_PositiveInteger
```

```
   |  |  |  |  |  +--rw nrp:cir?                  nrp-types:NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:cirMax?               nrp-types:NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:cbs?                  nrp-types:NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:eir?                  nrp-types:NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:eirMax?               nrp-types:NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:ebs?                  nrp-types:NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:couplingFlag?         nrp-types:NRP_NaturalNumber
   |  |  |  |  |  +--rw nrp:colorMode?            nrp-types:NRP_ColorMode
   |  |  |  |  |  +--rw nrp:rank?                 nrp-types:NRP_PositiveInteger
   |  |  |  |  |  +--rw nrp:tokenRequestOffset?   nrp-types:NRP_NaturalNumber
   |  |  |  |  +--rw nrp:l2cpAddressSet?          nrp-types:NRP_L2cpAddressSet
   |  |  |  |  +--rw nrp:l2cpPeering* [linkId]
   |  |  |  |     +--rw nrp:destinationMacAddress?   string
   |  |  |  |     +--rw nrp:protocolType?            NRP_ProtocolFrameType
   |  |  |  |     +--rw nrp:linkId                   string
   |  |  |  |     +--rw nrp:protocolId?              string
   |  |  |  +--rw nrp:nrp-ivc-endpoint-conn-adapt-spec-attrs
   |  |  |  |  +--rw nrp:ivcEndPointId?             string
   |  |  |  |  +--rw nrp:testMegEnabled?            boolean
   |  |  |  |  +--rw nrp:ivcEndPointRole?           nrp-types:NRP_EndPointRole
   |  |  |  |  +--rw nrp:ivcEndPointMap* [vlanId]
   |  |  |  |  |  +--rw nrp:vlanId        nrp-types:NRP_PositiveInteger
   |  |  |  |  |  +--rw (endpoint-map-form)?
   |  |  |  |  |     +--:(map-form-e)
   |  |  |  |  |     |  +--rw nrp:enni-svid* [vid]
   |  |  |  |  |     |     +--rw nrp:vid    nrp-types:NRP_PositiveInteger
   |  |  |  |  |     +--:(map-form-t)
   |  |  |  |  |     |  +--rw nrp:root-svid?    nrp-types:NRP_PositiveInteger
   |  |  |  |  |     |  +--rw nrp:leaf-svid?    nrp-types:NRP_PositiveInteger
   |  |  |  |  |     +--:(map-form-v)
   |  |  |  |  |     |  +--rw nrp:vuni-vid?     nrp-types:NRP_PositiveInteger
   |  |  |  |  |     |  +--rw nrp:enni-cevid* [vid]
   |  |  |  |  |     |     +--rw nrp:vid    nrp-types:NRP_PositiveInteger
   |  |  |  |  |     +--:(map-form-u)
   |  |  |  |  |        +--rw nrp:cvid* [vid]
   |  |  |  |  |           +--rw nrp:vid    nrp-types:NRP_PositiveInteger
   |  |  |  |  +--rw nrp:subscriberMegMipEnabled?   boolean
   |  |  |  +--rw nrp:nrp-evc-endpoint-conn-adapt-spec-attrs
   |  |  |     +--rw nrp:sourceMacAddressLimit
   |  |  |     |  +--rw nrp:enabled?        boolean
   |  |  |     |  +--rw nrp:limit?          NRP_NaturalNumber
   |  |  |     |  +--rw nrp:timeInterval?   NRP_NaturalNumber
   |  |  |     +--rw nrp:CeExternalInterface
   |  |  |     |  +--rw nrp:physicalLayer?             nrp-types:NRP_PhysicalLayer
   |  |  |     |  +--rw nrp:syncMode* [linkId]
   |  |  |     |  |  +--rw nrp:linkId             string
   |  |  |     |  |  +--rw nrp:syncModeEnabled?   boolean
   |  |  |     |  +--rw nrp:numberOfLinks?             nrp-types:NRP_NaturalNumber
   |  |  |     |  +--rw nrp:resiliency?                nrp-types:NRP_
↪InterfaceResiliency
   |  |  |     |  +--rw nrp:portConvsIdToAggLinkMap
   |  |  |     |  |  +--rw nrp:conversationId?   NRP_NaturalNumber
   |  |  |     |  |  +--rw nrp:linkId?           NRP_NaturalNumber
   |  |  |     |  +--rw nrp:maxFrameSize?              nrp-types:NRP_NaturalNumber
   |  |  |     |  +--rw nrp:linkOamEnabled?            boolean
   |  |  |     |  +--rw nrp:tokenShareEnabled?         boolean
   |  |  |     |  +--rw nrp:serviceProviderUniId?      string
   |  |  |     +--rw nrp:coloridentifier
```

```
|  |  |     |  +--rw (identifier)?
|  |  |     |     +--:(sap-color-id)
|  |  |     |     |  +--rw nrp:serviceAccessPointColorId
|  |  |     |     |     +--rw nrp:color?   nrp-types:NRP_FrameColor
|  |  |     |     +--:(pcp-color-id)
|  |  |     |     |  +--rw nrp:pcpColorId
|  |  |     |     |     +--rw nrp:vlanTag?    nrp-types:NRP_VlanTag
|  |  |     |     |     +--rw nrp:pcpValue*   nrp-types:NRP_NaturalNumber
|  |  |     |     |     +--rw nrp:color?      nrp-types:NRP_FrameColor
|  |  |     |     +--:(dei-color-id)
|  |  |     |     |  +--rw nrp:deiColorId
|  |  |     |     |     +--rw nrp:vlanTag?    nrp-types:NRP_VlanTag
|  |  |     |     |     +--rw nrp:deiValue*   nrp-types:NRP_NaturalNumber
|  |  |     |     |     +--rw nrp:color?      nrp-types:NRP_FrameColor
|  |  |     |     +--:(desp-color-id)
|  |  |     |        +--rw nrp:despColorId
|  |  |     |           +--rw nrp:ipVersion?   nrp-types:NRP_IpVersion
|  |  |     |           +--rw nrp:dscpValue*   nrp-types:NRP_NaturalNumber
|  |  |     |           +--rw nrp:color?       nrp-types:NRP_FrameColor
|  |  |     +--rw nrp:ingressBwpFlow
|  |  |     |  +--rw nrp:bwpFlowIndex?         nrp-types:NRP_PositiveInteger
|  |  |     |  +--rw nrp:cir?                  nrp-types:NRP_NaturalNumber
|  |  |     |  +--rw nrp:cirMax?               nrp-types:NRP_NaturalNumber
|  |  |     |  +--rw nrp:cbs?                  nrp-types:NRP_NaturalNumber
|  |  |     |  +--rw nrp:eir?                  nrp-types:NRP_NaturalNumber
|  |  |     |  +--rw nrp:eirMax?               nrp-types:NRP_NaturalNumber
|  |  |     |  +--rw nrp:ebs?                  nrp-types:NRP_NaturalNumber
|  |  |     |  +--rw nrp:couplingFlag?         nrp-types:NRP_NaturalNumber
|  |  |     |  +--rw nrp:colorMode?            nrp-types:NRP_ColorMode
|  |  |     |  +--rw nrp:rank?                 nrp-types:NRP_PositiveInteger
|  |  |     |  +--rw nrp:tokenRequestOffset?   nrp-types:NRP_NaturalNumber
|  |  |     +--rw nrp:egressBwpFlow
|  |  |     |  +--rw nrp:bwpFlowIndex?         nrp-types:NRP_PositiveInteger
|  |  |     |  +--rw nrp:cir?                  nrp-types:NRP_NaturalNumber
|  |  |     |  +--rw nrp:cirMax?               nrp-types:NRP_NaturalNumber
|  |  |     |  +--rw nrp:cbs?                  nrp-types:NRP_NaturalNumber
|  |  |     |  +--rw nrp:eir?                  nrp-types:NRP_NaturalNumber
|  |  |     |  +--rw nrp:eirMax?               nrp-types:NRP_NaturalNumber
|  |  |     |  +--rw nrp:ebs?                  nrp-types:NRP_NaturalNumber
|  |  |     |  +--rw nrp:couplingFlag?         nrp-types:NRP_NaturalNumber
|  |  |     |  +--rw nrp:colorMode?            nrp-types:NRP_ColorMode
|  |  |     |  +--rw nrp:rank?                 nrp-types:NRP_PositiveInteger
|  |  |     |  +--rw nrp:tokenRequestOffset?   nrp-types:NRP_NaturalNumber
|  |  |     +--rw nrp:l2cpAddressSet?            nrp-types:NRP_L2cpAddressSet
|  |  |     +--rw nrp:l2cpPeering* [linkId]
|  |  |     |  +--rw nrp:destinationMacAddress?   string
|  |  |     |  +--rw nrp:protocolType?            NRP_ProtocolFrameType
|  |  |     |  +--rw nrp:linkId                   string
|  |  |     |  +--rw nrp:protocolId?              string
|  |  |     +--rw nrp:evcEndPointId?             nrp-types:NRP_PositiveInteger
|  |  |     +--rw nrp:testMegEnabled?            boolean
|  |  |     +--rw nrp:evcEndPointRole?           nrp-types:NRP_EvcEndPointRole
|  |  |     +--rw nrp:evcEndPointMap* [vid]
|  |  |     |  +--rw nrp:vid    nrp-types:NRP_PositiveInteger
|  |  |     +--rw nrp:subscriberMegMipEbabled?   boolean
|  |  +--rw terminationSpec
|  |  |  +--rw nrp:nrp-termination-spec-attrs
|  |  |  |  +--rw nrp:physicalLayer?             nrp-types:NRP_PhysicalLayer
```

```
|  |  |  |  +--rw nrp:syncMode* [linkId]
|  |  |  |  |  +--rw nrp:linkId              string
|  |  |  |  |  +--rw nrp:syncModeEnabled?   boolean
|  |  |  |  +--rw nrp:numberOfLinks?             nrp-types:NRP_NaturalNumber
|  |  |  |  +--rw nrp:resiliency?                nrp-types:NRP_InterfaceResiliency
|  |  |  |  +--rw nrp:portConvsIdToAggLinkMap
|  |  |  |  |  +--rw nrp:conversationId?   NRP_NaturalNumber
|  |  |  |  |  +--rw nrp:linkId?           NRP_NaturalNumber
|  |  |  |  +--rw nrp:maxFrameSize?              nrp-types:NRP_NaturalNumber
|  |  |  |  +--rw nrp:linkOamEnabled?            boolean
|  |  |  |  +--rw nrp:tokenShareEnabled?         boolean
|  |  |  |  +--rw nrp:serviceProviderUniId?      string
|  |  |  |  +--rw nrp:nrp-uni-termination-attrs
|  |  |  |     +--rw nrp:defaultCeVlanId?             nrp-types:NRP_PositiveInteger
|  |  |  |     +--rw nrp:uniMegEnabled?               boolean
|  |  |  |     +--rw nrp:elmiEnabled?                 boolean
|  |  |  |     +--rw nrp:serviceprovideruniprofile?   string
|  |  |  |     +--rw nrp:operatoruniprofile?          string
|  |  |  |     +--rw nrp:ingressBwpUni
|  |  |  |     |  +--rw nrp:bwpFlowIndex?         nrp-types:NRP_PositiveInteger
|  |  |  |     |  +--rw nrp:cir?                  nrp-types:NRP_NaturalNumber
|  |  |  |     |  +--rw nrp:cirMax?               nrp-types:NRP_NaturalNumber
|  |  |  |     |  +--rw nrp:cbs?                  nrp-types:NRP_NaturalNumber
|  |  |  |     |  +--rw nrp:eir?                  nrp-types:NRP_NaturalNumber
|  |  |  |     |  +--rw nrp:eirMax?               nrp-types:NRP_NaturalNumber
|  |  |  |     |  +--rw nrp:ebs?                  nrp-types:NRP_NaturalNumber
|  |  |  |     |  +--rw nrp:couplingFlag?         nrp-types:NRP_NaturalNumber
|  |  |  |     |  +--rw nrp:colorMode?            nrp-types:NRP_ColorMode
|  |  |  |     |  +--rw nrp:rank?                 nrp-types:NRP_PositiveInteger
|  |  |  |     |  +--rw nrp:tokenRequestOffset?   nrp-types:NRP_NaturalNumber
|  |  |  |     +--rw nrp:egressBwpUni
|  |  |  |        +--rw nrp:bwpFlowIndex?         nrp-types:NRP_PositiveInteger
|  |  |  |        +--rw nrp:cir?                  nrp-types:NRP_NaturalNumber
|  |  |  |        +--rw nrp:cirMax?               nrp-types:NRP_NaturalNumber
|  |  |  |        +--rw nrp:cbs?                  nrp-types:NRP_NaturalNumber
|  |  |  |        +--rw nrp:eir?                  nrp-types:NRP_NaturalNumber
|  |  |  |        +--rw nrp:eirMax?               nrp-types:NRP_NaturalNumber
|  |  |  |        +--rw nrp:ebs?                  nrp-types:NRP_NaturalNumber
|  |  |  |        +--rw nrp:couplingFlag?         nrp-types:NRP_NaturalNumber
|  |  |  |        +--rw nrp:colorMode?            nrp-types:NRP_ColorMode
|  |  |  |        +--rw nrp:rank?                 nrp-types:NRP_PositiveInteger
|  |  |  |        +--rw nrp:tokenRequestOffset?   nrp-types:NRP_NaturalNumber
|  |  |  +--rw adapterPropertySpecList* [uuid]
|  |  |  |  +--rw uuid    string
|  |  |  +--rw providerViewSpec
|  |  |  +--rw serverSpecList* [uuid]
|  |  |     +--rw uuid    string
|  |  +--rw configuredClientCapacity?   string
|  |  +--rw lpDirection?                onf-cnt:TerminationDirection
|  |  +--rw terminationState?           string
|  +--rw ltpSpec
|  +--rw ltpDirection?   onf-cnt:TerminationDirection
```

Unified Secure Channel

Overview

The Unified Secure Channel (USC) feature provides REST API, manager, and plugin for unified secure channels. The REST API provides a northbound api. The manager monitors, maintains, and provides channel related services. The plugin handles the lifecycle of channels.

USC Channel Architecture

- USC Agent
  - The USC Agent provides proxy and agent functionality on top of all standard protocols supported by the device. It initiates call-home with the controller, maintains live connections with with the controller, acts as a demuxer/muxer for packets with the USC header, and authenticates the controller.
- USC Plugin
  - The USC Plugin is responsible for communication between the controller and the USC agent . It responds to call-home with the controller, maintains live connections with the devices, acts as a muxer/demuxer for packets with the USC header, and provides support for TLS/DTLS.
- USC Manager
  - The USC Manager handles configurations, high availability, security, monitoring, and clustering support for USC.
- USC UI
  - The USC UI is responsible for displaying a graphical user interface representing the state of USC in the OpenDaylight DLUX UI.

USC Channel APIs and Interfaces

This section describes the APIs for interacting with the unified secure channels.

USC Channel Topology API

The USC project maintains a topology that is YANG-based in MD-SAL. These models are available via RESTCONF.

- Name: view-channel
- URL: http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/restconf/operations/usc-channel:view-channel
- Description: Views the current state of the USC environment.

API Reference Documentation

Go to http://\protect\T1\textdollar\protect\T1\textbraceleftipaddress\protect\T1\textbraceright:8181/apidoc/explorer/index.html, sign in, and expand the usc-channel panel. From there, users can execute various API calls to test their USC deployment.

Virtual Tenant Network (VTN)

OpenDaylight Virtual Tenant Network (VTN) Overview

OpenDaylight Virtual Tenant Network (VTN) is an application that provides multi-tenant virtual network on an SDN controller.

Conventionally, huge investment in the network systems and operating expenses are needed because the network is configured as a silo for each department and system. Therefore various network appliances must be installed for each tenant and those boxes cannot be shared with others. It is a heavy work to design, implement and operate the entire complex network.

The uniqueness of VTN is a logical abstraction plane. This enables the complete separation of logical plane from physical plane. Users can design and deploy any desired network without knowing the physical network topology or bandwidth restrictions.

VTN allows the users to define the network with a look and feel of conventional L2/L3 network. Once the network is designed on VTN, it will automatically be mapped into underlying physical network, and then configured on the individual switch leverage SDN control protocol. The definition of logical plane makes it possible not only to hide the complexity of the underlying network but also to better manage network resources. It achieves reducing reconfiguration time of network services and minimizing network configuration errors. OpenDaylight Virtual Tenant Network (VTN) is an application that provides multi-tenant virtual network on an SDN controller. It provides API for creating a common virtual network irrespective of the physical network.

It is implemented as two major components

- *VTN Manager*
- *VTN Coordinator*

VTN Manager

An OpenDaylight Plugin that interacts with other modules to implement the components of the VTN model. It also provides a REST interface to configure VTN components in OpenDaylight. VTN Manager is implemented as one plugin to the OpenDaylight. This provides a REST interface to create/update/delete VTN components. The user command in VTN Coordinator is translated as REST API to VTN Manager by the OpenDaylight Driver component. In addition to the above mentioned role, it also provides an implementation to the OpenStack L2 Network Functions API.

Function Outline

The table identifies the functions and the interface used by VTN Components:

Component	Interface	Purpose
VTN Manager	RESTful API	Configure VTN Virtualization model components in OpenDaylight
VTN Manager	Neutron API implementation	Handle Networks API from OpenStack (Neutron Interface)
VTN Coordinator	RESTful API	(1) Uses the RESTful interface of VTN Manager and configures VTN Virtualization model components in OpenDaylight. (2) Handles multiple OpenDaylight orchestration. (3) Provides API to read the physical network details. See samples for usage.

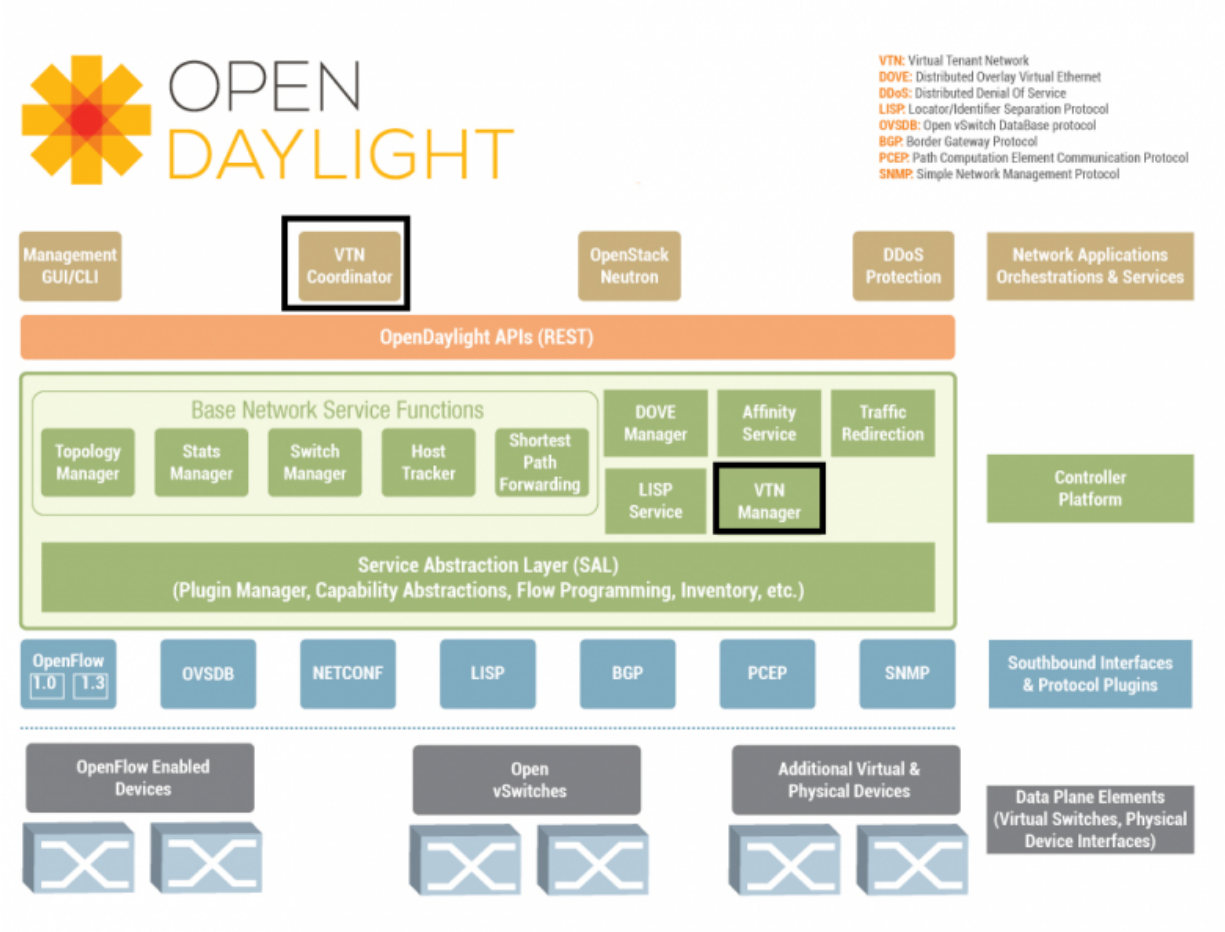


Fig. 2.78: VTN Architecture

Feature Overview

There are three features

- **odl-vtn-manager** provides VTN Manager's JAVA API.
- **odl-vtn-manager-rest** provides VTN Manager's REST API.
- **odl-vtn-manager-neutron** provides the integration with Neutron interface.

REST Conf documentation for VTN Manager, please refer to: https://nexus.opendaylight.org/content/sites/site/org.opendaylight.vtn/boron/manager.model/apidocs/index.html

For VTN Java API documentation, please refer to: https://nexus.opendaylight.org/content/sites/site/org.opendaylight.vtn/boron/apidocs/index.html

Once the Karaf distribution is up, install dlux and apidocs.

```
feature:install odl-dlux-core odl-dluxapps-applications odl-mdsal-apidocs
```

Logging In

To Log in to DLUX, after installing the application:

- Open a browser and enter the login URL as http://<OpenDaylight-IP>:8181/index.html

---

Note: Replace "<OpenDaylight-IP>" with the IP address of OpenDaylight based on your environment.

- Login to the application with user ID and password credentials as admin.

Note: admin is the only default user available for DLUX in this release.

- In the right hand side frame, click "Yang UI".

YANG documentation for VTN Manager, please refer to: https://nexus.opendaylight.org/content/sites/site/org.opendaylight.vtn/boron/manager.model/apidocs/index.html

VTN Coordinator

The VTN Coordinator is an external application that provides a REST interface for an user to use OpenDaylight VTN Virtualization. It interacts with the VTN Manager plugin to implement the user configuration. It is also capable of multiple OpenDaylight orchestration. It realizes VTN provisioning in OpenDaylight instances. In the OpenDaylight architecture VTN Coordinator is part of the network application, orchestration and services layer. VTN Coordinator will use the REST interface exposed by the VTN Manger to realize the virtual network using OpenDaylight. It uses OpenDaylight APIs (REST) to construct the virtual network in OpenDaylight instances. It provides REST APIs for northbound VTN applications and supports virtual networks spanning across multiple OpenDaylight by coordinating across OpenDaylight.

VTN Coordinator Components:

- Transaction Coordinator
- Unified Provider Physical Layer (UPPL)
- Unified Provider Logical LAyer (UPLL)

- OpenDaylight Controller Diver (ODC Driver)

OpenDaylight Virtual Tenant Network (VTN) API Overview

The VTN API module is a sub component of the VTN Coordinator and provides the northbound REST API interface for VTN applications. It consists of two subcomponents:

- Web Server
- VTN service Java API Library

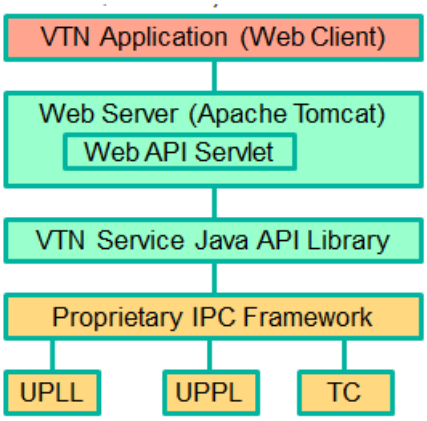


Fig. 2.79: VTN-Coordinator_api-architechture

Web Server

The Web Server module handles the REST APIs received from the VTN applications. It translates the REST APIs to the appropriate Java APIs.

The main functions of this module are:

- Starts via the startup script `catalina.sh`.
- VTN Application sends HTTP request to Web server in XML or JSON format.
- Creates a session and acquire a read/write lock.
- Invokes the VTN Service Java API Library corresponding to the specified URI.
- Returns the response to the VTN Application.

WebServer Class Details

The list below shows the classes available for Web Server module and their descriptions:

Init Manager It is a singleton class for executing the acquisition of configuration information from properties file, log initialization, initialization of VTN Service Java API Library. Executed by init() of VtnServiceWebAPIServlet.

Configuration Manager Maintains the configuration information acquired from properties file.

VtnServiceCommonUtil Utility class

VtnServiceWebUtil Utility class

VtnServiceWebAPIServlet Receives HTTP request from VTN Application and calls the method of corresponding VtnServiceWebAPIHandler. herits class HttpServlet, and overrides doGet(), doPut(), doDelete(), doPost().

VtnServiceWebAPIHandler Creates JsonObject(com.google.gson) from HTTP request, and calls method of corresponding VtnServiceWebAPIController.

VtnServiceWebAPIController Creates RestResource() class and calls UPLL API/UPPL API through Java API. the time of calling UPLL API/UPPL API, performs the creation/deletion of session, acquisition/release of configuration mode, acquisition/release of read lock by TC API through Java API.

Data Converter Converts HTTP request to JsonObject and JsonXML to JSON.

VTN Service Java API Library

It provides the Java API library to communicate with the lower layer modules in the VTN Coordinator. The main functions of this library are:

- Creates an IPC client session to the lower layer.
- Converts the request to IPC framework format.
- Invokes the lower layer API (i.e. UPPL API, UPLL API, TC API).
- Returns the response from the lower layer to the web server
- VTN Service Java API Library Class Details

Feature Overview

VTN Coordinator doesn't have Karaf features.

For VTN Coordinator REST API, please refer to: https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_%28VTN%29:VTN_Coordinator:RestApi

Usage Examples

- L2 Network using Single Controller

YANG Tools Developer Guide

Overview

YANG Tools is set of libraries and tooling providing support for use YANG for Java (or other JVM-based language) projects and applications.

YANG Tools provides following features in OpenDaylight:

- parsing of YANG sources and semantic inference of relationship across YANG models as defined in RFC6020
- representation of YANG-modeled data in Java
  - **Normalized Node** representation - DOM-like tree model, which uses conceptual meta-model more tailored to YANG and OpenDaylight use-cases than a standard XML DOM model allows for.
- serialization / deserialization of YANG-modeled data driven by YANG models
  - XML - as defined in RFC6020

  - JSON - as defined in draft-lhotka-netmod-yang-json-01
  - support for third-party generators processing YANG models.

Architecture

YANG Tools project consists of following logical subsystems:

- **Commons** - Set of general purpose code, which is not specific to YANG, but is also useful outside YANG Tools implementation.
- **YANG Model and Parser** - YANG semantic model and lexical and semantic parser of YANG models, which creates in-memory cross-referenced represenation of YANG models, which is used by other components to determine their behaviour based on the model.
- **YANG Data** - Definition of Normalized Node APIs and Data Tree APIs, reference implementation of these APIs and implementation of XML and JSON codecs for Normalized Nodes.
- **YANG Maven Plugin** - Maven plugin which integrates YANG parser into Maven build lifecycle and provides code-generation framework for components, which wants to generate code or other artefacts based on YANG model.

Concepts

Project defines base concepts and helper classes which are project-agnostic and could be used outside of YANG Tools project scope.

Components

- yang-common
- yang-data-api
- yang-data-codec-gson
- yang-data-codec-xml
- yang-data-impl
- yang-data-jaxen
- yang-data-transform
- yang-data-util
- yang-maven-plugin
- yang-maven-plugin-it
- yang-maven-plugin-spi
- yang-model-api
- yang-model-export
- yang-model-util
- yang-parser-api
- yang-parser-impl

YANG Model API

Class diagram of yang model API

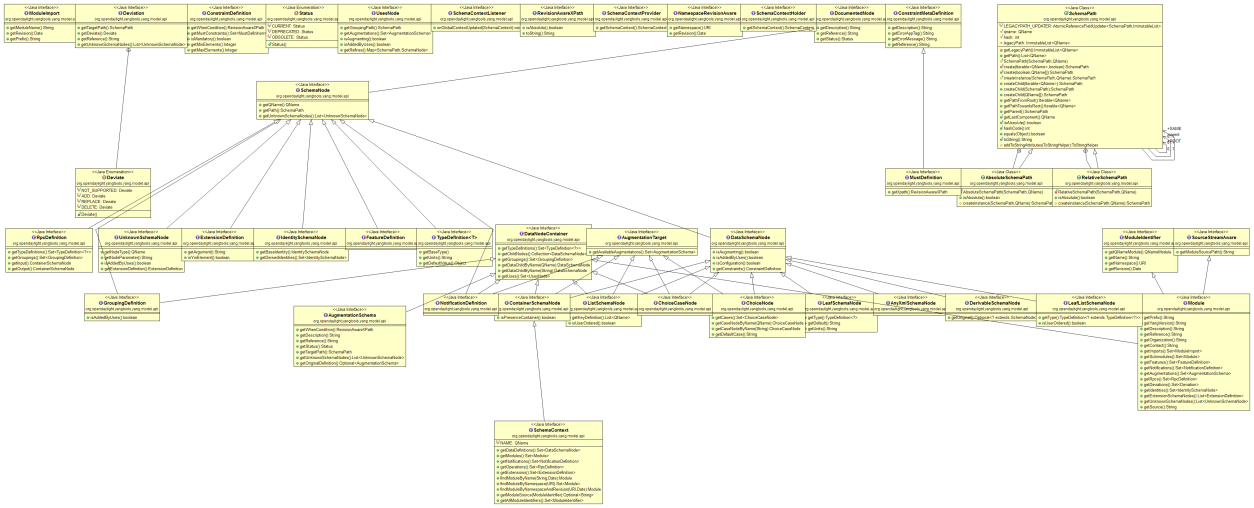


Fig. 2.80: YANG Model API

YANG Parser

Yang Statement Parser works on the idea of statement concepts as defined in RFC6020, section 6.3. We come up here with basic ModelStatement and StatementDefinition, following RFC6020 idea of having sequence of statements, where every statement contains keyword and zero or one argument. ModelStatement is extended by DeclaredStatement (as it comes from source, e.g. YANG source) and EffectiveStatement, which contains other substatements and tends to represent result of semantic processing of other statements (uses, augment for YANG). IdentifierNamespace represents common superclass for YANG model namespaces.

Input of the Yang Statement Parser is a collection of StatementStreamSource objects. StatementStreamSource interface is used for inference of effective model and is required to emit its statements using supplied StatementWriter. Each source (e.g. YANG source) has to be processed in three steps in order to emit different statements for each step. This package provides support for various namespaces used across statement parser in order to map relations during declaration phase process.

Currently, there are two implementations of StatementStreamSource in Yangtools:

- YangStatementSourceImpl - intended for yang sources
- YinStatementSourceImpl - intended for yin sources

YANG Data API

Class diagram of yang data API

YANG Data Codecs

Codecs which enable serialization of NormalizedNodes into YANG-modeled data in XML or JSON format and deserialization of YANG-modeled data in XML or JSON format into NormalizedNodes.

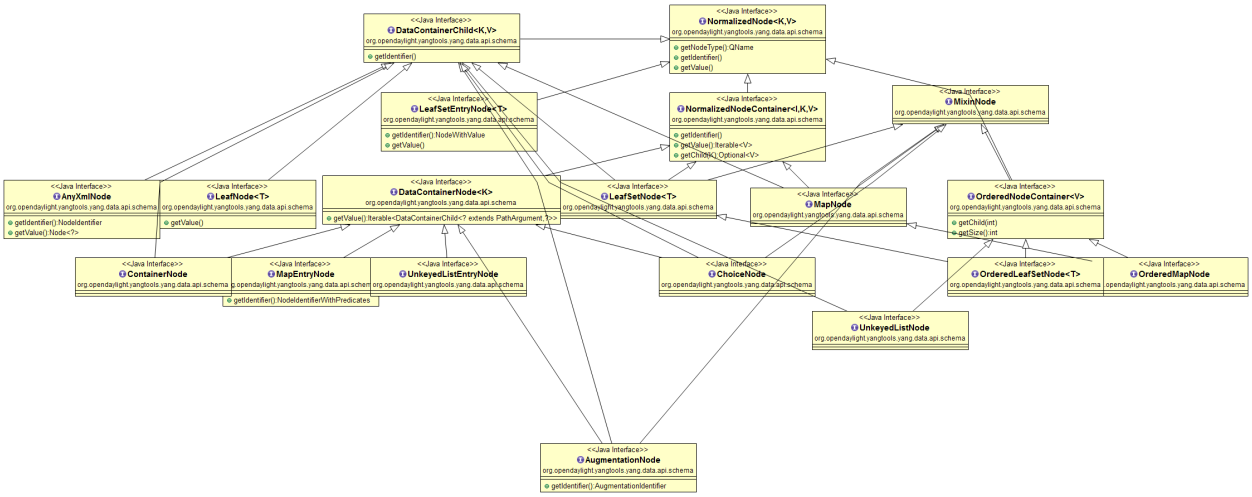


Fig. 2.81: YANG Data API

YANG Maven Plugin

Maven plugin which integrates YANG parser into Maven build lifecycle and provides code-generation framework for components, which wants to generate code or other artefacts based on YANG model.

How to / Tutorials

Working with YANG Model

First thing you need to do if you want to work with YANG models is to instantiate a SchemaContext object. This object type describes one or more parsed YANG modules.

In order to create it you need to utilize YANG statement parser which takes one or more StatementStreamSource objects as input and then produces the SchemaContext object.

StatementStreamSource object contains the source file information. It has two implementations, one for YANG sources - YangStatementSourceImpl, and one for YIN sources - YinStatementSourceImpl.

Here is an example of creating StatementStreamSource objects for YANG files, providing them to the YANG statement parser and building the SchemaContext:

```
StatementStreamSource yangModuleSource == new YangStatementSourceImpl("/example.yang",
↪ false);
StatementStreamSource yangModuleSource2 == new YangStatementSourceImpl("/example2.yang
↪", false);

CrossSourceStatementReactor.BuildAction reactor == YangInferencePipeline.RFC6020_
↪REACTOR.newBuild();
reactor.addSources(yangModuleSource, yangModuleSource2);

SchemaContext schemaContext == reactor.buildEffective();
```

First, StatementStreamSource objects with two constructor arguments should be instantiated: path to the yang source file (which is a regular String object) and a boolean which determines if the path is absolute or relative.

Next comes the initiation of new yang parsing cycle - which is represented by CrossSourceStatementReactor.BuildAction object. You can get it by calling method newBuild() on CrossSourceStatementReactor object (RFC6020_REACTOR) in YangInferencePipeline class.

Then you should feed yang sources to it by calling method addSources() that takes one or more StatementStreamSource objects as arguments.

Finally you call the method buildEffective() on the reactor object which returns EffectiveSchemaContext (that is a concrete implementation of SchemaContext). Now you are ready to work with contents of the added yang sources.

Let us explain how to work with models contained in the newly created SchemaContext. If you want to get all the modules in the schemaContext, you have to call method getModules() which returns a Set of modules. If you want to get all the data definitions in schemaContext, you need to call method getDataDefinitions, etc.

```
Set<Module> modules == schemaContext.getModules();
Set<DataSchemaNodes> dataSchemaNodes == schemaContext.getDataDefinitions();
```

Usually you want to access specific modules. Getting a concrete module from SchemaContext is a matter of calling one of these methods:

- findModuleByName(),
- findModuleByNamespace(),
- findModuleByNamespaceAndRevision().

In the first case, you need to provide module name as it is defined in the yang source file and module revision date if it specified in the yang source file (if it is not defined, you can just pass a null value). In order to provide the revision date in proper format, you can use a utility class named SimpleDateFormatUtil.

```
Module exampleModule == schemaContext.findModuleByName("example-module", null);
// or
Date revisionDate == SimpleDateFormatUtil.getRevisionFormat().parse("2015-09-02");
Module exampleModule == schemaContext.findModuleByName("example-module",␣
↪revisionDate);
```

In the second case, you have to provide module namespace in form of an URI object.

```
Module exampleModule == schema.findModuleByNamespace(new URI("opendaylight.org/
↪example-module"));
```

In the third case, you provide both module namespace and revision date as arguments.

Once you have a Module object, you can access its contents as they are defined in YANG Model API. One way to do this is to use method like getIdentities() or getRpcs() which will give you a Set of objects. Otherwise you can access a DataSchemaNode directly via the method getDataChildByName() which takes a QName object as its only argument. Here are a few examples.

```
Set<AugmentationSchema> augmentationSchemas == exampleModule.getAugmentations();
Set<ModuleImport> moduleImports == exampleModule.getImports();

ChoiceSchemaNode choiceSchemaNode == (ChoiceSchemaNode) exampleModule.
↪getDataChildByName(QName.create(exampleModule.getQNameModule(), "example-choice"));

ContainerSchemaNode containerSchemaNode == (ContainerSchemaNode) exampleModule.
↪getDataChildByName(QName.create(exampleModule.getQNameModule(), "example-container
↪"));
```

The YANG statement parser can work in three modes:

- default mode

- mode with active resolution of if-feature statements
- mode with active semantic version processing

The default mode is active when you initialize the parsing cycle as usual by calling the method newBuild() without passing any arguments to it. The second and third mode can be activated by invoking the newBuild() with a special argument. You can either activate just one of them or both by passing proper arguments. Let us explain how these modes work.

Mode with active resolution of if-features makes yang statements containing an if-feature statement conditional based on the supported features. These features are provided in the form of a QName-based java.util.Set object. In the example below, only two features are supported: example-feature-1 and example-feature-2. The Set which contains this information is passed to the method newBuild() and the mode is activated.

```
Set<QName> supportedFeatures = ImmutableSet.of(
    QName.create("example-namespace", "2016-08-31", "example-feature-1"),
    QName.create("example-namespace", "2016-08-31", "example-feature-2"));

CrossSourceStatementReactor.BuildAction reactor = YangInferencePipeline.RFC6020_
↪REACTOR.newBuild(supportedFeatures);
```

In case when no features should be supported, you should provide an empty Set<QName> object.

```
Set<QName> supportedFeatures = ImmutableSet.of();

CrossSourceStatementReactor.BuildAction reactor = YangInferencePipeline.RFC6020_
↪REACTOR.newBuild(supportedFeatures);
```

When this mode is not activated, all features in the processed YANG sources are supported.

Mode with active semantic version processing changes the way how YANG import statements work - each module import is processed based on the specified semantic version statement and the revision-date statement is ignored. In order to activate this mode, you have to provide StatementParserMode.SEMVER_MODE enum constant as argument to the method newBuild().

```
CrossSourceStatementReactor.BuildAction reactor == YangInferencePipeline.RFC6020_
↪REACTOR.newBuild(StatementParserMode.SEMVER_MODE);
```

Before you use a semantic version statement in a YANG module, you need to define an extension for it so that the YANG statement parser can recognize it.

In the example above, you see a YANG module which defines semantic version as an extension. This extension can be imported to other modules in which we want to utilize the semantic versioning concept.

Below is a simple example of the semantic versioning usage. With semantic version processing mode being active, the foo module imports the bar module based on its semantic version. Notice how both modules import the module with the semantic-version extension.

Every semantic version must have the following form: x.y.z. The x corresponds to a major version, the y corresponds to a minor version and the z corresponds to a patch version. If no semantic version is specified in a module or an import statement, then the default one is used - 0.0.0.

A major version number of 0 indicates that the model is still in development and is subject to change.

Following a release of major version 1, all modules will increment major version number when backwards incompatible changes to the model are made.

The minor version is changed when features are added to the model that do not impact current clients use of the model.

The patch version is incremented when non-feature changes (such as bugfixes or clarifications of human-readable descriptions that do not impact model functionality) are made that maintain backwards compatibility.

When importing a module with activated semantic version processing mode, only the module with the newest (highest) compatible semantic version is imported. Two semantic versions are compatible when all of the following conditions are met:

- the major version in the import statement and major version in the imported module are equal. For instance, 1.5.3 is compatible with 1.5.3, 1.5.4, 1.7.2, etc., but it is not compatible with 0.5.2 or 2.4.8, etc.
- the combination of minor version and patch version in the import statement is not higher than the one in the imported module. For instance, 1.5.2 is compatible with 1.5.2, 1.5.4, 1.6.8 etc. In fact, 1.5.2 is also compatible with versions like 1.5.1, 1.4.9 or 1.3.7 as they have equal major version. However, they will not be imported because their minor and patch version are lower (older).

If the import statement does not specify a semantic version, then the default one is chosen - 0.0.0. Thus, the module is imported only if it has a semantic version compatible with the default one, for example 0.0.0, 0.1.3, 0.3.5 and so on.

Working with YANG Data

If you want to work with YANG Data you are going to need NormalizedNode objects that are specified in the YANG Data API. NormalizedNode is an interface at the top of the YANG Data hierarchy. It is extended through sub-interfaces which define the behaviour of specific NormalizedNode types like AnyXmlNode, ChoiceNode, LeafNode, ContainerNode, etc. Concrete implemenations of these interfaces are defined in yang-data-impl module. Once you have one or more NormalizedNode instances, you can perform CRUD operations on YANG data tree which is an in-memory database designed to store normalized nodes in a tree-like structure.

In some cases it is clear which NormalizedNode type belongs to which yang statement (e.g. AnyXmlNode, ChoiceNode, LeafNode). However, there are some normalized nodes which are named differently from their yang counterparts. They are listed below:

- LeafSetNode - leaf-list
- OrderedLeafSetNode - leaf-list that is ordered-by user
- LeafSetEntryNode - concrete entry in a leaf-list
- MapNode - keyed list
- OrderedMapNode - keyed list that is ordered-by user
- MapEntryNode - concrete entry in a keyed list
- UnkeyedListNode - unkeyed list
- UnkeyedListEntryNode - concrete entry in an unkeyed list

In order to create a concrete NormalizedNode object you can use the utility class Builders or ImmutableNodes. These classes can be found in yang-data-impl module and they provide methods for building each type of normalized node. Here is a simple example of building a normalized node:

```
\\ example 1
ContainerNode containerNode == Builders.containerBuilder().withNodeIdentifier(new␣
↪YangInstanceIdentifier.NodeIdentifier(QName.create(moduleQName, "example-container
↪")).build();

\\ example 2
ContainerNode containerNode2 == Builders.containerBuilder(containerSchemaNode).
↪build();
```

Both examples produce the same result. NodeIdentifier is one of the four types of YangInstanceIdentifier (these types are described in the javadoc of YangInstanceIdentifier). The purpose of YangInstanceIdentifier is to uniquely identify

a particular node in the data tree. In the first example, you have to add NodeIdentifier before building the resulting node. In the second example it is also added using the provided ContainerSchemaNode object.

ImmutableNodes class offers similar builder methods and also adds an overloaded method called fromInstanceId() which allows you to create a NormalizedNode object based on YangInstanceIdentifier and SchemaContext. Below is an example which shows the use of this method.

```
YangInstanceIdentifier.NodeIdentifier contId == new YangInstanceIdentifier.
↪NodeIdentifier(QName.create(moduleQName, "example-container");

NormalizedNode<?, ?> contNode == ImmutableNodes.fromInstanceId(schemaContext,␣
↪YangInstanceIdentifier.create(contId));
```

Let us show a more complex example of creating a NormalizedNode. First, consider the following YANG module:

In the following example, two normalized nodes based on the module above are written to and read from the data tree.

```
TipProducingDataTree inMemoryDataTree ==     InMemoryDataTreeFactory.getInstance().
↪create(TreeType.OPERATIONAL);
inMemoryDataTree.setSchemaContext(schemaContext);

// first data tree modification
MapEntryNode parentOrderedListEntryNode == Builders.mapEntryBuilder().
↪withNodeIdentifier(
new YangInstanceIdentifier.NodeIdentifierWithPredicates(
parentOrderedListQName, parentKeyLeafQName, "pkval1"))
.withChild(Builders.leafBuilder().withNodeIdentifier(
new YangInstanceIdentifier.NodeIdentifier(parentOrdinaryLeafQName))
.withValue("plfval1").build()).build();

OrderedMapNode parentOrderedListNode == Builders.orderedMapBuilder().
↪withNodeIdentifier(
new YangInstanceIdentifier.NodeIdentifier(parentOrderedListQName))
.withChild(parentOrderedListEntryNode).build();

ContainerNode parentContainerNode == Builders.containerBuilder().withNodeIdentifier(
new YangInstanceIdentifier.NodeIdentifier(parentContainerQName))
.withChild(Builders.containerBuilder().withNodeIdentifier(
new NodeIdentifier(childContainerQName)).withChild(parentOrderedListNode).build()).
↪build();

YangInstanceIdentifier path1 == YangInstanceIdentifier.of(parentContainerQName);

DataTreeModification treeModification == inMemoryDataTree.takeSnapshot().
↪newModification();
treeModification.write(path1, parentContainerNode);

// second data tree modification
MapEntryNode childOrderedListEntryNode == Builders.mapEntryBuilder().
↪withNodeIdentifier(
new YangInstanceIdentifier.NodeIdentifierWithPredicates(
childOrderedListQName, childKeyLeafQName, "chkval1"))
.withChild(Builders.leafBuilder().withNodeIdentifier(
new YangInstanceIdentifier.NodeIdentifier(childOrdinaryLeafQName))
.withValue("chlfval1").build()).build();

OrderedMapNode childOrderedListNode == Builders.orderedMapBuilder().
↪withNodeIdentifier(
new YangInstanceIdentifier.NodeIdentifier(childOrderedListQName))
```

```
.withChild(childOrderedListEntryNode).build();

ImmutableMap.Builder<QName, Object> builder == ImmutableMap.builder();
ImmutableMap<QName, Object> keys == builder.put(parentKeyLeafQName, "pkval1").build();

YangInstanceIdentifier path2 == YangInstanceIdentifier.of(parentContainerQName).
↪node(childContainerQName)
.node(parentOrderedListQName).node(new␣
↪NodeIdentifierWithPredicates(parentOrderedListQName, keys)).
↪node(childOrderedListQName);

treeModification.write(path2, childOrderedListNode);
treeModification.ready();
inMemoryDataTree.validate(treeModification);
inMemoryDataTree.commit(inMemoryDataTree.prepare(treeModification));

DataTreeSnapshot snapshotAfterCommits == inMemoryDataTree.takeSnapshot();
Optional<NormalizedNode<?, ?>> readNode == snapshotAfterCommits.readNode(path1);
Optional<NormalizedNode<?, ?>> readNode2 == snapshotAfterCommits.readNode(path2);
```

First comes the creation of in-memory data tree instance. The schema context (containing the model mentioned above) of this tree is set. After that, two normalized nodes are built. The first one consists of a parent container, a child container and a parent ordered list which contains a key leaf and an ordinary leaf. The second normalized node is a child ordered list that also contains a key leaf and an ordinary leaf.

In order to add a child node to a node, method withChild() is used. It takes a NormalizedNode as argument. When creating a list entry, YangInstanceIdentifier.NodeIdentifierWithPredicates should be used as its identifier. Its arguments are the QName of the list, QName of the list key and the value of the key. Method withValue() specifies a value for the ordinary leaf in the list.

Before writing a node to the data tree, a path (YangInstanceIdentifier) which determines its place in the data tree needs to be defined. The path of the first normalized node starts at the parent container. The path of the second normalized node points to the child ordered list contained in the parent ordered list entry specified by the key value “pkval1”.

Write operation is performed with both normalized nodes mentioned earlier. It consist of several steps. The first step is to instantiate a DataTreeModification object based on a DataTreeSnapshot. DataTreeSnapshot gives you the current state of the data tree. Then comes the write operation which writes a normalized node at the provided path in the data tree. After doing both write operations, method ready() has to be called, marking the modification as ready for application to the data tree. No further operations within the modification are allowed. The modification is then validated - checked whether it can be applied to the data tree. Finally we commit it to the data tree.

Now you can access the written nodes. In order to do this, you have to create a new DataTreeSnapshot instance and call the method readNode() with path argument pointing to a particular node in the tree.

Serialization / deserialization of YANG Data

If you want to deserialize YANG-modeled data which have the form of an XML document, you can use the XML parser found in the module yang-data-codec-xml. The parser walks through the XML document containing YANG-modeled data based on the provided SchemaContext and emits node events into a NormalizedNodeStreamWriter. The parser disallows multiple instances of the same element except for leaf-list and list entries. The parser also expects that the YANG-modeled data in the XML source are wrapped in a root element. Otherwise it will not work correctly.

Here is an example of using the XML parser.

```
InputStream resourceAsStream == ExampleClass.class.getResourceAsStream("/example-
↪module.yang");
```

```
XMLInputFactory factory == XMLInputFactory.newInstance();
XMLStreamReader reader == factory.createXMLStreamReader(resourceAsStream);

NormalizedNodeResult result == new NormalizedNodeResult();
NormalizedNodeStreamWriter streamWriter == ImmutableNormalizedNodeStreamWriter.
↪from(result);

XmlParserStream xmlParser == XmlParserStream.create(streamWriter, schemaContext);
xmlParser.parse(reader);

NormalizedNode<?, ?> transformedInput == result.getResult();
```

The XML parser utilizes the javax.xml.stream.XMLStreamReader for parsing an XML document. First, you should create an instance of this reader using XMLInputFactory and then load an XML document (in the form of InputStream object) into it.

In order to emit node events while parsing the data you need to instantiate a NormalizedNodeStreamWriter. This writer is actually an interface and therefore you need to use a concrete implementation of it. In this example it is the ImmutableNormalizedNodeStreamWriter, which constructs immutable instances of NormalizedNodes.

There are two ways how to create an instance of this writer using the static overloaded method from(). One version of this method takes a NormalizedNodeResult as argument. This object type is a result holder in which the resulting NormalizedNode will be stored. The other version takes a NormalizedNodeContainerBuilder as argument. All created nodes will be written to this builder.

Next step is to create an instance of the XML parser. The parser itself is represented by a class named XmlParserStream. You can use one of two versions of the static overloaded method create() to construct this object. One version accepts a NormalizedNodeStreamWriter and a SchemaContext as arguments, the other version takes the same arguments plus a SchemaNode. Node events are emitted to the writer. The SchemaContext is used to check if the YANG data in the XML source comply with the provided YANG model(s). The last argument, a SchemaNode object, describes the node that is the parent of nodes defined in the XML data. If you do not provide this argument, the parser sets the SchemaContext as the parent node.

The parser is now ready to walk through the XML. Parsing is initiated by calling the method parse() on the XmlParserStream object with XMLStreamReader as its argument.

Finally you can access the result of parsing - a tree of NormalizedNodes containg the data as they are defined in the parsed XML document - by calling the method getResult() on the NormalizedNodeResult object.

Introducing schema source repositories

Writing YANG driven generators

Introducing specific extension support for YANG parser

Diagnostics

2.2 Java API Documentation

- bgpcep
- controller
- genius

- infrautils
- lispflowmapping
- mdsal
- netvirt
- odlparent
- openflowplugin
- ovsdb
- sfc
- yangtools

CHAPTER 3

Content for OpenDaylight Contributors

The following content is intended for developers who either currently participate in the development of OpenDaylight or would like to start.

3.1 Infrastructure Guide

This guide provides details into OpenDaylight Infrastructure and services.

Contents:

3.1.1 Jenkins

The Release Engineering Project consolidates the Jenkins jobs from project-specific VMs to a single Jenkins server. Each OpenDaylight project has a tab for their jobs on the jenkins-master. The system utilizes Jenkins Job Builder for the creation and management of the Jenkins jobs.

Sections:

New Project Quick Start

This section attempts to provide details on how to get going as a new project quickly with minimal steps. The rest of the guide should be read and understood by those who need to create and contribute new job types that is not already covered by the existing job templates provided by OpenDaylight's JJB repo.

As a new project you will be mainly interested in getting your jobs to appear in the jenkins-master silo and this can be achieved by simply creating a <project>.yaml in the releng/builder project's jjb directory.

```
git clone --recursive https://git.opendaylight.org/gerrit/releng/builder
cd builder
mkdir jjb/<new-project>
```

Where <new-project> should be the same name as your project's git repo in Gerrit. If your project is called "aaa" then create a new jjb/aaa directory.

Next we will create <new-project>.yaml as follows:

```
---
- project:
    name: <NEW_PROJECT>-carbon
    jobs:
      - '{project-name}-clm-{stream}'
      - '{project-name}-integration-{stream}'
      - '{project-name}-merge-{stream}'
      - '{project-name}-verify-{stream}-{maven}-{jdks}'

    project: '<NEW_PROJECT>'
    project-name: '<NEW_PROJECT>'
    stream: carbon
    branch: 'master'
    jdk: openjdk8
    jdks:
```

```
      - openjdk8
    maven:
      - mvn33:
          mvn-version: 'mvn33'
    mvn-settings: '<NEW_PROJECT>-settings'
    mvn-goals: 'clean install -Dmaven.repo.local=/tmp/r -Dorg.ops4j.pax.url.mvn.
↪localRepository=/tmp/r'
    mvn-opts: '-Xmx1024m -XX:MaxPermSize=256m'
    dependencies: 'odlparent-merge-{stream},yangtools-merge-{stream},controller-merge-
↪{stream}'
    email-upstream: '[<NEW_PROJECT>] [odlparent] [yangtools] [controller]'
    archive-artifacts: ''

- project:
    name: <NEW_PROJECT>-sonar
    jobs:
      - '{project-name}-sonar'

    project: '<NEW_PROJECT>'
    project-name: '<NEW_PROJECT>'
    branch: 'master'
    mvn-settings: '<NEW_PROJECT>-settings'
    mvn-goals: 'clean install -Dmaven.repo.local=/tmp/r -Dorg.ops4j.pax.url.mvn.
↪localRepository=/tmp/r'
    mvn-opts: '-Xmx1024m -XX:MaxPermSize=256m'
```

Replace all instances of <new-project> with the name of your project. This will create the jobs with the default job types we recommend for Java projects. If your project is participating in the simultanious-release and ultimately will be included in the final distribution, it is required to add the following job types into the job list for the release you are participating.

```
- '{project-name}-distribution-check-{stream}'
- '{project-name}-validate-autorelease-{stream}'
```

If you'd like to explore the additional tweaking options available please refer to the *Jenkins Job Templates* section.

Finally we need to push these files to Gerrit for review by the releng/builder team to push your jobs to Jenkins.

```
git add jjb/<new-project>
git commit -sm "Add <new-project> jobs to Jenkins"
git review
```

This will push the jobs to Gerrit and your jobs will appear in Jenkins once the releng/builder team has reviewed and merged your patch.

Jenkins Master

The jenkins-master is the home for all project's Jenkins jobs. All maintenance and configuration of these jobs must be done via JJB through the releng-builder-repo. Project contributors can no longer edit the Jenkins jobs directly on the server.

Build Minions

The Jenkins jobs are run on build minions (executors) which are created on an as-needed basis. If no idle build minions are available a new VM is brought up. This process can take up to 2 minutes. Once the build minion has finished a job, it will be destroyed.

Our Jenkins master supports many types of dynamic build minions. If you are creating custom jobs then you will need to have an idea of what type of minions are available. The following are the current minion types and descriptions. Minion Template Names are needed for jobs that take advantage of multiple minions as they must be specifically called out by template name instead of label.

Adding New Components to the Minions

If your project needs something added to one of the minions, you can help us get things added faster by doing one of the following:

- Submit a patch to RelEng/Builder for the appropriate *jenkins-scripts* definition which configure software during minion boot up.
- Submit a patch to RelEng/Builder for the *packer/provision* scripts that configures software during minion instance imaging.
- Submit a patch to RelEng/Builder for the Packer's templates in the *packer/templates* directory that configures a new instance definition along with changes in *packer/provision*.

Going the first route will be faster in the short term as we can inspect the changes and make test modifications in the sandbox to verify that it works.

Note: The first route may add additional setup time considering this is run every time the minion is booted.

The second and third routes, however, is better for the community as a whole as it will allow others to utilize our Packer setups to replicate our systems more closely. It is, however, more time consuming as an image snapshot needs to be created based on the updated Packer definitions before it can be attached to the Jenkins configuration on sandbox for validation testing.

In either case, the changes must be validated in the sandbox with tests to make sure that we don't break current jobs and that the new software features are operating as intended. Once this is done the changes will be merged and the updates applied to the RelEng Jenkins production silo. Any changes to files under *releng/builder/packer* will be validated and images would be built triggered by verify-packer and merge-packer jobs.

Please note that the combination of a Packer definitions from *vars*, *templates* and the *provision* scripts is what defines a given minion. For instance, a minion may be defined as *centos7-builder* which is a combination of Packer OS image definitions from *vars/centos.json*, Packer template definitions from *templates/builder.json* and spinup scripts from *provision/builder.sh*. This combination provides the full definition of the realized minion.

Jenkins starts a minion using the latest image which is built and linked into the Jenkins configuration. Once the base instance is online Jenkins checks out the RelEng/Builder repo on it and executes two scripts. The first is *provision/baseline.sh*, which is a baseline for all of the minions.

The second is the specialized script, which handles any system updates, new software installs or extra environment tweaks that don't make sense in a snapshot. Examples could include installing new package or setting up a virtual environment. Its imperative to ensure modifications to these spinup scripts have considered time taken to install the packages, as this could increase the build time for every job which runs on the image. After all of these scripts have executed Jenkins will finally attach the minion as an actual minion and start handling jobs on it.

Flavors

Performance flavors come with dedicated CPUs and are not shared with other accounts in the cloud so should ensure consistent performance.

Table 3.1: Flavors

Instance Type	CPUs	Memory
v2-standard-1	1	4
v2-standard-2	2	8
v2-standard-4	4	16
v2-standard-8	8	32
v2-standard-16	16	64
v2-highcpu-1	1	1
v2-highcpu-2	2	2
v2-highcpu-4	4	4
v2-highcpu-8	8	8
v2-highcpu-16	16	16
v2-highcpu-32	32	32

Pool: ODLVEX

Pool: ODLVEX - HOT (Heat Orchestration Templates)

HOT integration enables to spin up integration labs servers for CSIT jobs using heat, rathar than using jclouds (deprecated). Image names are updated on the project specific job templates using the variable *{odl,docker,openstack,tools}_system_image* followed by image name in the format <*platform*> - <*template*> - <*date-stamp*>.

Following are the list of published images available to be used with Jenkins jobs.

- ZZCI - CentOS 7 - autorelease - 20180125-2240
- ZZCI - CentOS 7 - builder - 20180109-0417
- ZZCI - CentOS 7 - builder - 20180110-1659
- ZZCI - CentOS 7 - builder - 20180201-2139
- ZZCI - CentOS 7 - devstack - 20171208-1648
- ZZCI - CentOS 7 - devstack-ocata - 20171208-1649
- ZZCI - CentOS 7 - devstack-pike - 20171208-1649
- ZZCI - CentOS 7 - docker - 20171209-0317
- ZZCI - CentOS 7 - docker - 20180109-0346
- ZZCI - CentOS 7 - docker - 20180110-1659
- ZZCI - CentOS 7 - java-builder - 20171206-1842
- ZZCI - CentOS 7 - java-builder - 20171209-0032
- ZZCI - CentOS 7 - robot - 20171207-1911
- ZZCI - Ubuntu 14.04 - gbp - 20171208-2336
- ZZCI - Ubuntu 16.04 - gbp - 20171213-2018
- ZZCI - Ubuntu 16.04 - mininet-ovs-25 - 20171208-1847
- ZZCI - Ubuntu 16.04 - mininet-ovs-26 - 20171208-1847

Creating Jenkins Jobs

Jenkins Job Builder takes simple descriptions of Jenkins jobs in YAML format and uses them to configure Jenkins.

- Jenkins Job Builder (JJB) documentation
- RelEng/Builder Gerrit
- RelEng/Builder Git repository

Getting Jenkins Job Builder

OpenDaylight uses Jenkins Job Builder to translate our in-repo YAML job configuration into job descriptions suitable for consumption by Jenkins. When testing new Jenkins Jobs in the *Jenkins Sandbox*, you'll need to use the *jenkins-jobs* executable to translate a set of jobs into their XML descriptions and upload them to the sandbox Jenkins server.

We document *installing jenkins-jobs* below.

Installing Jenkins Job Builder

We recommend using *pip* to assist with JJB installs, but we also document *installing from a git repository manually*. For both, we recommend using Python *Virtual Environments* to isolate JJB and its dependencies.

The builder/jjb/requirements.txt file contains the currently recommended JJB version. Because JJB is fairly unstable, it may be necessary to debug things by installing different versions. This is documented for both *pip-assisted* and *manual* installs.

Virtual Environments

For both *pip-assisted* and *manual* JJB installs, we recommend using Python Virtual Environments to manage JJB and its Python dependencies. The python-virtualenvwrapper tool can help you do so.

Documentation is available for installing python-virtualenvwrapper. On Linux systems with pip (typical), they amount to:

```
sudo pip install virtualenvwrapper
```

A virtual environment is simply a directory that you install Python programs into and then append to the front of your path, causing those copies to be found before any system-wide versions.

Create a new virtual environment for JJB.

```
# Virtaulenvwrapper uses this dir for virtual environments
$ echo $WORKON_HOME
/home/daniel/.virtualenvs
# Make a new virtual environment
$ mkvirtualenv jjb
# A new venv dir was created
(jjb)$ ls -rc $WORKON_HOME | tail -n 1
jjb
# The new venv was added to the front of this shell's path
(jjb)$ echo $PATH
/home/daniel/.virtualenvs/jjb/bin:<my normal path>
# Software installed to venv, like pip, is found before system-wide copies
(jjb)$ command -v pip
/home/daniel/.virtualenvs/jjb/bin/pip
```

With your virtual environment active, you should install JJB. Your install will be isolated to that virtual environment's directory and only visible when the virtual environment is active.

You can easily leave and return to your venv. Make sure you activate it before each use of JJB.

```
(jjb)$ deactivate
$ command -v jenkins-jobs
# No jenkins-jobs executable found
$ workon jjb
(jjb)$ command -v jenkins-jobs
$WORKON_HOME/jjb/bin/jenkins-jobs
```

Installing JJB using pip

The recommended way to install JJB is via pip.

First, clone the latest version of the releng-builder-repo.

```
$ git clone --recursive https://git.opendaylight.org/gerrit/p/releng/builder.git
```

Before actually installing JJB and its dependencies, make sure you've *created and activated* a virtual environment for JJB.

```
$ mkvirtualenv jjb
```

The recommended version of JJB to install is the version specified in the builder/jjb/requirements.txt file.

```
# From the root of the releng/builder repo
(jjb)$ pip install -r jjb/requirements.txt
```

To validate that JJB was successfully installed you can run this command:

```
(jjb)$ jenkins-jobs --version
```

TODO: Explain that only the currently merged jjb/requirements.txt is supported, other options described below are for troubleshooting only.

To change the version of JJB specified by builder/jjb/requirements.txt to install from the latest commit to the master branch of JJB's git repository:

```
$ cat jjb/requirements.txt
-e git+https://git.openstack.org/openstack-infra/jenkins-job-builder#egg=jenkins-job-
↪builder
```

To install from a tag, like 1.4.0:

```
$ cat jjb/requirements.txt
-e git+https://git.openstack.org/openstack-infra/jenkins-job-builder@1.4.0
↪#egg=jenkins-job-builder
```

Installing JJB Manually

This section documents installing JJB from its manually cloned repository.

Note that *installing via pip* is typically simpler.

Checkout the version of JJB's source you'd like to build.

For example, using master:

```
$ git clone https://git.openstack.org/openstack-infra/jenkins-job-builder
```

Using a tag, like 1.4.0:

```
$ git clone https://git.openstack.org/openstack-infra/jenkins-job-builder
$ cd jenkins-job-builder
$ git checkout tags/1.4.0
```

Before actually installing JJB and its dependencies, make sure you've *created and activated* a virtual environment for JJB.

```
$ mkvirtualenv jjb
```

You can then use JJB's requirements.txt file to install its dependencies. Note that we're not using *sudo* to install as root, since we want to make use of the venv we've configured for our current user.

```
# In the cloned JJB repo, with the desired version of the code checked out
(jjb)$ pip install -r requirements.txt
```

Then install JJB from the repo with:

```
(jjb)$ pip install .
```

To validate that JJB was successfully installed you can run this command:

```
(jjb)$ jenkins-jobs --version
```

Jenkins Job Templates

The OpenDaylight RelEng/Builder project provides jjb-templates that can be used to define basic jobs.

The *Gerrit Trigger* listed in the jobs are keywords that can be used to trigger the job to run manually by simply leaving a comment in Gerrit for the patch you wish to trigger against.

All jobs have a default build-timeout value of 360 minutes (6 hrs) but can be overrided via the opendaylight-infra-wrappers' build-timeout property.

TODO: Group jobs into categories: every-patch, after-merge, on-demand, etc. TODO: Reiterate that "remerge" triggers all every-patch jobs at once, because when only a subset of jobs is triggered, Gerrit forgets valid -1 from jobs outside the subset. TODO: Document that only drafts and commit-message-only edits do not trigger every-patch jobs. TODO: Document test-{project}-{feature} and test-{project}-all.

Maven Properties

We provide a properties which your job can take advantage of if you want to do something different depending on the job type that is run. If you create a profile that activates on a property listed blow. The JJB templated jobs will be able to activate the profile during the build to run any custom code you wish to run in your project.

```
-Dmerge   : This flag is passed in our Merge job and is equivalent to the
            Maven property
            <merge>true</merge>.
-Dsonar   : This flag is passed in our Sonar job and is equivalent to the
            Maven property
            <sonar>true</sonar>.
```

Jenkins Sandbox

The jenkins-sandbox instance's purpose is to allow projects to test their JJB setups before merging their code over to the RelEng master silo. It is configured similarly to the master instance, although it cannot publish artifacts or vote in Gerrit.

If your project requires access to the sandbox please open an OpenDaylight Helpdesk ticket (<helpdesk@opendaylight.org>) and provide your ODL ID.

Notes Regarding the Sandbox

- Jobs are automatically deleted every Saturday at 08:00 UTC
- Committers can login and configure Jenkins jobs in the sandbox directly (unlike with the master silo)
- Sandbox configuration mirrors the master silo when possible
- Sandbox jobs can NOT upload artifacts to Nexus
- Sandbox jobs can NOT vote on Gerrit

Configuration

Make sure you have Jenkins Job Builder [properly installed](#jjb_install).

If you do not already have access, open an OpenDaylight Helpdesk ticket (<helpdesk@opendaylight.org>) to request access to ODL's sandbox instance. Integration/Test (integration-test-wiki) committers have access by default.

JJB reads user-specific configuration from a jenkins.ini. An example is provided by releng/builder at example-jenkins.ini.

```
# If you don't have RelEng/Builder's repo, clone it
$ git clone --recursive https://git.opendaylight.org/gerrit/p/releng/builder.git
# Make a copy of the example JJB config file (in the builder/ directory)
$ cp jenkins.ini.example jenkins.ini
# Edit jenkins.ini with your username, API token and ODL's sandbox URL
$ cat jenkins.ini
<snip>
[jenkins]
user=<your ODL username>
password=<your ODL Jenkins sandbox API token>
url=https://jenkins.opendaylight.org/sandbox
<snip>
```

To get your API token, login to the Jenkins **sandbox** instance (*not the main master Jenkins instance, different tokens*), go to your user page (by clicking on your username, for example), click "Configure" and then "Show API Token".

Manual Method

If you *installed JJB locally into a virtual environment*, you should now activate that virtual environment to access the *jenkins-jobs* executable.

```
$ workon jjb
(jjb)$
```

You'll want to work from the root of the RelEng/Builder repo, and you should have your *jenkins.ini* file [properly configured](#sandbox_config).

Testing Jobs

It's good practice to use the *test* command to validate your JJB files before pushing them.

```
jenkins-jobs --conf jenkins.ini test jjb/ <job-name>
```

If the job you'd like to test is a template with variables in its name, it must be manually expanded before use. For example, the commonly used template *{project}-csit-verify-1node-{functionality}* might expand to *ovsdb-csit-verify-1node-netvirt*.

```
jenkins-jobs --conf jenkins.ini test jjb/ ovsdb-csit-verify-1node-netvirt
```

Successful tests output the XML description of the Jenkins job described by the specified JJB job name.

Pushing Jobs

Once you've *configured your 'jenkins.ini'* and *verified your JJB jobs* produce valid XML descriptions of Jenkins jobs you can push them to the Jenkins sandbox.

Important: When pushing with *jenkins-jobs*, a log message with the number of jobs you're pushing will be issued, typically to stdout. **If the number is greater than 1** (or the number of jobs you passed to the command to push) then you are pushing too many jobs and should **'ctrl+c' to cancel the upload**. Else you will flood the system with jobs.

```
INFO:jenkins_jobs.builder:Number of jobs generated:  1
```

Failing to provide the final '<job-name>' param will push all jobs!

```
# Don't push all jobs by omitting the final param! (ctrl+c to abort)
jenkins-jobs --conf jenkins.ini update jjb/ <job-name>
```

Alternatively, you can push a job to the Jenkins sandbox with a special comment in a releng/builder gerrit patch. The job will be based off of the code your patch is based upon. Meaning, if your patch is changing something related to the job you are pushing, those changes will exist in the sandbox job. The format of the comment is:

```
jjb-deploy <job name>
```

Running Jobs

Once you have your Jenkins job configuration *pushed to the Sandbox* you can trigger it to run.

Find your newly-pushed job on the Sandbox's web UI. Click on its name to see the job's details.

Make sure you're logged in to the Sandbox.

Click "Build with Parameters" and then "Build".

Wait for your job to be scheduled and run. Click on the job number to see details, including console output.

Make changes to your JJB configuration, re-test, re-push and re-run until your job is ready.

3.1.2 Release Workflow

This page documents the workflow for releasing for projects that are not built and released via the Autorelease project.

Sections:

Workflow

OpenDaylight uses Nexus as it's artifact repository for releasing artifacts to the world. The workflow involves using Nexus to produce a staging repository which can be tested and reviewed before being approved to copy to the final destination opendaylight.release repo. The workflow in general is as follows:

1. Project create release tag and push to Gerrit
2. Project will contact helpdesk@opendaylight.org with project name and build tag to produce a release candidate / staging repo
3. Helpdesk will run a build and notify project of staging repo location
4. Project tests staging repo and notifies Helpdesk with go ahead to release
5. Helpdesk clicks Release repo button in Nexus
6. (optional) Helpdesk runs Jenkins job to push update-site.zip to p2repos sites repo

Step 6 is only necessary for Eclipse projects that need to additionally deploy an update site to a webserver.

Release Job

There is a JJB template release job which should be used for a project if the project needs to produce a staging repo for release. The supported Job types are listed below, use the one relevant to your project.

Maven|Java {name}-release-java – this job type will produce a staging repo in Nexus for Maven projects.

P2 Publisher {name}-publish-p2repo – this job type is useful for projects that produce a p2 repo that needs to be published to a special URL.

3.2 Integration Testing Guide

The Integration Testing Guide provides details on how to contribute test code to OpenDaylight.

Contents:

3.2.1 Cluster testing

Contents:

Carbon cluster testing

Contents:

Description of test scenarios

This is a test plan written around M1 of Carbon cycle.

During the cycle several limitations were found, which resulted in tests which implement the scenarios is ways different from what is described here.

For list of limitations and differences, see caveats page. For more detailed descriptions of test cases as implemented, see test description page.

Controller Cluster Service Functional Tests

The purpose of functional tests is to establish a known baseline behavior for basic services exposed to application plugins when the cluster member nodes encounter problems.

Isolation Mechanics Three-node scenarios executed in tests below need to be repeated for three distinct modes of isolation:

1. JVM freeze, initiated by 'kill -STOP <pid>' on the JVM process, followed by a 'kill -CONT <pid>' after three minutes. This simulates a long-running garbage collection cycle, VM suspension or similar, after which the JVM recovers without losing state and scheduled timers going off simultaneously.
2. Network-level isolation via firewalling. Simulates a connectivity issue between member nodes, while all nodes continue to work as usual. This should be done by firewalling all traffic to and from the target node.
3. JVM restart. This simulates a hard error, such as JVM error, VM reboot, and similar. The JVM loses its state and the scenario tests whether the failed node is able to result its operations as a member of the cluster.

Leader Shutdown The Shard implementation allows a leader to be shut down at run time, which is expected to perform a clean hand over to a new leader, elected from the remaining shard members.

DOMDataBroker

Also known as 'the datastore', provides MVCC transaction and data change notifications.

Leader Stability

The goal is to ensure that a single-established shard does not flap, i.e. does not trigger leader movement by causing crashes or timeouts. This is performed by having the BGP load generator run injection of 1 million prefixes, followed by their removal.

This test is executed in three scenarios:

- Single node
- Three-node, with shard leader being local
- Three-node, with shard leader being remote

Success criteria are:

- Both injection and removal succeed
- No transaction errors reported to the generator
- No leader movement on the backend

Clean Leader Shutdown

The goal is to ensure that applications do not observe disruption when a shard leader is shut down cleanly. This is performed by having a steady-stream producer execute operations against the shard and then initiate leader shard shutdown, then the producer is shut down cleanly.

This test is executed in two scenarios:

- Three-node, with shard leader being local
- Three-node, with shard leader being remote

Success criteria are:

- No transaction errors occur
- Producer shuts down cleanly (i.e. all transactions complete successfully)

Test tool: *test-transaction-producer*, running at 1K tps

- Steady, configurable producer started with:
- A transaction chain
- Single transactions (note: these cannot overlap)
- Configurable transaction rate (i.e. transactions-per-second)
- Single-operation transactions
- Random mix across 1M entries

Explicit Leader Movement

The goal is to ensure that applications do not observe disruption when a shard leader is moved as the result of explicit application request. This is performed by having a steady-stream producer execute operations against the shard and then initiate shard leader shutdown, then the producer is shut down cleanly.

This test is executed in three scenarios:

- Three-node, with shard leader being local and becoming remote
- Three-node, with shard leader being remote and remaining remote
- Three-node, with shard leader being remote and becoming local

Success criteria are:

- No transaction errors occur
- Producer shuts down cleanly (i.e. all transactions complete successfully)

Test tool: test-transaction-producer, running at 1K tps Test tool: *test-leader-mover*

- Uses cds-dom-api to request shard movement

Leader Isolation

The goal is to ensure the datastore succeeds in basic isolation/rejoin scenario, simulating either a network partition, or a prolonged GC pause.

This test is executed in the following two scenarios:

- Three-node, partition heals within TRANSACTION_TIMEOUT

- Three-node, partition heals after 2*TRANSACTION_TIMEOUT

Using following steps:

1. Start test-transaction producer, running at 1K tps, non-overlapping, from all nodes to a single shard
2. Isolate leader
3. Wait for followers to initiate election
4. Un-isolate leader
5. Wait for partition to heal
6. Restart failed producer

Success criteria:

- Followers win election in 3
- No transaction failures occur if the partition is healed within TRANSACTION_TIMEOUT
- Producer on old leader works normally after step 6)

Test tool: test-transaction-producer

Client Isolation

The purpose of this test is to ascertain that the failure modes of cds-access-client work as expected. This is performed by having a steady stream of transactions flowing from the frontend and isolating the node hosting the frontend from the rest of the cluster.

This test is executed in one scenario:

- Three node, test-transaction-producer running on a non-leader
- Three node, test-transaction-producer running on the leader

Success criteria:

- After TRANSACTION_TIMEOUT failures occur
- After HARD_TIMEOUT client aborts

Test tool: test-transaction-producer

Listener Isolation

The goal is to ensure listeners do no observe disruption when the leader moves. This is performed by having a steady stream of transactions being observed by the listeners and having the leader move.

This test is executed in two scenarios:

- Three node, test-transaction-listener running on the leader
- Three node, test-transaction-listener running on a non-leader

Using these steps:

- Start the listener on target node
- Start test-transaction-producer on each node, with 1K tps, non-overlapping data
- Trigger shard movement by shutting down shard leader

- Stop producers without erasing data
- Stop listener

Success criteria:

- Listener-internal data tree has to match data stored in the data tree

Test tool: *test-transaction-listener*

- Subscribes a DTCL to multiple subtrees (as specified)
- DTCL applies reported changes to an internal DataTree

DOMRpcBroker

Responsible for routing RPC requests to their implementations and routing responses back to the caller.

RPC Provider Precedence

The aim is to establish that remote RPC implementations have lower priority than local ones, which is to say that any movement of RPCs on remote nodes does not affect routing as long as a local implementation is available.

Test is executed only in a three-node scenario, using the following steps:

1. Register an RPC implementation on each node
2. Invoke RPC on each node
3. Unregister implementation on one node
4. Invoke RPC on that node
5. Re-register implementation on than node
6. Invoke RPC on that node

Success criteria:

- Invocation in steps 2) and 6) results in a response from local node
- Invocation in step 4) results in a response from one of the other two nodes

RPC Provider Partition and Heal

This tests establishes that the RPC service operates correctly when faced with node failures.

Test is executed only in a three-node scenario, using the following steps:

1. Register an RPC implementation on two nodes
2. Invoke RPC on each node
3. Isolate one of the nodes where RPC is registered
4. Invoke RPC on each node
5. Un-isolate the node
6. Invoke RPC on all nodes

Success criteria:

- Step 2) routes the RPC the node nearest node (local or remote)
- Step 4) works, routing the RPC request to the implementation in the same partition
- Step 6) routes the RPC the node nearest node (local or remote)

Action Provider Precedence

The aim is to establish that remote action implementations have lower priority than local ones, which is to say that any movement of actions on remote nodes does not affect routing as long as a local implementation is available.

Test is executed only in a three-node scenario, using the following steps:

1. Register an action implementation on each node
2. Invoke action on each node
3. Unregister implementation on one node
4. Invoke action on that node
5. Re-register implementation on than node
6. Invoke action on that node

Success criteria:

- Invocation in steps 2) and 6) results in a response from local node
- Invocation in step 4) results in a response from one of the other two nodes

Action Provider Partition and Heal

This tests establishes that the RPC service for actions operates correctly when faced with node failures.

Test is executed only in a three-node scenario, using the following steps:

1. Register an action implementation on two nodes
2. Invoke action on each node
3. Isolate one of the nodes where RPC is registered
4. Invoke action on each node
5. Un-isolate the node
6. Invoke action on all nodes

Success criteria:

- Step 2) routes the action request the node nearest node (local or remote)
- Step 4) works, routing the action request to the implementation in the same partition
- Step 6) routes the RPC the node nearest node (local or remote)

DOMNotificationBroker

Provides routing of YANG notifications from publishers to subscribers.

No-loss rate

The purpose of this test is to determine the broker can forward messages without loss. We do this on a single-node setup by incrementally adding publishers and subscribers.

This test is executed in one scenario:

- Single-node

Steps:

- Start test-notification-subscriber
- Start test-notification-publisher at 5K notifications/sec
- Run for 5 minutes, verify no notifications lost
- Add another pair of publisher/subscriber, repeat for rate of 60K notifications/sec

Success criteria:

- No notifications lost at rate of 60K notifications/sec

Test tool: *test-notification-publisher*

- Publishes notifications containing instance id and sequence number
- Configurable rate (i.e. notifications-per-second)

Test tool: *test-notification-subscriber*

- Subscribes to specified notifications from publisher
- Verifies notification sequence numbers
- Records total number of notifications received and number of sequence errors

Cluster Singleton

Cluster Singleton service is designed to ensure that only one instance of an application is registered globally in the cluster.

Master Stability

The goal is to establish the service operates correctly in face of application registration changing without moving the active instance.

The test is performed in a three-node cluster using following steps:

1. Register candidate on each node
2. Wait for master activation
3. Remove non-master candidate,
4. Wait one minute
5. Restore the removed candidate

Success criteria:

- After step 2) there is exactly one master in the cluster
- The master does not move to a different node for the duration of the test

Partition and Heal

The goal is to establish the service operates correctly in face of node failures.

The test is performed in a three-node cluster using following steps:

1. Register candidate on each node
2. Wait for master activation
3. Isolate master node
4. Wait two minutes
5. Un-isolate (former) master node
6. Wait one minute

Success criteria:

- After step 3), master instance is brought down on isolated node
- During step 4) majority partition elects a new master
- Until 5) occurs, old master remains deactivated
- After 6) old master remains deactivated

Chasing the Leader

This test aims to establish the service operates correctly when faced with rapid application transitions without having a stabilized application.

This test is performed in a three-node setup using the following steps:

1. Register a candidate on each node
2. Wait for master activation
3. Newly activated master unregisters itself
4. Repeat 2

Success criteria:

- No failures occur for 5 minutes
- Transition speed is at least 100 movements per second

Controller Cluster Services Longevity Tests

1. Run No-Loss Rate test for 24 hours. No message loss, instability or memory leaks may occur.
2. Repeat Leader Stability test for 24 hours. No transaction failures, instability, leader movement or memory leaks may occur.
3. Repeat Explicit Leader Movement test for 24 hours. No transaction failures, instability, leader movement or memory leaks may occur.
4. Repeat RPC Provider Precedence test for 24 hours. No failures or memory leaks may occur.
5. Repeat RPC partition and Heal test for 24 hours. No failures or memory leaks may occur.
6. Repeat Chasing the Leader test for 24 hours. No memory leaks or failures may occur.

7. Repeat Partition and Heal test for 24 hours. No memory leaks or failures may occur.

NETCONF System Tests

Netconf is an MD-SAL application, which listens to config datastore changes, registers a singleton for every configured device, instantiated singleton is updating device connection data in operational datastore, maintaining a mount point and handling access to the mounted device.

Basic configuration and mount point access

No disruptions, ordinary netconf operation with restconf calls to different cluster members.

Test is executed in a three-node scenario, using the following steps:

1. Configure connection to test device on member-1.
2. Create, update and delete data on the device using calls to member-2.
3. Each state change confirmed by reading device data on member-3.
4. De-configure the device connection.

Success criteria:

- All reads confirm data operations are applied correctly.

Device owner killed

Killing current device owner leads to electing new owner. Operations are still applied.

The test is performed in a three-node cluster using following steps:

1. Configure connection to test device on member-1.
2. Create data on the device using a call to member-2.
3. Locate and kill the device owner member.
4. Wait for a new owner to get elected.
5. Update data on the device using a call to one of the surviving members.
6. Restart the killed member.
7. Update the data again using a call to the restarted member.

Success criteria:

- Each operation (including restart) is confirmed by reads on all members currently up.

Rolling restarts

Each member is restarted (start is waiting for cluster sync) in succession, this is to guarantee each Leader is affected.

The test is performed in a three-node cluster using following steps:

1. Configure connection to test device on member-1.
2. Kill member-1.

3. Create data on the device using a call to member-2.
4. Start member-1.
5. Kill member-2.
6. Update data on the device using a call to member-3.
7. Start member-2.
8. Kill member-3.
9. Delete data on the device using a call to member-1.
10. Start member-3.

Success criteria:

- After every operation, reads on both living members confirm it was applied.
- After every start, a read on the started node confirms it sees the device data from the previous operation.

Caveats

This sub-page describes ways the test implementation (or results) differs from the original specification and which information motivates the difference.

Jenkins job structure

- Information

At the start of test implementation, all the Controller 3node test cases were added into an existing Jenkins job.

During test development it was become clear, that adding all possible tests would make the job to run too long.

Dividing the job into several smaller ones is possible, but most likely the history would be lost, unless Linux Foundation admins figure out a way to create multiple job clones with history copied.

- Testing consequence

Even with number of test cases reduced (see below), the job duration is around three and half hours.

- How to fix

After Carbon SR2 release, the jobs can be split, as there will be enough time to generate new history till Carbon SR3.

Akka bugs

These are bugs which need either a fix in Akka codebase, or a workaround which would be too time-consuming to implement in ODL.

Both bugs manifest as UnreachableMember event (without intentional isolation).

Slow heartbeats

- Information

Akka sends periodic heartbeats in order to detect when the other member is being unresponsive.

The heartbeats are being serialized into the same TCP channel as ordinary data, which means if ODL is processing big amount of data, the heartbeats can spend a long time in TCP (or other) buffers before being processed. When this time exceeds a specific value (currently 6 seconds), the peer memeber is declared unreachable, generally leading to leader movement.

This affects BGP test results on 3node setup, as ODL is processing BGP data as quickly as possible, but the current BGP implementation does not handle rib owner movement gracefully (and leader movement is explicitly checked by the test, as the scenario dictates it should not happen). This does not affect other data broker tests, 1000 transactions per second do not generate critical throughput.

- Testing consequence

Three test cases are failing due to Bug 8318.

- How to fix

Possibly, a different akka configuration could be applied to separate akka cluster status messages into a different TCP stream than ordinary data stream.

Otherwise, a contribution to Akka project would be needed.

Reachability gossip

- Information

Akka uses a gossip protocol to advertize one member's reachability to other members. There is a logic which allows for faster detection of unreachable members, when a member can declare its peer unreachable if it got information from another peer which is considered more up-to-date.

Ocassionally, this logic results in undesired behavior. This is when the supposedly up-to-date peer has been isolated and now it is rejoining. Depending on timing, this can introduce additional leader movement, or a very brief moment when a member "forgets" RPC registrations from other member.

This is causing bugs 8420 and 8430.

- Testing consequence

This affects "partition and heal" scenarios in singleton testing. In functional tests, the failures are infrequent enough to consider the test mostly stable overall, but the corresponding longevity jobs are failing consistently.

The tests for "partition and heal" scenarios in RPC testing have been changed to tolerate wrong RPC results for 10 seconds to work around this Akka bug.

- How to fix

This does not seem fixable on ODL level, contribution to Akka project is needed.

Missing features

Cluster yang notifications

- Information

Yang notifications are not delivered to peer members. Bug 2139 is only fixed for data change notifications, not Yang notifications.

Bug 2140 tracks adding this missing functionality.

- Testing consequence

Notification suites are running on 1-node setup only.

- How to fix

After the funtionality is added, it will be straightforward to add 3node tests.

New features

Tell-based protocol

- Information

Tell-based protocol is an alternative to ask-based protocol from Boron. Which protocol to use is decided by a line in a configuration file (org.opendaylight.controller.cluster.datastore.cfg).

Some scenarios are expected to fail due to known limitations of ask-based protocol. More specifically, if a shard leader moves while a transaction is open in ask-based protocol, the transaction will fail (AskTimeoutException).

This affects only data broker tests, not RPC calls.

- Testing consequence

In principle, this doubles the number of configurations to be tested, but see below.

- How to fix

It is planned for tell-based protocol to become the default setting after Carbon SR2. After that, tests for ask-based protocol can be converted or removed.

Prefix-based shards

- Information

Tell-based shards are an alternative to module-based shards from Boron. Tell-based shards can be only created dynamically (as opposed to being read from a configuration file at startup). It is possible to use both types of shards, but data writes and reads use different API, so any Mdsal application needs to know which API to use.

The implementation of prefix-based shards is hardwired to tell-based protocol (even if ask-based protocol is configured as the default).

- Testing consequence

This doubles the number of configurations to be tested, for tests related to data droker (RPCs are unaffected).

- How to fix

ODL contains great many applications which use APIs for module-based shards. It is expected that multiple releases would still need both types of tests cases. Module-based shards will be deprecated and removed eventually.

Producer options

- Information

Data producers for module-based shards can produce either chained transactions or standalone transactions. Data producers for prefix-based shards can produce either non-isolated transactions (change notifications can combine several transactions together) or isolated transactions.

- Testing consequence

In principle, this results in multiple Robot test cases for the same documented scenario case, but see below.

- How to fix

All test cases will be needed in forseeable future. Instead, more negative test cases may need be added to verify different options lead to different behavior.

Initial leader placement

- Information

Some scenarios do not specify initial locations of relevant shard leaders. Test results can depend on it in presence of bugs.

This is mostly relevant to BGP test, which has three relevant members: Rib owner, default operation shard leader and topology operational shard leader.

- Testing consequence

Two test cases are tested. The two shard leaders are always together, rib owner is either co-located or not. This is done by suite moving shard leaders after detecting rib owner location.

- How to fix

Even more placements can be tested when job duration stops being the limiting factor.

Reduced BGP scaling

- Information

Rib owner maintains de-duplicated data structures. Other members get serialized copies and they do not de-duplicate.

Even single node strugless to fit into 6GB heap with tell-based protocol, see Bug 8649.

- Testing consequence

Scale from reported tests reduced from 1 million prefixes to 300 thousand prefixes.

- How to fix

Other members should be able to perform de-duplication, but developing that takes effort.

In the meantime, Linux Foundation could be convinced to allow for bigger VMs, currently limited by infrastructure available.

Increased timeouts

RequestTimeoutException

- Information

With tell-based protocol, restconf requests might stay open up to 120 seconds before returning an error. Even shard state reads using Jolokia can take long time if the shard actor is busy processing other messages.

- Testing consequence

This increases duration for tests which need to verify transaction errors do happen after sufficiently long isolation. Also, duration is increased if a test fails on a read which is otherwise quick.

- How to fix

This involves a trade-off between stability and responsiveness. As MD-SAL applications rarely tolerate transaction failures, users would prefer stability. That means relatively longer timeouts are there to stay, which means test case duration will stay high in negative (or failing positive) tests.

Client abort timeout

- Information

Client abort timeout is currently set to 15 minutes. The operational consequence is just an inability to start another data producer on a member isolated for that long. This test has too long duration compared to its usefulness.

- Testing consequence

This test case has never been implemented.

Instead a test with isolation shorter than 120 seconds is implemented, the test verifies the data producer continues its operation without RequestTimeoutException.

- How to fix

It is straighforward to add the missing test cases when job duration stops being a limiting factor.

No shard shutdown

- Common information.

There are multiple RPCs offering different "severity" of shard shutdown. For technical details see comments on change 58580.

If tests perform rigorous teardown, the shard replica should be re-activated, which is an operation not every RPC supports.

Listener stability suite

- Information

Current implementation of data listeners relies on a shard replica to be active on a member which is to receive the notification. Until that is imroved, Bug 8629 prevents this scenario from being tested as described.

- Testing consequence

The suite uses become-leader RPC instead. This has an added benefit of test case being able to pick which member is to become the new leader (adding one more test case when the old leader was not co-located with the listener).

Also, no teardown step is needed, the final cluster state is not missing any shard replica.

- How to fix

The original test can be implemented when listener implementation changes. But the test which uses become-leader might be better overall.

Clean leader shutdown suite

- Information

Some implementations of shutdown RPCs have a side effect of also shutting down shard state notifier. For details see Bug 8794.

The remove-shard-replica RPC does not have this downside, but it changes shard configuration, which was not intended by the original scenario definition.

- Testing consequence

Test cases for this scenario were switched to use remove-shard-replica.

- How to fix

There is an open debate on whether "shard shutdown" RPC with less operations (compared to remove-shard-replica) is something user wants and should be given access to.

If yes, tests can be switched to such an RPC, assuming the shard notifier issue is also fixed.

Hard reboots between test cases

- Information

Timing errors in Robot code lead to Robot being unable to restore original state without restarts.

During development, we started without any hard reboots, and that was finding bugs in teardown steps of scenarios. But test independence was more important at that time, so current tests are less sensitive to teardown failures.

- Testing consequence

Around 115 second per ODL reboot, this time is added to every test case running time. Together with increased timeouts, this motivates leaving out some test cases to allow faster change verification.

- How to fix

Ideally, we would want both jobs with hard resets and jobs without them. The jobs without resets can be added gradually after splitting the current single job.

Isolation mechanics

- Information

During development, it was found that freeze and kill mechanics affect the co-located java test driver without exposing any new bugs.

Turns out AAA functionality attempts to read from datastore, so isolated member returns http status code 401.

- Testing consequence

Only iptables filtering is used in order to reduce test job duration.

Isolated members are never queried directly. A leader member is considered isolated when other members elect a lew leader. A member is considered rejoined when it responds reporting itself as a follower.

- How to fix

It is straightforward to add test cases for kill and freeze where appropriate, but once again this can be done gradually when job duration is not a limiting factor.

Reduced number of combinations

- Information

Prefix-based shards always use tell-based protocol, so suites which test them with ask-based protocol configuration can be skipped.

Ask-based protocol is known to fail on AskTimeoutException on leader movement, so suites which produce transactions constantly can be skipped.

Most test cases are not sensitive to data producer options.

- Testing consequence

BGP tests and singleton tests use module-based shards only, both protocols. Other suites related to data broker are testing only tell-based protocol, both shard types. Netconf tests and RPC tests use module-based shards with ask-based protocol only. Only client isolaton suite tests different producer options.

- How to fix

More ests can be added gradually (see above).

Possibly, not every combination is worth the duration it takes, but that could be alleviated if Linux Foundation infrastructure grows in size significantly.

Reduced performance

- Information

In order to reduce test job duration, suites wait for minimal functionality (jolokia reporting shards are in sync) after restarting ODL. That means unrelated karaf features might still being installed whet test is in progress. This should not affect functional tests, but it can reduce performance observed.

The only suite observing strong enough performance inpact is chasing the leader.

- Testing consequence

Functional tests for chasing the leader suite tolerate frequencies higher than 50 un-registrations per second. Longevity suite still requires full 100 unregistrations per second.

- How to fix

Suite can wait for better symptom of ODL being ready, for example by requiring CPU usage to become less that a chosen threshold.

Missing logs

- Information

Robot VM has only 2GB of RAM and longevity jobs tend to produce large output.xml files.

Ocasionally, a job can create karaf.log files so large they fail to download, in extreme cases filling ODL VM disk and causing failures.

This affects mostly longevity jobs (and runs with verbose logging) if they pass.

- Testing consequence

Robot data stored is reduced to avoid this issue, sometimes leading to less details available. This issue is still not fully resolved, so ocassionally Robot log or karaf log is still missing if the job in question fails in an unexpected way.

- How to fix

It is possible for Robot test to put additional data into separate files. Unnecessarily verbose logs could be fixed where needed.

As this limitation only hurts in newly occuring bugs, it is not really possible to entirely avoid this.

Weekend outages

- Information

Linux foundation ifrastructure teem occasionally needs to perform changes which affect running jobs. To reduce this impact, such changes are usually done over weekend.

Cluster testing currently contains seve longevity jobs which block resources for 23 hours. As that is a significant portion of available resources, the longevity jobs are only run on weekend where the impact on frequency of other job is less critical.

- Testing consequence

Sometimes, the longevity jobs are affected by infrastructure team activities, leading to lost results or spurious failures. One such symptom is tracked as Bug 8959.

- How to fix

It might be possible to spread longevity jobs over work days. As distributing jobs manually is not a scalable option, a considerable work would be needed to create an automatic way.

Infrastructure changes are not very frequent, and having jobs run at the same predictable time is convenient from reporting point of view, so perhaps it is okay to keep the current setup.

List of test cases

Each test case has a shorter code, tables with results use that code. In result tables, the code is a link to this document, due to coala ReST requirements, the codes are (self-pointing) links also in this document.

Other links point to scenario definitions ao caveat items.

- DOMDataBroker: Producers make 1000 transactions per second, except BGP which works full speed.
  - Leader stability: BGP inject benchmark (thus module shards only), 300k prefixes, 1 Python peer. Progress tracked by counting prefixes in example-ipv4-topology.
    - Ask-based protocol:
      - Single member: *bgp-1n-300k-a*
    - Tell-based protocol:
      - Single member: *bgp-1n-300k-t*
      - Three members:
        - Leaders local: *bgp-3n-300k-ll-t*
        - Leaders remote: *bgp-3n-300k-lr-t*
        - Longevity: *bgp-3n-300k-t-long*
  - Clean leader shutdown, Tell-based protocol:
    - Module-based shards:
      - Shard leader local to producer: *ddb-cls-ms-ll-t*

        - Shard leader remote to producer: *ddb-cls-ms-lr-t*
    - Prefix-based shards:
        - Shard leader local to producer: *ddb-cls-ps-ll-t*
        - Shard leader remote to producer: *ddb-cls-ps-lr-t*
- Explicit leader movement, Tell-based protocol:
    - Module-based shards:
        - Local leader to remote: *ddb-elm-ms-lr-t*
        - Remote leader to other remote: *ddb-elm-ms-rr-t*
        - Remote leader to local: *ddb-elm-ms-rl-t*
        - Longevity (randomized direction): *ddb-elm-mc-t-long*
    - Prefix-based shards:
        - Local leader to remote: *ddb-elm-ps-lr-t*
        - Remote leader to other remote: *ddb-elm-ps-rr-t*
        - Remote leader to local: *ddb-elm-ps-rl-t*
- Leader isolation (network partition only), Tell-based protocol:
    - Module-based shards:
        - Heal within transaction timeout: *ddb-li-ms-st-t*
        - Heal after transaction timeout: *ddb-li-ms-dt-t*
    - Prefix-based shards:
        - Heal within transaction timeout: *ddb-li-ps-st-t*
        - Heal after transaction timeout: *ddb-li-ps-dt-t*
- Client isolation, Tell-based protocol:
    - Module-based shards:
        - Leader local:
            - Simple transactions: *ddb-ci-ms-ll-st-t*
            - Transaction chain: *ddb-ci-ms-ll-ct-t*
        - Leader remote:
            - Simple transactions: *ddb-ci-ms-lr-st-t*
            - Transaction chain: *ddb-ci-ms-lr-ct-t*
    - Prefix-based shards:
        - Leader local:
            - Isolated transactions: *ddb-ci-ps-ll-it-t*
            - Non-isolated transactions: *ddb-ci-ps-ll-nt-t*
        - Leader remote:
            - Isolated transactions: *ddb-ci-ps-lr-it-t*
            - Non-isolated transactions: *ddb-ci-ps-lr-nt-t*

    - Listener stablity, Tell-based protocol:
        - Module-based shards:
            - Local to remote: *ddb-ls-ms-lr-t*
            - Remote to remote: *ddb-ls-ms-rr-t*
            - Remote to local: *ddb-ls-ms-rl-t*
        - Prefix-based shards:
            - Local to remote: *ddb-ls-ps-lr-t*
            - Remote to remote: *ddb-ls-ps-rr-t*
            - Remote to local: *ddb-ls-ps-rl-t*
- DOMRpcBroker, ask-based protocol:
    - RPC Provider Precedence:
        - Functional: *drb-rpp-ms-a*
        - Longevity: *drb-rpp-ms-a-long*
    - RPC Provider Partition and Heal:
        - Functional: *drb-rph-ms-a*
        - Longevity: *drb-rph-ms-a-long*
    - Action Provider Precedence: *drb-app-ms-a*
    - Action Provider Partition and Heal: *drb-aph-ms-a*
- DOMNotificationBroker: Only for 1 member, ask-based protocol.
    - No-loss rate: Publisher-subscriber pairs, 5k nps per pair.
        - Functional (5 minute tests for 1, 4 and 12 pairs): *dnb-1n-60k-a*
        - Longevity (12 pairs): *dnb-1n-60k-a-long*
- Cluster Singleton:
    - Ask-based protocol:
        - Master Stability: *ss-ms-ms-a*
        - Partition and Heal:
            - Functional: *ss-ph-ms-a*
            - Longevity: *ss-ph-ms-a-long*
        - Chasing the Leader:
            - Functional: *ss-cl-ms-a*
            - Longevity: *ss-cl-ms-a-long*
    - Tell-based protocol:
        - Master Stability: *ss-ms-ms-t*
        - Partition and Heal: *ss-ph-ms-t*
        - Chasing the Leader: *ss-cl-ms-t*
- Netconf system tests (ask-based protocol, module-based shards):

- Basic access: *netconf-ba-ms-a*
- Owner killed: *netconf-ok-ms-a*
- Rolling restarts: *netconf-rr-ms-a*

Permanent draft, inaccessible: Sandbox test report

Test Case Summary

RelEng stability summary.

- tba: Recent failures to be analyzed yet: 0.
- test: Recent failures caused by wrong assumptions in test: 0.
- akka: Recent failures related to pure UnreachableMember: 4.
- tell: Recent failures not clearly caused by UnreachableMember: 6.
- few: Tests passing unless low frequency failure happens: 2 (1 without duplication). (Low frequency means UnreachableMemeber or similar, related to Akka where Controller code has not real control.)
- pass: Tests passing consistently: 41 (39 without duplication).
- Total: 53 (50 without duplication).
- Total minus akka: 49 (46 without duplication).
- Total minus akka passing always or mostly: 43 (40 without duplication).
- Acceptance rate: 43/49=87.75% (40/46=86.95% without duplication).

Table

S017 instead of 2017 means Sandbox run (includes changes not merged to stable/carbon yet).

Last fail is date of last failure not caused by infra (or by a typo in test or by netconf/bgp failing to initialize properly).

“S 17” or “2 17” in Last run means the documented run was superseded by a newer one, but not analyzed yet.

“no sr3” means this test was not run on Sandbox, SR2 result is reported instead. “few” status from SR2 is not inherited (such tests are marked as “pass”). “long ago” means the last real test failue happened somewhere before SR2 release (or never).

TODO: Copy formatting from sr2 page.

Table 3.2: Releng stability results (pre-SR2)

Scenario name	Type	Last fail	Last run	Bugs	Robot link
bgp-1n-1m-a	pass	no sr3	no sr3		no sr3
bgp-1n-300k-t	pass	no sr3	no sr3		no sr3
bgp-3n-300k-ll-t	akka	no sr3	no sr3	8318	no sr3
bgp-3n-300k-lr-t	akka	no sr3	no sr3	8318	no sr3
ddb-cls-ms-ll-t	pass	long ago	S017-08-24		no fail this week
ddb-cls-ms-lr-t	pass	long ago	S017-08-24		no fail this week
ddb-cls-ps-ll-t	pass	long ago	S017-08-24		no fail this week
ddb-cls-ps-lr-t	pass	long ago	S017-08-24		no fail this week

Continued on next page

Table 3.2 – continued from previous page

Scenario name	Type	Last fail	Last run	Bugs	Robot link
ddb-elm-ms-lr-t	pass	long ago	S017-08-24		no fail this week
ddb-elm-ms-rr-t	pass	long ago	S017-08-24		no fail this week
ddb-elm-ms-rl-t	pass	long ago	S017-08-24		no fail this week
ddb-elm-ps-lr-t	pass	long ago	S017-08-24		no fail this week
ddb-elm-ps-rr-t	pass	long ago	S017-08-24		no fail this week
ddb-elm-ps-rl-t	pass	long ago	S017-08-24		no fail this week
ddb-li-ms-st-t	pass	long ago	S017-08-24		no fail this week
ddb-li-ms-dt-t	pass	long ago	S017-08-24		no fail this week
ddb-li-ps-st-t	pass	long ago	S017-08-24		no fail this week
ddb-li-ps-dt-t	tell	S017-08-24	S017-08-24	8845	link
ddb-ci-ms-ll-ct-t	pass	long ago	S017-08-24		no fail this week
ddb-ci-ms-ll-st-t	pass	long ago	S017-08-24		no fail this week
ddb-ci-ms-lr-ct-t	pass	long ago	S017-08-24		no fail this week
ddb-ci-ms-lr-st-t	pass	long ago	S017-08-24		no fail this week
ddb-ci-ps-ll-ct-t	pass	long ago	S017-08-24		no fail this week
ddb-ci-ps-ll-st-t	pass	long ago	S017-08-24		no fail this week
ddb-ci-ps-lr-ct-t	pass	long ago	S017-08-24		no fail this week
ddb-ci-ps-lr-st-t	pass	long ago	S017-08-24		no fail this week
ddb-ls-ms-lr-t	pass	long ago	S017-08-24		no fail this week
ddb-ls-ms-rr-t	pass	long ago	S017-08-24		no fail this week
ddb-ls-ms-rl-t	pass	long ago	S017-08-24		no fail this week
ddb-ls-ps-lr-t	tell	S017-08-24	S017-08-24	8733	link
ddb-ls-ps-rr-t	tell	S017-08-24	S017-08-24	8733	link
ddb-ls-ps-rl-t	pass	long ago	S017-08-24		no fail this week
drb-rpp-ms-a	pass	long ago	S017-08-24		no fail this week
drb-rph-ms-a	pass	long ago	S017-08-24		no fail this week
drb-app-ms-a	pass	long ago	S017-08-24		no fail this week
drb-aph-ms-a	pass	long ago	S017-08-24		no fail this week
dnb-1n-60k-a	pass	no sr3	no sr3		no sr3
ss-ms-ms-a	pass	long ago	S017-08-24		no fail this week
ss-ph-ms-a	few	S017-08-24	S017-08-24	8420	link
ss-cl-ms-a	pass	long ago	S017-08-24		no fail this week
ss-ms-ms-t	pass	long ago	S017-08-24		no fail this week
ss-ph-ms-t	few	S017-08-24	S017-08-24	8420	link
ss-cl-ms-t	pass	long ago	S017-08-24		no fail this week
netconf-ba-ms-a	pass	no sr3	no sr3		no fail this week
netconf-ok-ms-a	tell	no sr3	no sr3	9027	no fail this week
netconf-rr-ms-a	tell	no sr3	no sr3	9027	no fail this week
bgp-3n-300k-t-long	akka	no sr3	no sr3	8318	no sr3
ddb-elm-mc-t-long	pass	no sr3	no sr3		no sr3
drb-rpp-ms-a-long	pass	no sr3	no sr3		no sr3
drb-rph-ms-a-long	pass	no sr3	no sr3	8430	no sr3
dnb-1n-60k-a-long	pass	no sr3	no sr3		no sr3
ss-ph-ms-a-long	akka	no sr3	no sr3	8420	no sr3
ss-cl-ms-a-long	tell	S017-08-23	S017-08-23	9054	link

For descriptions of test cases, see description page. Note that the link contains current description, the details might have been implemented differently at SR1 release.

Draft, outdated: Carbon release test report

Table

Table 3.3: Test results (pre-release)

Scenario name	Run date	Bug numbers	Result
bgp-1n-1m-a	2017-05-23		PASS
bgp-1n-1m-t	2017-05-23		PASS
bgp-3n-300k-ll-t	2017-05-23	8318	FAIL
bgp-3n-300k-lr-t	2017-05-23	8318	FAIL
ddb-cls-ms-ll-t	2017-05-23	8403	FAIL
ddb-cls-ms-lr-t	2017-05-23		PASS
ddb-cls-ps-ll-t	2017-05-23	8403	FAIL
ddb-cls-ps-lr-t	2017-05-23		PASS
ddb-elm-ms-lr-t	2017-05-23	8403	FAIL
ddb-elm-ms-rr-t	2017-05-23		PASS
ddb-elm-ms-rl-t	2017-05-23	8403	FAIL
ddb-elm-ps-lr-t	2017-05-23		PASS
ddb-elm-ps-rr-t	2017-05-23		PASS
ddb-elm-ps-rl-t	2017-05-23	8403	FAIL
ddb-li-ms-st-t	2017-05-23	8445	FAIL
ddb-li-ms-dt-t	2017-05-23	8494	FAIL
ddb-li-ps-st-t	2017-05-23	8371	FAIL
ddb-li-ps-dt-t	2017-05-23	8371	FAIL
ddb-ci-ms-ll-ct-t	2017-05-23	8494	FAIL
ddb-ci-ms-ll-st-t	2017-05-23	8494	FAIL
ddb-ci-ms-lr-ct-t	2017-05-23		PASS
ddb-ci-ms-lr-st-t	2017-05-23		PASS
ddb-ci-ps-ll-ct-t	2017-05-23	8494	FAIL
ddb-ci-ps-ll-st-t	2017-05-23	8494	FAIL
ddb-ci-ps-lr-ct-t	2017-05-23		PASS
ddb-ci-ps-lr-st-t	2017-05-23		PASS
ddb-ls-ms-ll-t	2017-05-23	8524	FAIL
ddb-ls-ms-lr-t	2017-05-23	8534	FAIL
ddb-ls-ps-ll-t	2017-05-23	8524	FAIL
ddb-ls-ps-lr-t	2017-05-23	8524	FAIL
drb-rpp-ms-a	2017-05-23		PASS
drb-rph-ms-a	2017-05-23		PASS
drb-app-ms-a	2017-05-23		PASS
drb-aph-ms-a	2017-05-23		PASS
dnb-1n-60k-a	2017-05-23		PASS
ss-ms-ms-a	2017-05-23		PASS
ss-ph-ms-a	2017-05-23		PASS
ss-cl-ms-a	2017-05-23		PASS
ss-ms-ms-t	2017-05-23		PASS
ss-ph-ms-t	2017-05-23		PASS
ss-cl-ms-t	2017-05-23		PASS
netconf-ba-ms-a	2017-05-23		PASS
netconf-ok-ms-a	2017-05-23		PASS

Continued on next page

Table 3.3 – continued from previous page

Scenario name	Run date	Bug numbers	Result
netconf-rr-ms-a	2017-05-23		PASS
bgp-3n-300k-t-long	2017-05-14	8443	FAIL
ddb-elm-mc-a-long	2017-05-14	8434	FAIL
drb-rpp-ms-a-long	2017-05-14		PASS
drb-rph-ms-a-long	2017-05-14		PASS
dnb-1n-60k-a-long	2017-05-14		PASS
ss-ph-ms-a-long	2017-05-14	8420	FAIL
ss-cl-ms-a-long	2017-05-14		PASS

For descriptions of test cases, see description page. Note that the link contains current description, the details might have been implemented differently at SR1 release.

Draft, outdated: Carbon SR1 test report

Test Case Summary

RelEng stability summary.

- tba: Recent failures to be analyzed yet: 0.
- test: Recent failures caused by wrong assumptions in test: 0.
- akka: Recent failures related to pure UnreachableMember: 5.
- tell: Recent failures not clearly caused by UnreachableMember: 9.
- few: Tests passing unless low frequency failure happens: 22 (21 without duplication). (Low frequency means UnreachableMemeber or "Message was not delivered, dead letters encountered", both are related to Akka where Controller code has not real control.)
- pass: Tests passing consistently: 17 (15 without duplication).
- Total: 53 (50 without duplication).
- Total minus akka: 48 (45 without duplication).
- Total minus akka passing always or mostly: 39 (36 without duplication).
- Acceptance rate: 39/48=81.25% (36/45=80.00% without duplication).

Table

S017 instead of 2017 means Sandbox run (includes changes not merged to stable/carbon yet).

Last fail is date of last failure not caused by infra (or by a typo in test or by netconf/bgp failing to initialize properly).

"S 17" or "2 17" in Last run means the documented run was superseded by a newer one, but not analyzed yet.

"long ago" means the last real test failue happened before around 2017-05-19, or never.

Table 3.4: Releng stability results (pre-SR1)

Scenario name	Type	Last fail	Last run	Bugs	Robot link
bgp-1n-1m-a	pass	long ago	2017-07-14		link

Continued on next page

Table 3.4 – continued from previous page

Scenario name	Type	Last fail	Last run	Bugs	Robot link
bgp-1n-300k-t	pass	long ago	2017-07-14		link
bgp-3n-300k-ll-t	akka	2017-07-14	2017-07-14	8318	link
bgp-3n-300k-lr-t	akka	2017-07-13	2017-07-14	8318	link
ddb-cls-ms-ll-t	few	2017-07-04	2017-07-15	8794	link
ddb-cls-ms-lr-t	few	2017-07-08	2017-07-15	8618	link
ddb-cls-ps-ll-t	few	2017-07-09	2017-07-15	8794	link
ddb-cls-ps-lr-t	pass	long ago	2017-07-15		link
ddb-elm-ms-lr-t	few	2017-06-13	2017-07-15	8618	link
ddb-elm-ms-rr-t	few	2017-06-10	2017-07-15	8618	link
ddb-elm-ms-rl-t	few	2017-06-27	2017-07-15	8749	link
ddb-elm-ps-lr-t	few	2017-06-11	2017-07-15	8664	link
ddb-elm-ps-rr-t	pass	long ago	2017-07-15		link
ddb-elm-ps-rl-t	few	2017-06-07	2017-07-15	8403	link
ddb-li-ms-st-t	tell	2017-07-15	2017-07-15	8792	link
ddb-li-ms-dt-t	tell	2017-07-15	2017-07-15	8619	link
ddb-li-ps-st-t	few	2017-06-08	2017-07-15	8371	link
ddb-li-ps-dt-t	tell	2017-07-15	2017-07-15	8845	link
ddb-ci-ms-ll-ct-t	few	2017-06-07	2017-07-15	8494	link
ddb-ci-ms-ll-st-t	tell	2017-07-15	2017-07-15	8494	link
ddb-ci-ms-lr-ct-t	few	2017-06-08	2017-07-15	8636	link
ddb-ci-ms-lr-st-t	tell	2017-07-15	2017-07-15	8494	link
ddb-ci-ps-ll-ct-t	few	2017-06-28	2017-07-15	8494	link
ddb-ci-ps-ll-st-t	few	2017-06-28	2017-07-15	8494	link
ddb-ci-ps-lr-ct-t	few	2017-06-28	2017-07-15	8494	link
ddb-ci-ps-lr-st-t	few	2017-06-28	2017-07-15	8494	link
ddb-ls-ms-lr-t	tell	2017-07-15	2017-07-15	8792	link
ddb-ls-ms-rr-t	tell	2017-07-14	2017-07-15	8792	link
ddb-ls-ms-rl-t	tell	2017-07-12	2017-07-15	8792	link
ddb-ls-ps-lr-t	pass	long ago	2017-07-15		link
ddb-ls-ps-rr-t	few	2017-06-26	2017-07-15	8733	link
ddb-ls-ps-rl-t	pass	long ago	2017-07-15		link
drb-rpp-ms-a	pass	long ago	2017-07-15		link
drb-rph-ms-a	few	2017-06-28	2017-07-15	8430	link
drb-app-ms-a	pass	long ago	2017-07-15		link
drb-aph-ms-a	few	2017-07-02	2017-07-15	8430	link
dnb-1n-60k-a	pass	long ago	2017-07-15		link
ss-ms-ms-a	pass	long ago	2017-07-15		link
ss-ph-ms-a	few	2017-06-29	2017-07-15	8420	link
ss-cl-ms-a	pass	long ago	2017-07-15		link
ss-ms-ms-t	pass	long ago	2017-07-15		link
ss-ph-ms-t	few	2017-07-15	2017-07-15	8420	link
ss-cl-ms-t	pass	long ago	2017-07-15		link
netconf-ba-ms-a	pass	long ago	2017-07-14		link
netconf-ok-ms-a	few	2017-06-18	2017-07-14	8596	link
netconf-rr-ms-a	pass	long ago	2017-07-14		link
bgp-3n-300k-t-long	akka	2017-07-08	2017-07-08	8318	link
ddb-elm-mc-t-long	tell	2017-07-08	2017-07-08	8618	link
drb-rpp-ms-a-long	few	2017-05-07	2017-07-08	8430	link
drb-rph-ms-a-long	akka	2017-07-08	2017-07-08	8430	link

Continued on next page

Table 3.4 – continued from previous page

Scenario name	Type	Last fail	Last run	Bugs	Robot link
dnb-1n-60k-a-long	pass	long ago	2017-07-08		link
ss-ph-ms-a-long	akka	2017-07-08	2017-07-08	8420	link
ss-cl-ms-a-long	pass	long ago	2017-07-08		link

For descriptions of test cases, see description page. Note that the link contains current description, the details might have been implemented differently at SR1 release.

Carbon SR2 test report

Test Case Summary

RelEng stability summary.

- tba: Recent failures to be analyzed yet: 0.
- test: Recent failures caused by wrong assumptions in test: 0.
- akka: Recent failures related to pure UnreachableMember: 4.
- tell: Recent failures not clearly caused by UnreachableMember: 4.
- few: Tests passing unless low frequency failure happens: 7 (6 without duplication). (Low frequency means infra issues or UnreachableMemeber, related to Akka where Controller code has not real control.)
- pass: Tests passing consistently: 38 (36 without duplication).
- Total: 53 (50 without duplication).
- Total minus akka: 49 (46 without duplication).
- Total minus akka, passing always or mostly: 45 (42 without duplication).
- Acceptance rate: 45/49=91.83% (42/46=91.30% without duplication).

Table

S017 instead of 2017 means Sandbox run (includes changes not merged to stable/carbon yet).

Last fail is date of last failure not caused by infra (or by a typo in test or by netconf/bgp failing to initialize properly).

“S 17” or “2 17” in Last run means the documented run was superseded by a newer one, but not analyzed yet.

“few” status from SR1 is not inherited (such tests are marked as “pass”). “long ago” means the last real test failue happened somewhere around SR1 release (or before that, or never).

If status is a link, it points to the latest relevant robot failure, or a history to see the stability. In case of failure, Bugs field gives the reason of that failure.

Table 3.5: Releng stability results (post SR1, pre SR2)

Test case	Last fail	Last run	Bugs	Status
bgp-1n-300k-a	long ago	2017-09-18		PASS
bgp-1n-300k-t	long ago	2017-09-18		PASS
bgp-3n-300k-ll-t	2017-09-16	2017-09-18	8318	AKKA
bgp-3n-300k-lr-t	2017-09-16	2017-09-18	8318	AKKA

Continued on next page

Table 3.5 – continued from previous page

Test case	Last fail	Last run	Bugs	Status
ddb-cls-ms-ll-t	2017-08-24	2017-09-18		PASS
ddb-cls-ms-lr-t	long ago	2017-09-18		PASS
ddb-cls-ps-ll-t	long ago	2017-09-18		PASS
ddb-cls-ps-lr-t	long ago	2017-09-18		PASS
ddb-elm-ms-lr-t	long ago	2017-09-18		PASS
ddb-elm-ms-rr-t	long ago	2017-09-18		PASS
ddb-elm-ms-rl-t	long ago	2017-09-18		PASS
ddb-elm-ps-lr-t	long ago	2017-09-18		PASS
ddb-elm-ps-rr-t	long ago	2017-09-18		PASS
ddb-elm-ps-rl-t	long ago	2017-09-18		PASS
ddb-li-ms-st-t	2017-08-18	2017-09-18		PASS
ddb-li-ms-dt-t	2017-08-21	2017-09-18		PASS
ddb-li-ps-st-t	2017-09-01	2017-09-18		PASS
ddb-li-ps-dt-t	2017-09-18	2017-09-18	8845	TELL
ddb-ci-ms-ll-ct-t	long ago	2017-09-18		PASS
ddb-ci-ms-ll-st-t	long ago	2017-09-18		PASS
ddb-ci-ms-lr-ct-t	long ago	2017-09-18		PASS
ddb-ci-ms-lr-st-t	long ago	2017-09-18		PASS
ddb-ci-ps-ll-it-t	long ago	2017-09-18		PASS
ddb-ci-ps-ll-nt-t	long ago	2017-09-18		PASS
ddb-ci-ps-lr-it-t	long ago	2017-09-18		PASS
ddb-ci-ps-lr-nt-t	long ago	2017-09-18		PASS
ddb-ls-ms-lr-t	long ago	2017-09-18		PASS
ddb-ls-ms-rr-t	long ago	2017-09-18		PASS
ddb-ls-ms-rl-t	long ago	2017-09-18		PASS
ddb-ls-ps-lr-t	2017-09-18	2017-09-18	8733	TELL
ddb-ls-ps-rr-t	2017-09-18	2017-09-18	8733	TELL
ddb-ls-ps-rl-t	2017-09-18	2017-09-18	8733	FEW
drb-rpp-ms-a	long ago	2017-09-18		PASS
drb-rph-ms-a	long ago	2017-09-18		PASS
drb-app-ms-a	long ago	2017-09-18		PASS
drb-aph-ms-a	long ago	2017-09-18		PASS
dnb-1n-60k-a	long ago	2017-09-18		PASS
ss-ms-ms-a	long ago	2017-09-18		PASS
ss-ph-ms-a	2017-09-01	2017-09-18	8420	FEW
ss-cl-ms-a	long ago	2017-09-18		PASS
ss-ms-ms-t	long ago	2017-09-18		PASS
ss-ph-ms-t	2017-09-17	2017-09-18	9177	FEW
ss-cl-ms-t	long ago	2017-09-18		PASS
netconf-ba-ms-a	long ago	2017-09-18		PASS
netconf-ok-ms-a	long ago	2017-09-18		PASS
netconf-rr-ms-a	2017-09-06	2017-09-18	9006	TELL
bgp-3n-300k-t-long	2017-09-16	2017-09-16	8318	AKKA
ddb-elm-mc-t-long	2017-08-06	2017-09-16		FEW
drb-rpp-ms-a-long	long ago	2017-09-16		FEW
drb-rph-ms-a-long	2017-08-12	2017-09-16		PASS
dnb-1n-60k-a-long	long ago	2017-09-16		FEW
ss-ph-ms-a-long	2017-09-16	2017-09-16	8420	AKKA
ss-cl-ms-a-long	2017-08-06	2017-09-16		PASS

Note that release, sr1 and sandbox pages contain data from before test implementation and documentation structure were finalized, so there may be inconsistencies.

TODO: Re-test Carbon release and SR1 images (with retrofitted tests where needed) so users can see authoritative test results.

External resources:

- System Test Guide.
- Infrastructure Guide.
- Running System Tests.
- Test Code Guidelines.
- Test Case Expectations.
- Boron Test Requirements.
- Robot API docs.

3.3 Documentation Guide

This guide provides details on how to contribute to the OpenDaylight documentation. OpenDaylight currently uses reStructuredText for documentation and Sphinx to build it. These documentation tools are widely used in open source communities to produce both HTML and PDF documentation and can be easily versioned alongside the code. reStructuredText also offers similar syntax to Markdown, which is familiar to many developers.

Contents

3.3.1 Style Guide

This section serves two purposes:

1. A guide for those writing documentation.

2. A guide for those reviewing documentation.

Note: When reviewing content, assuming that the content is usable, the documentation team is biased toward merging the content rather than blocking it due to relatively minor editorial issues.

Formatting Preferences

In general, when reviewing content, the documentation team ensures that it is comprehensible but tries not to be overly pedantic. Along those lines, while it is preferred that the following formatting preferences are followed, they are generally not an exclusive reason to give a "-1" reply to a patch in Gerrit:

- No trailing whitespace
- Line wrapping at something reasonable, that is, 72–100 characters

Key terms

- Functionality: something useful a project provides abstractly
- Feature: a Karaf feature that somebody could install
- Project: a project within OpenDaylight; projects ship features to provide functionality
- OpenDaylight: this refers to the software we release; use this in place of OpenDaylight controller, the OpenDaylight controller, not ODL, not ODC
  - Since there is a controller project within OpenDaylight, using other terms is hard.

Common writing style mistakes

- In per-project user documentation, you should never say *git clone*, but should assume people have downloaded and installed the controller per the getting started guide and start with `feature:install <something>`
- Avoid statements which are true about part of OpenDaylight, but not generally true.
  - For example: "OpenDaylight is a NETCONF controller." It is, but that is not all it is.
- In general, developer documentation should target external developers to your project so should talk about what APIs you have and how they could use them. It *should not* document how to contribute to your project.

Grammar Preferences

- Avoid contractions: Use "cannot" instead of "can't", "it is" instead of "it's", and so on.

Word Choice

Note: The following word choice guidelines apply when using these terms in text. If these terms are used as part of a URL, class name, or any instance where modifying the case would create issues, use the exact capitalization and spacing associated with the URL or class name.

- ACL: not Acl or acl

- API: not api
- ARP: not Arp or arp
- datastore: not data store, Data Store, or DataStore (unless it is a class/object name)
- IPsec, not IPSEC or ipsec
- IPv4 or IPv6: not Ipv4, Ipv6, ipv4, ipv6, IPV4, or IPV6
- Karaf: not karaf
- Linux: not LINUX or linux
- NETCONF: not Netconf or netconf
- Neutron: not neutron
- OSGi: not osgi or OSGI
- Open vSwitch: not OpenvSwitch, OpenVSwitch, or Open V Switch.
- OpenDaylight: not Opendaylight, Open Daylight, or OpenDayLight.

Note: Also, avoid Opendaylight abbreviations like ODL and ODC.

- OpenFlow: not Openflow, Open Flow, or openflow.
- OpenStack: not Open Stack or Openstack
- QoS: not Qos, QOS, or qos
- RESTCONF: not Restconf or restconf
- RPC not Rpc or rpc
- URL not Url or url
- VM: not Vm or vm
- YANG: not Yang or yang

3.3.2 reStructuredText-based Documentation

When using reStructuredText, follow the Python documentation style guidelines. See: https://docs.python.org/devguide/documenting.html

One of the best references for reStrucutedText syntax is the Sphinx Primer on reStructuredText.

To build and review the reStructuredText documentation locally, you must have the following packages installed locally:

- python
- python-tox

Note: Both packages should be available in most distribution package managers.

Then simply run `tox` and open the HTML produced by using your favorite web browser as follows:

```
git clone https://git.opendaylight.org/gerrit/docs
cd docs
git submodule update --init
tox
firefox docs/_build/html/index.html
```

Directory Structure

The directory structure for the reStructuredText documentation is rooted in the `docs` directory inside the `docs git` repository.

Note: There are guides hosted directly in the `docs git` repository and there are guides hosted in remote `git` repositories. Documentation hosted in remote `git` repositories are generally provided for project-specific information.

For example, here is the directory layout on June, 28th 2016:

```
$ tree -L 2
.
- Makefile
- conf.py
- documentation.rst
- getting-started-guide
|   - api.rst
|   - concepts_and_tools.rst
|   - experimental_features.rst
|   - index.rst
|   - installing_opendaylight.rst
|   - introduction.rst
|   - karaf_features.rst
|   - other_features.rst
|   - overview.rst
|   - who_should_use.rst
- index.rst
- make.bat
- opendaylight-with-openstack
|   - images
|   - index.rst
|   - openstack-with-gbp.rst
|   - openstack-with-ovsdb.rst
|   - openstack-with-vtn.rst
- submodules
    - releng
        - builder
```

The `getting-started-guide` and `opendaylight-with-openstack` directories correspond to two guides hosted in the `docs` repository, while the `submodules/releng/builder` directory houses documentation for the RelEng/Builder project.

Each guide includes an `index.rst` file, which uses a `toctree` directive that includes the other files associated with the guide. For example:

```
.. toctree::
   :maxdepth: 1

   getting-started-guide/index
```

```
   opendaylight-with-openstack/index
   submodules/releng/builder/docs/index
```

This example creates a table of contents on that page where each heading of the table of contents is the root of the files that are included.

Note: When including `.rst` files using the `toctree` directive, omit the `.rst` file extension at the end of the file name.

Adding a submodule

If you want to import a project underneath the documentation project so that the docs can be kept in the separate repo, you can do it by using the `git submodule add` command as follows:

```
git submodule add -b master ../integration/packaging docs/submodules/integration/
↪packaging
git commit -s
```

Note: Most projects will not want to use `-b master`, but instead use the branch `.`, which tracks whatever branch of the documentation project you happen to be on.

Unfortunately, `-b .` does not work, so you have to manually edit the `.gitmodules` file to add `branch = .` and then commit it. For example:

```
<edit the .gitmodules file>
git add .gitmodules
git commit --amend
```

When you're done you should have a git commit something like:

```
$ git show
commit 7943ce2cb41cd9d36ce93ee9003510ce3edd7fa9
Author: Daniel Farrell <dfarrell@redhat.com>
Date:   Fri Dec 23 14:45:44 2016 -0500

    Add Int/Pack to git submodules for RTD generation

    Change-Id: I64cd36ca044b8303cb7fc465b2d91470819a9fe6
    Signed-off-by: Daniel Farrell <dfarrell@redhat.com>

diff --git a/.gitmodules b/.gitmodules
index 91201bf6..b56e11c8 100644
--- a/.gitmodules
+++ b/.gitmodules
@@ -38,3 +38,7 @@
        path = docs/submodules/ovsdb
        url = ../ovsdb
        branch = .
+[submodule "docs/submodules/integration/packaging"]
+       path = docs/submodules/integration/packaging
+       url = ../integration/packaging
+       branch = master
```

```
diff --git a/docs/submodules/integration/packaging b/docs/submodules/integration/
↪packaging
new file mode 160000
index 00000000..fd5a8185
--- /dev/null
+++ b/docs/submodules/integration/packaging
@@ -0,0 +1 @@
+Subproject commit fd5a81853e71d45945471d0f91bbdac1a1444386
```

As usual, you can push it to Gerrit with `git review`.

Important: It is critical that the Gerrit patch be merged before the git commit hash of the submodule changes. Otherwise, Gerrit is not able to automatically keep it up-to-date for you.

Documentation Layout and Style

As mentioned previously, OpenDaylight aims to follow the Python documentation style guidelines, which defines a few types of sections:

```
# with overline, for parts
* with overline, for chapters
=, for sections
-, for subsections
^, for subsubsections
", for paragraphs
```

OpenDaylight documentation is organized around the following structure based on that recommendation:

```
docs/index.rst                  -> entry point
docs/____-guide/index.rst       -> part
docs/____-guide/<chapter>.rst   -> chapter
```

In the ____-guide/index.rst we use the # with overline at the very top of the file to determine that it is a part and then within each chapter file we start the document with a section using * with overline to denote that it is the chapter heading and then everything in the rest of the chapter should use:

```
=, for sections
-, for subsections
^, for subsubsections
", for paragraphs
```

Referencing Sections

This section provides a quick primer for creating references in OpenDaylight documentation. For more information, refer to Cross-referencing documents.

Within a single document, you can reference another section simply by:

```
This is a reference to `The title of a section`_
```

Assuming that somewhere else in the same file, there a is a section title something like:

```
The title of a section
^^^^^^^^^^^^^^^^^^^^^^
```

It is typically better to use `:ref:` syntax and labels to provide links as they work across files and are resilient to sections being renamed. First, you need to create a label something like:

```
.. _a-label:

The title of a section
^^^^^^^^^^^^^^^^^^^^^^
```

Note: The underscore (_) before the label is required.

Then you can reference the section anywhere by simply doing:

```
This is a reference to :ref:`a-label`
```

or:

```
This is a reference to :ref:`a section I really liked <a-label>`
```

Note: When using `:ref:`-style links, you don't need a trailing underscore (_).

Because the labels have to be unique, a best practice is to prefix the labels with the project name to help share the label space; for example, use `sfc-user-guide` instead of just `user-guide`.

Troubleshooting

Nested formatting does not work

As stated in the reStructuredText guide, inline markup for bold, italic, and fixed-width font cannot be nested. Furthermore, inline markup cannot be mixed with hyperlinks, so you cannot have a link with bold text.

This is tracked in a Docutils FAQ question, but there is no clear current plan to fix this.

Make sure you have cloned submodules

If you see an error like this:

```
./build-integration-robot-libdoc.sh: line 6: cd: submodules/integration/test/csit/
↪libraries: No such file or directory
Resource file '*.robot' does not exist.
```

It means that you have not pulled down the git submodule for the integration/test project. The fastest way to do that is:

```
git submodule update --init
```

In some cases, you might wind up with submodules which are somehow out-of-sync. In that case, the easiest way to fix them is to delete the submodules directory and then re-clone the submodules:

```
rm -rf docs/submodules/
git submodule update --init
```

Warning: These steps delete any local changes or information you made in the submodules, which would only occur if you manually edited files in that directory.

Clear your tox directory and try again

Sometimes, tox will not detect when your `requirements.txt` file has changed and so will try to run things without the correct dependencies. This issue usually manifests as `No module named X` errors or an `ExtensionError` and can be fixed by deleting the `.tox` directory and building again:

```
rm -rf .tox
tox
```

Builds on Read the Docs

Read the Docs builds do not automatically clear the file structure between builds and clones. The result is that you may have to clean up the state of old runs of the build script.

As an example, refer to the following patch: https://git.opendaylight.org/gerrit/41679

This patch fixed the issue that caused builds to fail because they were taking too long removing directories associated with generated javadoc files that were present from previous runs.

Errors from Coala

As part of running `tox`, two environments run: `coala` which does a variety of reStructuredText (and other) linting, and `docs`, which runs Sphinx to build HTML and PDF documentation. You can run them independently by doing `tox -ecoala` or `tox -edocs`.

The `coala` linter for reStructuredText is not always the most helpful in explaining why it failed. So, here are some common ones. There should also be Jenkins Failure Cause Management rules that will highlight these for you.

Git Commit Message Errors

Coala checks that git commit messages adhere to the following rules:

- Shortlog (1st line of commit message) is less than 50 characters
- Shortlog (1st line of commit message) is in the imperative mood. For example, "Add foo unit test" is good, but "Adding foo unit test is bad""
- Body (all lines but 1st line of commit message) are less than 72 characters. Some exceptions seem to exist, such as for long URLs.

Some examples of those being logged are:

:: Project wide: | | [NORMAL] GitCommitBear: | | Shortlog of HEAD commit isn't in imperative mood! Bad words are 'Adding'

:: Project wide: | | [NORMAL] GitCommitBear: | | Body of HEAD commit contains too long lines. Commit body lines should not exceed 72 characters.

Error in "code-block" directive

If you see an error like this:

:: docs/gerrit.rst | 89| ···..·code-block::·bash | | [MAJOR] RSTcheckBear: | | (ERROR/3) Error in "code-block" directive:

It means that the relevant code-block is not valid for the language specified, in this case `bash`.

Note: If you do not specify a language, the default language is Python. If you want the code-block to not be an any particular language, instead use the `::` directive. For example:

::

    :: This is a code block that will not be parsed in any particluar langauge

3.3.3 Project Documentation Requirements

Submitting Documentation Outlines (M2)

1. Determine the features your project will have and which ones will be ''user-facing''.
   - In general, a feature is user-facing if it creates functionality that a user would directly interact with.
   - For example, `odl-openflowplugin-flow-services-ui` is likely user-facing since it installs user-facing OpenFlow features, while `odl-openflowplugin-flow-services` is not because it provides only developer-facing features.
2. Determine pieces of documentation that you need to provide based on the features your project will have and which ones will be user-facing.
   - The kinds of required documentation can be found below in the *Requirements for projects* section.

   Note: You might need to create multiple documents for the same kind of documentation. For example, the controller project will likely want to have a developer section for the config subsystem as well as for the MD-SAL.

   ---

3. Clone the docs repo: `git clone https://git.opendaylight.org/gerrit/docs`
4. For each piece of documentation find the corresponding template in the docs repo.
   - For user documentation: `docs.git/docs/templates/template-user-guide.rst`
   - For developer documentation: `ddocs/templates/template-developer-guide.rst`
   - For installation documentation (if any): `docs/templates/template-install-guide.rst`

   Note: You can find the rendered templates here:

<Feature> User Guide

Refer to this template to identify the required sections and information that you should provide for a User Guide. The user guide should contain configuration, administration, management, using, and troubleshooting sections for the feature.

Overview

Provide an overview of the feature and the use case. Also include the audience who will use the feature. For example, audience can be the network administrator, cloud administrator, network engineer, system administrators, and so on.

<Feature> Architecture

Provide information about feature components and how they work together. Also include information about how the feature integrates with OpenDaylight. An architecture diagram could help.

Note: Please *do not* include detailed internals that somebody using the feature wouldn't care about. For example, the fact that there are four layers of APIs between a user command and a message being sent to a device is probably not useful to know unless they have some way to influence how those layers work and a reason to do so.

Configuring <feature>

Describe how to configure the feature or the project after installation. Configuration information could include day-one activities for a project such as configuring users, configuring clients/servers and so on.

Administering or Managing <feature>

Include related command reference or operations that you could perform using the feature. For example viewing network statistics, monitoring the network, generating reports, and so on.

For example:

To configure L2switch components perform the following steps.

(a) Step 1:

(b) Step 2:

(c) Step 3:

Tutorials

optional

If there is only one tutorial, you skip the "Tutorials" section and instead just lead with the single tutorial's name. If you do, also increase the header level by one, i.e., replace the carets (^^^) with dashes (- - -) and the dashes with equals signs (===).

<Tutorial Name>

Ensure that the title starts with a gerund. For example using, monitoring, creating, and so on.

Overview

An overview of the use case.

Prerequisites

Provide any prerequisite information, assumed knowledge, or environment required to execute the use case.

Target Environment

Include any topology requirement for the use case. Ideally, provide visual (abstract) layout of network diagrams and any other useful visual aides.

Instructions

Use case could be a set of configuration procedures. Including screenshots to help demonstrate what is happening is especially useful. Ensure that you specify them separately. For example:

Setting up the VM

To set up a VM perform the following steps.

(a) Step 1

(b) Step 2

(c) Step 3

Installing the feature

To install the feature perform the following steps.

(a) Step 1

(b) Step 2

(c) Step 3

Configuring the environment

To configure the system perform the following steps.

(a) Step 1

(b) Step 2

(c) Step 3

<Feature> Developer Guide

Overview

Provide an overview of the feature, what it logical functionality it provides and why you might use it as a developer. To be clear the target audience for this guide is a developer who will be *using* the feature to build something separate, but *not* somebody who will be developing code for this feature itself.

Note: More so than with user guides, the guide may cover more than one feature. If that is the case, be sure to list all of the features this covers.

<Feature> Architecture

Provide information about feature components and how they work together. Also include information about how the feature integrates with OpenDaylight. An architecture diagram could help. This may be the same as the diagram used in the user guide, but it should likely be less abstract and provide more information that would be applicable to a developer.

Key APIs and Interfaces

Document the key things a user would want to use. For some features, there will only be one logical grouping of APIs. For others there may be more than one grouping.

Assuming the API is MD-SAL- and YANG-based, the APIs will be available both via RESTCONF and via Java APIs. Giving a few examples using each is likely a good idea.

API Group 1

Provide a description of what the API does and some examples of how to use it.

API Group 2

Provide a description of what the API does and some examples of how to use it.

API Reference Documentation

Provide links to JavaDoc, REST API documentation, etc.

<Feature> Installation Guide

Note: *Only* use this template if installation is more complicated than simply installing a feature in the Karaf distribution. Otherwise simply provide the names of all user-facing features in your M3 readout.

This is a template for installing a feature or a project developed in the ODL project. The *feature* could be interfaces, protocol plug-ins, or applications.

Overview

Add overview of the feature. Include Architecture diagram and the positioning of this feature in overall controller architecture. Highlighting the feature in a different color within the overall architecture must help. Include information to describe if the project is within ODL installation package or to be installed separately.

Pre Requisites for Installing <Feature>

- Hardware Requirements
- Software Requirements

Preparing for Installation

Include any pre configuration, database, or other software downloads required to install <feature>.

Installing <Feature>

Include if you have separate procedures for Windows and Linux

Verifying your Installation

Describe how to verify the installation.

Troubleshooting

optional

Text goes here.

Post Installation Configuration

Post Installation Configuration section must include some basic (must-do) procedures if any, to get started.

Mandatory instructions to get started with the product.

- Logging in
- Getting Started
- Integration points with controller

Upgrading From a Previous Release

Text goes here.

Uninstalling <Feature>

Text goes here.

---

5. Copy the template into the appropriate directory for your project.
   - For user documentation: `docs.git/docs/user-guide/${feature-name}-user-guide.rst`
   - For developer documentation: `docs.git/docs/developer-guide/${feature-name}-developer-guide.rst`
   - For installation documentation (if any): `docs.git/docs/getting-started-guide/project-specific-guides/${project-name}.rst`

Note: These naming conventions are not set in stone, but are used to maintain a consistent document taxonomy. If these conventions are not appropriate or do not make sense for a document in development, use the convention that you think is more appropriate and the documentation team will review it and give feedback on the gerrit patch.

---

6. Edit the template to fill in the outline of what you will provide using the suggestions in the template. If you feel like a section is not needed, feel free to omit it.
7. Link the template into the appropriate core `.rst` file.
   - For user documentation: `docs.git/docs/user-guide/index.rst`
   - For developer documentation: `docs.git/docs/developer-guide/index.rst`
   - For installation documentation (if any): `docs.git/docs/getting-started-guide/project-specific-guides/index.rst`
   - In each file, it should be pretty clear what line you need to add. In general if you have an `.rst` file `project-name.rst`, you include it by adding a new line `project-name` without the `.rst` at the end.
8. Make sure the documentation project still builds.
   - Run `tox` from the root of the cloned docs repo.
     - After that, you should be able to find the HTML version of the docs at `docs.git/docs/_build/html/index.html`.
     - See *reStructuredText-based Documentation* for more details about building the docs.
   - The *reStructuredText Troubleshooting* section provides common errors and solutions.
   - If you still have problems e-mail the documentation group at documentation@lists.opendaylight.org
9. Commit and submit the patch.

   (a) Commit using:

```
git add --all && git commit -sm "Documentation outline for ${project-
↪shortname}"
```

   (b) Submit using:

```
git review
```

   See the Git-review Workflow page if you don't have git-review installed.

10. Wait for the patch to be merged or to get feedback
    - If you get feedback, make the requested changes and resubmit the patch.
    - When you resubmit the patch, it is helpful if you also post a "+0" reply to the patch in Gerrit, stating what patch set you just submitted and what you fixed in the patch set.

Expected Output From Documentation Project

The expected output is (at least) 3 PDFs and equivalent web-based documentation:

- User/Operator Guide
- Developer Guide
- Installation Guide

These guides will consist of "front matter" produced by the documentation group and the per-project/per-feature documentation provided by the projects.

Note: This requirement is intended for the person responsible for the documentation and should not be interpreted as preventing people not normally in the documentation group from helping with front matter nor preventing people from the documentation group from helping with per-project/per-feature documentation.

Project Documentation Requirements

Content Types

These are the expected kinds of documentation and target audiences for each kind.

- **User/Operator:** for people looking to use the feature without writing code
  - Should include an overview of the project/feature
  - Should include description of availble configuration options and what they do
- **Developer:** for people looking to use the feature in code without modifying it
  - Should include API documentation, such as, enunciate for REST, Javadoc for Java, ??? for REST-CONF/models
- **Contributor:** for people looking to extend or modify the feature's source code

  ---

  Note: You can find this information on the wiki.

- **Installation:** for people looking for instructions to install the feature after they have downloaded the ODL release

  ---

  Note: The audience for this content is the same as User/Operator docs

  ---

  - For most projects, this will be just a list of top-level features and options
    * As an example, l2switch-switch as the top-level feature with the -rest and -ui options
    * Features should also note if the options should be checkboxes (that is, they can each be turned on/off independently) or a drop down (that is, at most one can be selected)

        * What other top-level features in the release are incompatible with each feature
        * This will likely be presented as a table in the documentation and the data will likely also be consumed by automated installers/configurators/downloaders
    - For some projects, there is extra installation instructions (for external components) and/or configuration
        * In that case, there will be a (sub)section in the documentation describing this process.
- **HowTo/Tutorial:** walk throughs and examples that are not general-purpose documentation
    - Generally, these should be done as a (sub)section of either user/operator or developer documentation.
    - If they are especially long or complex, they may belong on their own
- **Release Notes:**
    - Release notes are required as part of each project's release review. They must also be translated into reStructuredText for inclusion in the formal documentation.

Requirements for projects

- Projects must provide reStructuredText documentation including:
    - Developer documentation for every feature
        * Most projects will want to logically nest the documentation for individual features under a single project-wide chapter or section
        * The feature documentation can be provided as a single `.rst` file or multiple `.rst` files if the features fall into different groups
        * Feature documentation should start with appromimately 300 word overview of the project and include references to any automatically-generated API documentation as well as more general developer information (see *Content Types*).
    - User/Operator documentation for every every user-facing feature (if any)
        * This documentation should be per-feature, not per-project. Users should not have to know which project a feature came from.
        * Intimately related features can be documented together. For example, l2switch-switch, l2switch-switch-rest, and l2switch-switch-ui, can be documented as one noting the differences.
        * This documentation can be provided as a single `.rst` file or multiple `.rst` files if the features fall into different groups
    - Installation documentation
        * Most projects will simply provide a list of user-facing features and options. See *Content Types* above.
    - Release Notes (both on the wiki and reStructuredText) as part of the release review.
- Documentation must be contributed to the docs repo (or possibly imported from the project's own repo with tooling that is under development)
    - Projects may be encouraged to instead provide this from their own repository if the tooling is developed
    - Projects choosing to meet the requirement in this way must provide a patch to docs repo to import the project's documentation
- Projects must cooperate with the documentation group on edits and enhancements to documentation

Timeline for Deliverables from Projects

- **M2:** Documentation Started

  The following tasks for documentation deliverables must be completed for the M2 readout:

  - The kinds of documentation that will be provided and for what features must be identified.

    Note: Release Notes are not required until release reviews at **RC2**

  - The appropriate `.rst` files must be created in the docs repository (or their own repository if the tooling is available).
  - An outline for the expected documentation must be completed in those `.rst` files including the relevant (sub)sections and a sentence or two explaining what will be contained in these sections.

    Note: If an outline is not provided, delivering actual documentation in the (sub)sections meets this requirement.

  - M2 readouts should include
    1. the list of kinds of documentation
    2. the list of corresponding `.rst` files and their location, including repo and path
    3. the list of commits creating those `.rst` files
    4. the current word counts of those `.rst` files
- **M3:** Documentation Continues
  - The readout at M3 should include the word counts of all `.rst` files with links to commits
  - The goal is to have draft documentation complete at the M3 readout so that the documentation group can comment on it.
- **M4:** Documentation Complete
  - All (sub)sections in all `.rst` files have complete, readable, usable content.
  - Ideally, there should have been some interaction with the documentation group about any suggested edits and enhancements
- **RC2:** Release notes
  - Projects must provide release notes in `.rst` format pushed to integration (or locally in the project's repository if the tooling is developed)

3.4 OpenDaylight Release Process Guide

3.4.1 Overview

This guide provides details on various processes related to OpenDaylight's release process and attempts to document the steps used by OpenDaylight Release Engineers to perform release operations.

3.4.2 Processes

Project Standalone Release

This page explains how a project can release independently outside of the OpenDaylight simultanious release.

Preparing your project for release

A project can produce a staging repository by clicking "build" for their {project-name}-maven-release-{stream} job. This job performs the following duties:

1. Removes -SNAPSHOT from all pom files
2. Produces a taglist.log, project.patch, and project.bundle files
3. Runs a *mvn clean deploy* to a local staging repo
4. Pushes the staging repo to a Nexus staging repo https://nexus.opendaylight.org/content/repositories/<REPO_ID> (REPO_ID is saved to staging-repo.txt on the log server)
5. Archives taglist.log, project.patch, and project.bundle files to log server

The files taglist.log and project.bundle can be used later at release time to reproduce a byte exact commit of what was built by the Jenkins job. This can be used to tag the release at release time.

Releasing your project

Once testing against the staging repo has been completed and project has determined that the staged repo is ready for release. A release can the be performed as follows:

1. Ask helpdesk to sign the artifacts in staging repo
2. Ask helpdesk to promote the staging repo
3. Download taglist.log and project.bundle
4. Read taglist.log and checkout the commit hash listed
5. Merge the project.bundle patches
6. Git tag the release
7. Push release tag to Gerrit

Steps 4-7 as bash:

```
PATCH_DIR=/tmp/patches
PROJECT=odlparent
VERSION=1.2.3
git checkout $(cat "$PATCH_DIR/taglist.log")
git fetch "$PATCH_DIR/$PROJECT.bundle"
git merge --ff-only FETCH_HEAD
git tag -asm "$PROJECT $VERSION" "v$VERSION"
git push origin "v$VERSION"
```

Once complete the Git tag should be available in Gerrit and the Artifacts should appear in the Nexus opendaylight.release repo.

Namespaces

Project namespaces in OpenDaylight are used to ensure projects do not have name collisions in code and packages. OpenDaylight enforces namespaces in Nexus using the following patterns:

- ^/org.opendaylight.PROJECT/.*
- ^/org/opendaylight/PROJECT/.*

Where PROJECT is the name of an OpenDaylight project.

In cases where a project has a sub-project we recommend adding an additional level to the path for example *org.opendaylight.integration.test* however no strong enforcement is currently enforced and some projects do this already internally.

This restriction applies to all site repositories in Nexus as well in the event that a project wishes to push a static web site into their allocated site path.

Maven / Java

Maven has a built in namespace routing using <groupId> field in pom files. For example:

```
<project>
  <groupId>org.opendaylight.odlparent</groupId>
  <artifactId>odlparent-lite</artifactId>
  <version>1.8.0-SNAPSHOT</version>
</project>
```

Python

Python projects typically publish to artifacts to PyPi and use their shortname for modules rather than a full path like Java projects do.

setup.py:

```
setup(
    name='spectrometer',
)
```

The structure of a Python project typically determines it's package routing. So a project package spectrometer.reporttool might have a layout like this inside their project root.

```
./  # This is the root of the repository
./setup.py
./spectrometer
./spectrometer/__init__.py
./spectrometer/reporttool
./spectrometer/reporttool/__init__.py
```

Autorelease

The Release Engineering - Autorelease project is targeted at building the artifacts that are used in the release candidates and final full release.

- Open Gerrit Patches

- Jenkins Jobs

Cloning Autorelease

To clone all the autorelease repo including it's submodules simply run the clone command with the ''–recursive'' parameter.

```
git clone --recursive https://git.opendaylight.org/gerrit/releng/autorelease
```

If you forgot to add the –recursive parameter to your git clone you can pull the submodules after with the following commands.

```
git submodule init
git submodule update
```

Creating Autorelease - Release and RC build

An autorelease release build comes from the autorelease-release-<branch> job which can be found on the autorelease tab in the releng master:

- https://jenkins.opendaylight.org/releng/view/autorelease/

For example to create a Boron release candidate build launch a build from the autorelease-release-boron job by clicking the ''Build with Parameters'' button on the left hand menu:

- https://jenkins.opendaylight.org/releng/view/autorelease/job/autorelease-release-boron/

Note: The only field that needs to be filled in is the ''RELEASE_TAG'', leave all other fields to their default setting. Set this to Boron, Boron-RC0, Boron-RC1, etc... depending on the build you'd like to create.

Adding Autorelease staging repo to settings.xml

If you are building or testing this release in such a way that requires pulling some of the artifacts from the Nexus repo you may need to modify your settings.xml to include the staging repo URL as this URL is not part of ODL Nexus' public or snapshot groups. If you've already cloned the recommended settings.xml for building ODL you will need to add an additional profile and activate it by adding these sections to the "<profiles>" and "<activeProfiles>" sections (please adjust accordingly).

Note:

- This is an example and you need to "Add" these example sections to your settings.xml do not delete your existing sections.
- The URLs in the <repository> and <pluginRepository> sections will also need to be updated with the staging repo you want to test.

```
<profiles>
  <profile>
    <id>opendaylight-staging</id>
    <repositories>
      <repository>
```

```
        <id>opendaylight-staging</id>
        <name>opendaylight-staging</name>
        <url>https://nexus.opendaylight.org/content/repositories/
↪automatedweeklyreleases-1062</url>
        <releases>
          <enabled>true</enabled>
          <updatePolicy>never</updatePolicy>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </repository>
    </repositories>
    <pluginRepositories>
      <pluginRepository>
        <id>opendaylight-staging</id>
        <name>opendaylight-staging</name>
        <url>https://nexus.opendaylight.org/content/repositories/
↪automatedweeklyreleases-1062</url>
        <releases>
          <enabled>true</enabled>
          <updatePolicy>never</updatePolicy>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </pluginRepository>
    </pluginRepositories>
  </profile>
</profiles>

<activeProfiles>
  <activeProfile>opendaylight-staging</activeProfile>
</activeProfiles>
```

Project lifecycle

This page documents the current rules to follow when adding and removing a particular project to Simultaneous Release (SR).

List of states

The state names are short negative phrases describing what is missing to progress to the following state.

- **non-existent** The project is not recognized by Technical Steering Committee (TSC) to be part of OpenDaylight (ODL).
- **non-participating** The project is recognized byt TSC to be an ODL project, but the project has not confirmed participation in SR for given release cycle.
- **non-building** The recognized project is willing to participate, but its current codebase is not passing its own merge job, or the project artifacts are otherwise unavailable in Nexus.
- **not-in-autorelease** Project merge job passes, but the project is not added to autorelease (git submodule, maven module, validate-autorelease job passes).

- **failing-autorelease** The project is added to autorelease (git submodule, maven module, validate-autorelease job passes), but autorelease build fails when building project's artifact. Temporary state, timing out into not-in-autorelease.
- **repo-not-in-integration** Project is succesfully built within autorelease, but integration/distribution:features-index is not listing all its public feature repositories.
- **feature-not-in-integration** Feature repositories are referenced, distribution-check job is passing, but some user-facing features are absent from integration/distribution:features-test (possibly because adding them does not pass distribution SingleFeatureTest).
- **distribution-check-not-passing** Features are in distribution, but distribution-check job is either not running, or it is failing for any reason. Temporary state, timing out into feature-not-in-integration.
- **feature-is-experimental** All user-facing features are in features-test, but at least one of the corresponding functional CSIT jobs does not meet Integration/Test requirements.
- **feature-is-not-stable** Feature does meet Integration/Test requirements, but it does not meed all requirements for stable features.
- **feature-is-stable**

Note: A project may change its state in both directions, this list is to make sure a project is not left in an invalid state, for example distribution referencing feature repositories, but without passing distribution-check job.

Branch Cutting

This page documents the current branch cutting tasks that are needed to be performed at various milestones and which team has the necessary permissions in order to perform the necessary task in Parentheses.

M5 Offset 2

JJB

- Export ${NEXT_RELEASE} and ${CURR_RELEASE} with new and current release names. **(releng/builder committers)**

```
export NEXT_RELEASE="Nitrogen"
export CURR_RELEASE="Carbon"
```

- Change JJB yaml files from *stream:carbon* branch pointer from *master -> stable/${CURR_RELEASE,,}* and create new *stream: ${NEXT_RELEASE,,}* branch pointer to branch master. This requires handling two different file formats interspersed with in autorelease projects. **(releng/builder committers)**

```
stream:
  - Nitrogen:
      branch: master
  - Carbon:
      branch: stable/carbon
```

```
- project:
    name: aaa-carbon
    jobs:
      - '{project-name}-verify-{stream}-{maven}-{jdks}'
```

```
    stream: nitrogen
    branch: master
```

  - The above manual process of updating individual files is automated with the script. **(releng/builder committers)**

```
cd builder/scripts/branch_cut
./branch_cutter.sh -n $NEXT_RELEASE -c $CURR_RELEASE
```

- Review and submit the changes to releng/builder project. **(releng/builder committers)**

Autorelease

- Block submit permissions for registered users and elevate RE's committer rights on gerrit. **(Helpdesk)**

Note: Enable **Exclusive** checkbox for the submit button to override any existing persmissions.

- Setup releng/autorelease repository. **(Release Engineering Team)**

```
git review -s
git submodule foreach 'git review -s'
git checkout master
git submodule foreach 'git checkout master'
git pull --rebase
git submodule foreach 'git pull --rebase'
```

- Create stable/${CURR_RELEASE} branches based on HEAD master. **(Release Engineering Team)**

```
git submodule foreach 'git checkout -b stable/${CURR_RELEASE,,} origin/master'
git push gerrit stable/${CURR_RELEASE,,}
git submodule foreach 'git push gerrit stable/${CURR_RELEASE,,}'
```

- Enable create reference permissions on gerrit for RE's to submit .gitreview patches. **(Helpdesk)**

Note: Enable Exclusive checkbox override any existing persmissions.

- Contribute .gitreview updates to stable/${CURR_RELEASE,,}. **(Release Engineering Team)**

```
git submodule foreach sed -i -e "s#defaultbranch=master#defaultbranch=stable/$
↪{CURR_RELEASE,,}#" .gitreview
git submodule foreach git commit -asm "Update .gitreview to stable/${CURR_RELEASE,
↪,}"
git submodule foreach 'git review -t ${CURR_RELEASE,,}-branch-cut'
sed -i -e "s#defaultbranch=master#defaultbranch=stable/${CURR_RELEASE,,}#" .
↪gitreview
git add .gitreview
git commit -s -v -m "Update .gitreview to stable/${CURR_RELEASE,,}"
git review -t  ${CURR_RELEASE,,}-branch-cut
```

- Merge all .gitreview patches submitted in the above step. **(Release Engineering Team)**
- Remove create reference permissions set on gerrit for RE's. **(Helpdesk)**
- Version bump master by x.(y+1).z. **(Release Engineering Team)**

```
git checkout master
git submodule foreach 'git checkout master'
pip install lftools
lftools version bump ${CURR_RELEASE}
```

- Exclude version bump changes to release notes. **(Release Engineering Team)**

```
git checkout pom.xml scripts/
```

- Push version bump master changes to gerrit. **(Release Engineering Team)**

```
git submodule foreach 'git commit -asm "Bump versions by x.(y+1).z for next dev␣
↪cycle"'
git submodule foreach 'git review -t nitrogen-br-cut'
```

- Merge all version bump patches in the order of dependencies. **(Release Engineering Team)**
- Re-enable submit permissions for registered users and disable elevated RE committer rights on gerrit. **(Helpdesk)**
- Notify release list on branch cutting work completion. **(Release Engineering Team)**

Release Schedule

While OpenDaylight has always targeted two releases per year, in practice our release process for the first six releases (through Carbon) has, in practice, released approximately every 8 months. This has meant we don't quite release twice a year (Lithium was our only release in 2015) and we struggle to coordinate releases with other projects that release at regular times each year, e.g., OpenStack and OPNFV.

To try to fix this, we are having a short Nitrogen release and then moving to a date-based, six-month release calendar releasing at the same time each year.

Nitrogen

milestone	offset 0	offset 1	offset 2	description
M0/M1	6/7/2017	6/14/2017	6/21/2017	Draft Release Plan
M2/M3/M4	6/28/2017	7/7/2017	7/14/2017	Final Release Plan, Functionality Freeze, API Freeze
M5	7/28/2017	8/7/2017	8/14/2017	Code Freeze
RC0	8/14/2017			
RC1	8/21/2017			
RC2	8/28/2017			
RC3	9/3/2017			
Release	9/7/2017			
SR1	10/7/2017			
SR2	12/7/2017			
SR3	2/7/2018			
SR4	3/21–5/7			

Note: Dates are calendar based on the 7th, 14th, 21st, and 28th of each month instead of being on a particular day of the week. The intent is that projects will figure out how to meet the deadline in the way that best works for them even if that means getting work done ahead of time to avoid holidays, weekends, vacation or travel.

Future Odd Releases

Starting with Oxygen, our odd-numbered element releases will look like this:

mile-stone	off0	off1	off2	Description
M0	9/7			Draft Release Plan
M1	10/7	10/14	10/21	Final Release Plan, Project Setup
M2	11/7	11/14	11/21	Functionality Freeze
M3	12/7	12/14	12/21	API Freeze
M4	1/7	1/14	1/21	Code Freeze *(note M3-M4 will likely be short since it includes 12/25-1/1)*
RCs	1/21-3/7			(continuous build)
Release	3/7			
SR1	4/7			
SR2	6/7			
SR3	8/7			
SR4	9/21-11/7			

Future Even Releases

Starting with Fluorine, our even-numbered element releases will look like this:

milestone	off0	off1	off2	Description
M0	3/7			Draft Release Plan
M1	4/7	4/14	4/21	Final Release Plan, Project Setup
M2	5/7	5/14	5/21	Functionality Freeze
M3	6/7	6/14	6/21	API Freeze
M4	7/7	7/14	7/21	Code Freeze
RCs	7/21-9/7			(continuous build)
Release	9/7			
SR1	10/7			
SR2	12/7			
SR3	2/7			
SR4	3/21-5/7			

Simultaneous Release

This page explains how the OpenDaylight release process works once the TSC has approved a release.

Preparations

After release candidate is built gpg sign artifacts using the lftools sign command.

```
STAGING_REPO=autorelease-1903
STAGING_PROFILE_ID=abc123def456  # This Profile ID is listed in Nexus > Staging␣
↪Profiles
lftools sign deploy-nexus https://nexus.opendaylight.org $STAGING_REPO $STAGING_
↪PROFILE_ID
```

Verify the distribution-karaf file with the signature.

```
gpg2 --verify distribution-karaf-x.y.z-${RELEASE}.tar.gz.asc distribution-karaf-x.y.z-
↪${RELEASE}.tar.gz
```

Releasing OpenDaylight

- Block submit permissions for registered users and elevate RE's committer rights on Gerrit. **(Helpdesk)**

Note: For all Nitrogen releases, also lock down the YANG tools version of the nitrogen branch (v1.2.x).

Note: Enable **Exclusive** checkbox for the submit button to override any existing persmissions. Code-Review and Verify permissions are only needed during version bumping. DO NOT enable it during code freeze.

- Nexus: click release for staging repo **(Helpdesk)**
- Nexus: click release for gpgsign repo (created above in Preparations) **(Helpdesk)**
- Pull latest autorelease repository **(Release Engineering Team)**

Note: If you already cloned autorelease the clone line can be skipped below.


```
export RELEASE=Nitrogen-SR1
export STREAM=${RELEASE//-*}
export BRANCH=origin/stable/${STREAM,,}

git clone --recursive https://git.opendaylight.org/gerrit/releng/autorelease
cd autorelease
git fetch origin

# Ensure we are on the right branch. Note that we are wiping out all
# modifications in the repo so backup unsaved changes before doing this.
git checkout -f
git checkout ${BRANCH,,}
git clean -xdff
git submodule update --init

# Ensure git review is setup
git review -s
git submodule foreach 'git review -s'
```

- Make sure the latest lftools is installed **(Release Engineering Team)**

Note: If you already created an lftools virtualenv you can skip the mkvirtualenv step below.

```
mkvirtualenv lftools
workon lftools
pip install --upgrade lftools
```

- Publish release tags **(Release Engineering Team)**

```
export BUILD_NUM=55
export PATCH_URL="https://logs.opendaylight.org/releng/vex-yul-odl-jenkins-1/
↪autorelease-release-${STREAM,,}/${BUILD_NUM}/patches.tar.gz"
./scripts/release-tags.sh "${RELEASE}" /tmp/patches "$PATCH_URL"
```

- Run autorelease-version-bump-${STREAM} job **(Release Engineering Team)**
- Send email to Helpdesk with binary URL to update website **(Helpdesk)**
- Send email to TSC and Release mailing lists announcing release binaries location **(Release Engineering Team)**
- Merge all patches generated by the job **(Release Engineering Team)**

- Re-enable submit permissions for registered users and disable elevated RE committer rights on gerrit **(Helpdesk)**
- Send email to release/tsc/dev notifying tagging and version bump complete **(Release Engineering Team)**
- Run autorelease-generate-release-notes-${STREAM} job **(Release Engineering Team)**

  Trigger this job by leaving a Gerrit comment *generate-release-notes Carbon-SR2*

  Important: This job can only be used to generate service releases.

  For major releases the notes come from the projects themselves in the docs repo via the *docs/releaset-notes/projects* directory.

  Release notes can also be manually generated with the script:

```
git checkout stable/${BRANCH,,}
./scripts/release-notes-generator.sh ${RELEASE}
```

  A *release-notes.rst* will be generated in the working directory.

Milestone Readouts

M0: Declare Intent

(Project Name)

1. A statement to the effect: "The <Project Name> project formally joins the OpenDaylight Carbon Simultaneous Release and agrees to the activities and timeline documented on the Carbon Release Plan Page: https://wiki.opendaylight.org/view/Simultaneous_Release:Carbon_Release_Plan"
2. Project Offset: (Offset 0/Offset 1/Offset 2)
3. Project Category: (Kernel/Protocol/Services/Application/Support)
4. Project Labels: (List keywords and tags and fit the description of your project comma separated)
5. Project PTL: (name/email/IRC)
6. Do you confirm that the list of Project Committers is updated and accurate? (Yes/No)

[1] https://wiki.opendaylight.org/view/Simultaneous_Release:Carbon_Release_Plan#M0:_Declare_Intent

M1: Draft Plan

(Project Name)

1. Project Lead Contact: (name/email/IRC)
2. Review PTL Requirements [1].
3. Project Contact: (name/email/IRC)
4. Test Contact: (name/email/IRC)
5. Documentation Contact (name/email/IRC)
6. Draft Release Plan: (wiki link)

   ** FOR NEW PROJECTS ONLY **
7. Project Main Page: (wiki link) Use Project Facts Template [2].

[1] Be sure to read the responsibilities of being a project lead under Leadership & Communication in the Requirements for Participation section of the release plan: https://wiki.opendaylight.org/view/Simultaneous_Release:Carbon_Release_Plan#Requirements_for_Participation

[2] https://wiki.opendaylight.org/view/Template:Project_Facts

M2: Final Release Plan

(Project Name)

1. Does your project have any updates on any previously-incomplete items from prior milestone readouts? (Yes/No)
    - (If yes, list updates)
2. Were project-specific deliverables planned for this milestone delivered successfully? (No Deliverables/Yes/No)
    - (If no, list incomplete deliverables)
3. Does your project have any special needs in CI Infrastructure [2]? (Yes/No)
    - (If yes, link to helpdesk ticket number)
4. Is your project release plan finalized? (Yes/No)
    - (If yes, link to final release plan wiki page)
    - (If no, ETA to finalize release plan)
5. Do you have all APIs intended to be externally consumable listed? (Yes/No)
    - Does each API have a useful short name? (Yes/No)
    - Are the Java interface and/or YANG files listed for each API? (Yes/No)
    - Are they labeled as tentative, provisional, or stable as appropriate for each API? (Yes/No)
    - Do you call out the OSGi bundles and/or Karaf features providing the API for each API? (Yes/No)
6. Have all project dependencies requests on other project's release plans been acknowledged and documented by upstream projects? (Yes/No)
    - (List of all project dependencies and if they have been acknowledged, unacknowledged)
7. Will your project have top-level features not requiring system test? (Yes/No)
    - (If yes, link to system test waiver request email)
8. Will your project use the OpenDaylight CI infrastructure for testing top-level features requiring system test? (Yes/No)
    - (If no, link to system test plan explaining why [3])
    - (If no, link to system test plan identifying external lab testing [4])

    ** FOR NEW PROJECTS ONLY **
9. Have you completed the project checklist [1]? (Yes/No)
    - (link to a merged patch in gerrit)
    - (link to a mail from your mailing list)
    - (link to a bug for your project; you can create a dummy one and close it if need be)
    - (link to an artifact published from your project in nexus)

- (link to a sonar report)
- (link to your root pom file)

[0] https://wiki.opendaylight.org/view/Simultaneous_Release:Carbon_Release_Plan

[1] https://wiki.opendaylight.org/view/GettingStarted:Project_Main#New_Project_Checklist

[2] Special needs include tools or configuration. Note that generally, the only available tools in CI are basic RHEL/CentOS linux images with Java. You should note and ask for anything beyond that here. Email helpdesk@opendaylight.org

[3] It is recommended to use the OpenDaylight CI infrastructure unless there is some HW or SW resource that cannot be installed there. Update the test plan with explanation on why your top-level features will not be using the OpenDaylight CI Infrastructure: https://wiki.opendaylight.org/view/CrossProject:Integration_Group:Feature_Integration_System_Test_Template#Test_Infrastructure

[4] Projects running system test in external Labs are required to report system test results in a timely fashion after release creations, e.g., weekly, RC, and formal releases. Update the test plan with plans on testing in external lab: https://wiki.opendaylight.org/view/CrossProject:Integration_Group:Feature_Integration_System_Test_Template#Test_Infrastructure

M3: Functionality Freeze

<Project Name>

Please provide updates on any previously-incomplete items from prior milestone readouts.

Functionality Freeze:

1. Final list of externally consumable APIs defined: Yes/No
   - **If you had an Tentative APIs, have they been moved to Provisional or dropped? Yes/No (link to** release plan)
   - If any of your Tentative APIs were dropped, have you notified all projects that were expecting them? Yes/No (link to e-mail)
     - Also please list all dropped APIs.
2. Are all your inter-project dependencies are resolved (i.e., have the other projects you were counting on given you what you needed)? Yes/No
   - If no, please list the features you were expecting that haven't been delivered and the project you were expecting to receive them from.
   - Note that you can only reasonably hold a a project to something if you formally asked for it during the release planning process and they acknowledged that ask saying they would do it.
3. Were there any project-specific deliverables planned for this milestone? Yes/No
   - If so, were they delivered? Yes/No

Karaf Features Defined:

1. Are all your project's features that are intended for release added to the features.xml and checked into integration git repository. Yes/No (please provide link to the gerrit patch)
2. List all top-level, user-facing, and stable Karaf features for your project.
   - For top-level and user-facing features, please provide a one-sentence description which a developer and/or user would find helpful.

Documentation:

1. List the kinds of documentation you will provide including at least:
    - One user/operator guide section per user-facing feature.
    - One developer guide per top-level feature.
    - An installation guide for any top-level features that require more than `feature:install <feature-name>` to install.
    - Eventually, release notes, but it is a good idea to keep release notes as a living document when significant changes others should be aware of are made.
    - Optional tutorials and how tos.
2. Have you checked in a reStructuredText outline for each of the documents you will provide to the docs repository? Yes/No (link to gerrit patch)

Integration and Test:

1. Have you started automated system testing for your top-level features. Yes/No
    - If yes, link to test report
    - If no, why?
2. Have you filled out basic system test plan template for each top-level feature (karaf and not karaf) and a comprehensive system test plan template including functionality, cluster, scalability, performance, longevity/stability for each stable feature? Yes/No
    - If yes, link to test plans
    - If no, why?

Project Specific:

1. Were there any project-specific deliverables planned for this milestone? Yes/No
    - If so, were they delivered? Yes/No
2. Have you updated your project facts with the project type category? Yes/No
3. Do you acknowledge the changes to the RC Blocking Bug Policy for Carbon Release [1]? Yes/No

[1] https://lists.opendaylight.org/pipermail/tsc/2016-December/006468.html

M4: API Freeze

<Project Name>

1. Please provide updates on any previously-incomplete items from prior milestone readouts.
2. Has your project achieved API freeze such that all externally accessible Stable or Provisional APIs will not be modified after now? (Yes/No)
    - (Link to gerrit search for patches modifying the API [1])
3. Do you have content in your project documentation? (Yes/No)
    - (For each document, provide current word count)
    - (For each document, link to the file in gerrit)
    - (Link to pending gerrit patches waiting approval)
4. Has your project met the requirements to be included in Maven Central [2]? (Yes/No)
5. Were project-specific deliverables planned for this milestone delivered successfully? (No Deliverables/Yes/No)

6. Have you started automated system testing for your top-level features. (Yes/No)
    - (If yes, link to test report)
    - (If no, explain why)
7. Does your project use any ports, including for testing? (Yes/No)
    - (If yes, list of ports used)
    - (If yes, have you updated the wiki [3] with all ports used? Yes/No)
8. Does your project build successful in Autorelease?
    - (If yes, link to successful autorelease job [4])
    - (If not, explain why)

[1] Provide a link to a gerrit search for patches modifying the files defined as specifying the API. For example: https://git.opendaylight.org/gerrit/#/q/file:%255Eopendaylight/md-sal/sal-binding-api/.%252B+status:merged+project:controller

[2] http://central.sonatype.org/pages/requirements.html

[3] https://wiki.opendaylight.org/view/Ports

[4] https://wiki.opendaylight.org/view/RelEng/Autorelease/Project_Autorelease_Requirements

M5: Code Freeze

<Project Name>

1. Please provide updates on any previously-incomplete items from prior milestone readouts.
2. Has your project met code freeze, i.e., only bug fixes are allowed from now on? (Yes/No)
3. Are all externally visible strings frozen to allow for translation & documentation? (Yes/No)
4. Is your documentation complete such that only editing and enhancing should take place after this point? (Yes/No)
    - (For each document, link to the file in gerrit)
    - (Link to pending gerrit patches waiting approval)
5. Were project-specific deliverables planned for this milestone delivered successfully? (No Deliverables/Yes/No)
6. Are you running at least one basic automated system test job for each top-level feature? (Yes/No)
    - (If yes, link to test report)
    - (If not, explain why)

Stables Features (Only for Projects with Stable Features)

1. Do your stable features fulfill quality requirements (i.e. unit and/or integration test coverage of at least 75%)? (Yes/No)
    - (If yes, link to sonar report)
    - (If not, explain why)
2. Are you running several automated system test jobs including functionality, cluster, scalability, performance, longevity/stability for each stable feature? (Yes/No)
    - (If yes, link to test reports)

- (If not, explain why)

RCX: Release Candidate Testing

<Project Name>

1. Have you tested your code in the release candidate? Yes/No (provide a link to the release candidate you tested)
   - If yes, did you find any issues?
   - If you found issues, do you believe any of them should block this release of OpenDaylight until they are resolved?
   - Please list all the issues and note if they are blocking.

3.4.3 Supporting Documentation

The release management team maintains several documents in Google Drive to track releases. These documents can be found at this link:

https://drive.google.com/drive/folders/0ByPlysxjHHJaUXdfRkJqRGo4aDg

3.5 Spectrometer Documentation

Contents:

3.5.1 Quick Start Guide

The Spectrometer project consists of two sub-projects, the `server` and `web`.

Server side is Python driven and provides the API to collect Git and Gerrit statistics for various OpenDaylight projects.

The web project is NodeJS/React based and provides the visualization by using the APIs provided by the server side.

In order to run the application, you need to install both `server` and `web` sub-projects.

This Quick Started Guide assumes you have Python3 and NodeJS 4.3 installed. To install NodeJS using NVM, see Web > Installation section below.

The Spectrometer project collects data from repositories located locally in your system.

Setup spectrometer-server

Installing spectrometer from pypi is simple and will get you the latest version that is released. Then create a config.py file in /etc/spectrometer/config.py (Example file can be found here)

```
pip install spectrometer
sudo mkdir /etc/spectrometer
sudo vi /etc/spectrometer/config.py
spectrometer server start
```

Verify that spectrometer-server is running by going to **http://localhost:5000**. You should see a Hello World page.

Setup spectrometer-web

Spectrometer Web is still in development so you will need to install it from Git at the time being as there is no package for it yet.

```
git clone https://git.opendaylight.org/gerrit/spectrometer.git
cd spectrometer/web
npm install
npm start
```

Goto **http://localhost:8000**

Testing the setup

By default the OpenDaylight project repositories will be mirrored every 5 minutes (300s), so if this is the first time starting you may have to wait until all repos are mirrored before you can exercise some of the apis.

Once the repos are mirrored you can try a few basic examples to make sure things are working properly:

Examples:

```
http://127.0.0.1:5000/gerrit/branches?project=controller
http://127.0.0.1:5000/gerrit/projects
http://127.0.0.1:5000/git/commits?project=integration/packaging
```

The full Rest APIs are documented here: https://opendaylight-spectrometer.readthedocs.io/en/latest/restapi.html

3.5.2 User Guide

Spectrometer consists of 3 components:

- Spectrometer API Server (backend)
- Spectrometer Web Server (frontend)
- Spectrometer Report Tool

This guide will describe the uses of the 3 systems.

Spectrometer API Server

Production Deployment

When running in production the recommended way is to deploy with gunicorn.

```
gunicorn -b 0.0.0.0:5000 'spectrometer:run_app()'
```

If deploying behind a proxy under a sub-directory additional configuration is necessary for gunicorn application to operate correctly.

example-nginx:

```
location /api {
    proxy_pass         http://127.0.0.1:5000;
    proxy_redirect     http://127.0.0.1:5000/api/ http://$host/api/;

```

```
    proxy_set_header   Host              $host;
    proxy_set_header   X-Real-IP         $remote_addr;
    proxy_set_header   X-Forwarded-For   $proxy_add_x_forwarded_for;
    proxy_set_header   SCRIPT_NAME       /api;
}
```

Logging

Spectrometer logs to /var/log/spectrometer by default but that directory must be writeable by the spectrometer user.

```
sudo chown spectrometer /var/log/spectrometer
```

It is possible to override the default log directory by configuring the LOG_DIR parameter in config.py.

```
LOG_DIR = '/path/to/log/directory'
```

Spectrometer Web Server

TODO

Spectrometer Report Tool

The Spectrometer Report Tool can be used to generate reports between 2 reference points in time. Reference points are git commit hashs, branches, or tags. A project like OpenDaylight that tags projects with the same tag name for every release can use this tool to Generate release reports.

```
# spectrometer reporttool full <ref1> <ref2>
spectrometer reporttool --server-url=https://spectrometer.opendaylight.org/api full␣
↪release/beryllium-sr2 release/beryllium-sr1
```

3.5.3 Project Info Specification

Spectrometer supports a PROJECT_INFO.yaml file placed in the root of a project repo. This file is used by spectrometer to parse meta information about the project including things like project description, project contact, committers irc, mailing lists, release names, etc...

```
# This file is used by Spectrometer to determine project meta information
# Please refer to the spec file located here:
# https://opendaylight-spectrometer.readthedocs.io/en/latest/project-info-spec.html

name: spectrometer
display-name: Spectrometer
creation-date: 2015-11-19
termination-date: n/a
description: |
    This is an example summary description of project

    After leaving a blank line in the description we can provide a longer
    more detailed description of the project.

    The details can be as many lines as necessary.
```

```
primary-contact: Firstname Lastname <first.last@example.com>
project-lead: Firstname Lastname <first.last@example.com>
categories:
    - application
    - community
    - documentation
    - extensions
    - kernel
    - library
    - protocols
    - services
committers:
    - Firstname Lastname <first.last@example.com>
    - Another Committer <another.committer@example.com>
# When Committers who have made significant contributions to OpenDaylight
# become inactive and thus no longer committers. This key can be used to
# acknowledge their huge contributions by appointing them to Committer
# Emeritus status.
committers-emeritus:
    - Firstname Lastname <first.last@example.com>
contributors:
    - Firstname Lastname <first.last@example.com>
    - Another Contributor <another.contributor@example.com>
wiki: https://wiki.example.org/project
irc: irc://irc.freenode.net/opendaylight-spectrometer
mailing-lists:
    - email: spectrometer-dev@lists.opendaylight.org
      archives: http://lists.opendaylight.org/pipermail/spectrometer-dev/
    - email: spectrometer-users@lists.opendaylight.org
      archives: http://lists.opendaylight.org/pipermail/spectrometer-users/
ci-server: https://jenkins.opendaylight.org
issue-tracker: https://bugs.opendaylight.org
static-analysis: https://sonar.opendaylight.org
repository: https://git.opendaylight.org/gerrit/#/admin/projects/spectrometer
meetings: |
    Free from text field for providing meeting information.
    It can be multiple lines long as necessary.
releases:
    - helium
    - lithium
    - beryllium
    - boron
```

Required fields:

- name
- creation_date
- description
- primary_contact
- project_lead

3.5.4 Documentation Guide

This guide provides details on how to contribute to the documetantion of Spectrometer. The style guide we follow for documentation is the python documentation style guide. See:

https://docs.python.org/devguide/documenting.html

To build and review the documentation locally you can simply run tox and open the html via your favourite web browser.

```
tox -edocs
firefox .tox/docs/tmp/html/index.html
```

3.5.5 Developer Guide

This doc provides details for developers who want to hack on spectrometer. If you have not done so already please refer to the *Quick Start Guide*.

Style Guide

We follow the Python PEP8 style guide. See: https://www.python.org/dev/peps/pep-0008/

For documentation we follow the Python Documentation Guide. See: https://docs.python.org/devguide/documenting.html

Spectrometer Server

Installing in Dev Mode

In development we want to install spectrometer so that we can modify the code and use it as if in production with changes taking effect immediately. We can achieve this using pip's editable install mode.

```
cd server  # From spectrometer repo root
pip install -e .
spectrometer server -c example-config/config.py start
```

Testing Code

We use tox to manage and run our unit tests. Simply run **tox** in the server directory to initiate the tests. If you don't have tox installed typically it is packaged as **python-tox** in most distros.

```
cd server/  # From spectrometer repo root
tox
```

Spectrometer Web

Installation

To install NodeJS in your system, use the Node Version Manager (NVM), which allows to co-exist multiple NodeJS versions in the same system.

If you already have NodeJS older versions (<= 0.12), it is strongly recommended to completely remove them and reinstall using NVM.

For Linux systems, you can do the following to remove NodeJS:

```
which node # Note down the path
sudo rm -r /path/bin/node /path/bin/npm /path/include/node /path/lib/node_modules ~/.
↪npm
```

Install NVM, NodeJS 4.3.x and NPM:

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.31.0/install.sh | bash
nvm install 4.3.1   # By default this installs npm 2.14.x
npm install npm -g  # This will upgrade npm to 3.7.x
```

Run spectrometer-web

cd web # From the root of the git repo npm install npm start

Goto `http://localhost:8000`

The web project is configured to hot-reload when any changes are made to the code. Most of the time the web browser should auto refresh, if not simply refresh the page.

UI Technology Stack

- NodeJS 4.3 - Bootstrapping and Universal (isomorphic) Javascript execution
- ExpressJS - Web-server-side bootstrap for UI
- ReactJS 0.14 - View Layer
- Redux - Data and State management (Flux pattern)
- Webpack - Build tool
- Babel - Asset compilation, ES6 Transpiler
- FormidableLabs VictoryChart - D3-based React components
- Redux Dev Tools - Tool that allows to track state management

Run spectrometer-web in Production

Production build does not have Devtools and hot reloading middleware. It also minifies scripts and css.

For Production build, execute the following commands:

```
npm run build
npm run start-prod
```

Run Test

Unit Tests are executed using Mocha and Chai assert libraries.

```
npm test
```

Roadmap

1. Dynamic loading of repositories as opposed to loading via config.json

Troubleshooting

Adding new repository

In order to add a new repository to collect statistics, you must make the following changes:

1. Create a soft link in ~/odl-spectrometer to the new repository
2. Edit the server/spectrometer/etc/repositories.yaml and specify the key and path to ~/odl-spectrometer/$repo
3. Edit the web/src/config.json add the project name in the list (this makes it appear in the dropdown)
4. Reload the web page
5. If reload web page does not work, restart python `python spectrometer-server` and web `npm start`)

3.5.6 Rest API

Gerrit API

spectrometer.api.gerrit.**branches**()

Returns a list of branches in a given repository by querying Gerrit.

GET /gerrit/branches?param=<value>

Parameters **project** (*str*) – Project to query branches from. (required)

JSON:

```
{
  "branches": [
    {
      "ref": "refs/heads/stable/beryllium",
      "revision": "8f72284f3808328604bdff7f91a6999094f7c6d7"
```

```
    },
    ...
    ]
}
```

spectrometer.api.gerrit.**merged_changes**()

Returns a list of merged changes in a given repository by querying Gerrit.

GET /gerrit/changes?param=<value>

Parameters

- **project** (*str*) – Project to query changes from. (required)
- **branch** (*str*) – Branch to pull changes from. (default: master)

JSON:

```
{
  "changes": [
    {
      "_number": 37706,
      "branch": "master",
      "change_id": "I4168e023b77bfddbb6f72057e849925ba2dffa17",
      "created": "2016-04-18 02:42:33.000000000",
      "deletions": 0,
      "hashtags": [],
      "id": "spectrometer~master~I4168e023b77bfddbb6f72057e849925ba2dffa17",
      "insertions": 119,
      "owner": {
        "_account_id": 2759
      },
      "project": "spectrometer",
      "status": "MERGED",
      "subject": "Add API to return commits since ref",
      "submittable": false,
      "topic": "git-api",
      "updated": "2016-04-19 09:03:03.000000000"
    },
    ...
    ]
}
```

spectrometer.api.gerrit.**projects**()

Returns a list of projects by querying Gerrit.

GET /gerrit/projects

JSON:

```
{
  "projects": [
    "groupbasedpolicy",
    "spectrometer",
    "releng/autorelease",
    "snmp4sdn",
    "ovsdb",
    "nemo",
    ...
```

```
    ]
}
```

spectrometer.api.gerrit.**tags**()
Returns a list of tags in a given repository by querying Gerrit.

GET /gerrit/tags?param=<value>

Parameters **project** (*str*) – Project to query tags from. (required)

JSON:

```
{
  "tags": [
    {
      "message": "OpenDaylight Beryllium-SR1 release",
      "object": "f76cc0a12dc8f06dae3cedc31d06add72df8de5d",
      "ref": "refs/tags/release/beryllium-sr1",
      "revision": "8b92d614ee48b4fc5ba11c3f38c92dfa14d43655",
      "tagger": {
        "date": "2016-03-23 13:34:09.000000000",
        "email": "thanh.ha@linuxfoundation.org",
        "name": "Thanh Ha",
        "tz": -240
      }
    },
    ...
    ]
}
```

Git API

spectrometer.api.git.**branches**()
Returns a list of branches in a given repository.

GET /git/branches?param=<value>

Parameters **project** (*str*) – Project to query commits from. (required)

JSON:

```
{
  "branches": [
    "master",
    "stable/beryllium",
    "stable/helium",
    "stable/lithium",
    ...
  ]
}
```

spectrometer.api.git.**commits**()
Returns a list of commit messages in a repository.

GET /git/commits?param=<value>

Parameters

- **project** (*str*) – Project to query commits from. (required)

- **branch** (*str*) – Branch to pull commits from. (default: master)

JSON:

```
{
  "commits": [
    {
      "author": "Thanh Ha",
      "author_email": "thanh.ha@linuxfoundation.org",
      "author_tz_offset": 14400,
      "authored_date": 1460316386,
      "committed_date": 1460392605,
      "committer": "Thanh Ha",
      "committer_email": "thanh.ha@linuxfoundation.org",
      "committer_tz_offset": 14400,
      "hash": "1e409af62fd99413c5be86c5b43ad602a8cebc1e",
      "lines": {
        "deletions": 55,
        "files": 7,
        "insertions": 103,
        "lines": 158
      },
      "message": "Refactor Gerrit API into a Flask Blueprint..."
    },
    ...
  ]
}
```

Note:

date The date represented in seconds since epoch

tz_offset The seconds offset west of UTC.

---

spectrometer.api.git.**commits_since_ref**()

Returns a list of commits in branch until common parent of ref.

Searches Git for a common_parent between *ref1* and *ref2* and returns a the commit log of all the commits until the common parent excluding the common_parent commit itself.

GET /git/commits_since_ref?param=<value>

Parameters

- **project** (*str*) – Project to query commits from. (required)
- **ref1** (*str*) – Reference to get commit information from. (required)
- **ref2** (*str*) – Reference to start at until ref1. (required)

JSON:

```
{
  "commits": [
    {
      "author": "Thanh Ha",
      "author_email": "thanh.ha@linuxfoundation.org",
      "author_tz_offset": 14400,
      "authored_date": 1460316386,
      "committed_date": 1460392605,
```

```
      "committer": "Thanh Ha",
      "committer_email": "thanh.ha@linuxfoundation.org",
      "committer_tz_offset": 14400,
      "hash": "1e409af62fd99413c5be86c5b43ad602a8cebc1e",
      "lines": {
        "deletions": 55,
        "files": 7,
        "insertions": 103,
        "lines": 158
      },
      "message": "Refactor Gerrit API into a Flask Blueprint..."
    },
    ...
  ]
}
```


spectrometer.api.git.**project_info**()
Provides meta information on project.

Refer to the specfile located here: https://opendaylight-spectrometer.readthedocs.io/en/latest/project-info-spec.html

3.6 Genius Documentation

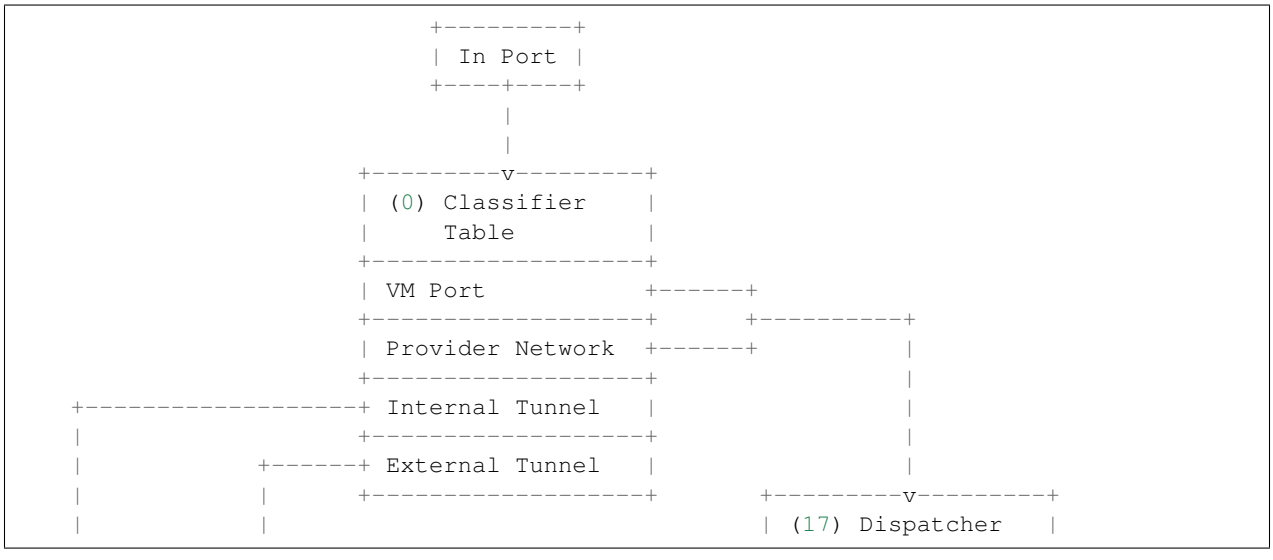
This documentation provides critical information needed to help you write ODL Applications/Projects that can co-exist with other ODL Projects.

Contents:

3.6.1 Genius Pipeline

This document captures current OpenFlow pipeline as use by Genius and projects using Genius for app-coexistence.

High Level Pipeline



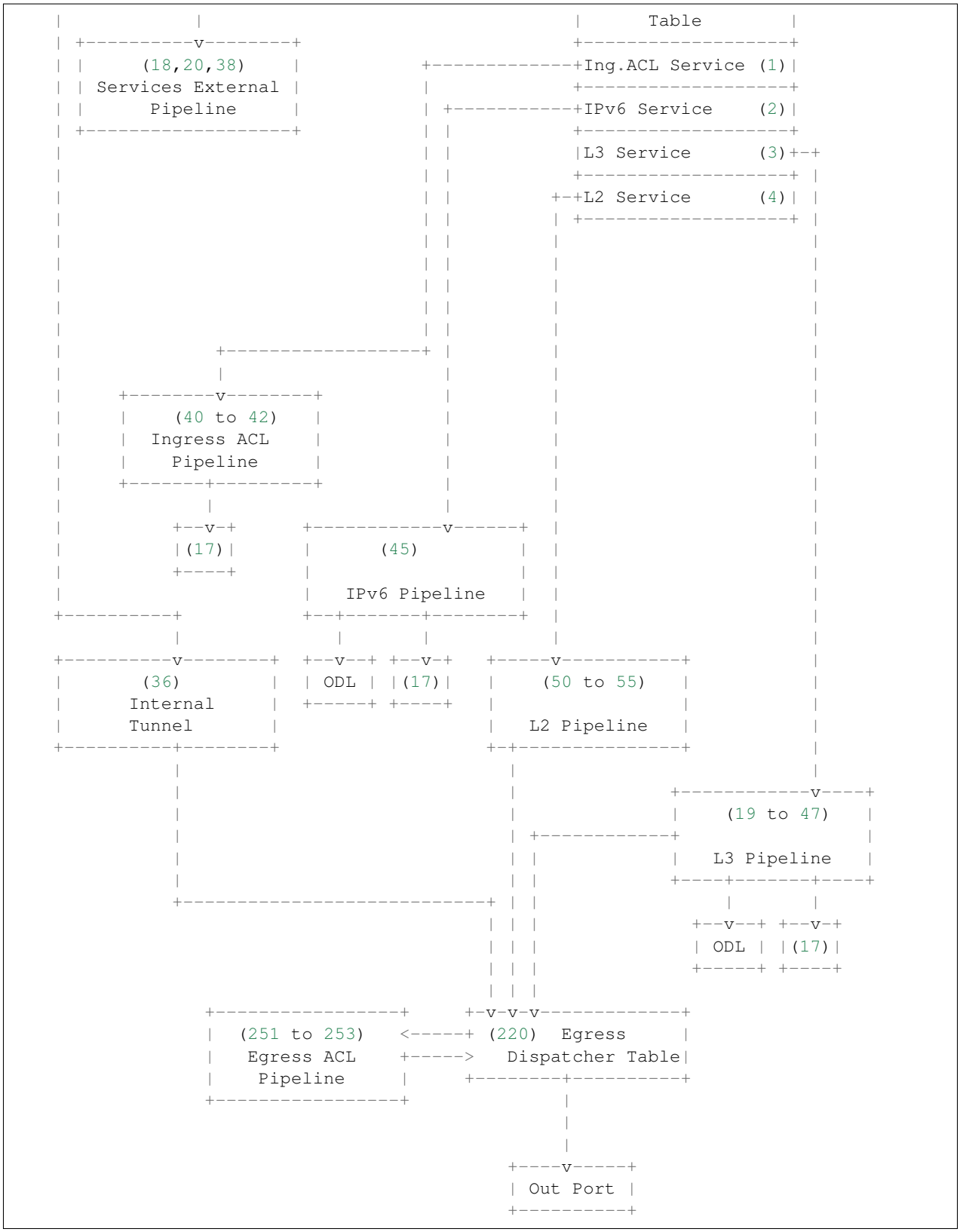
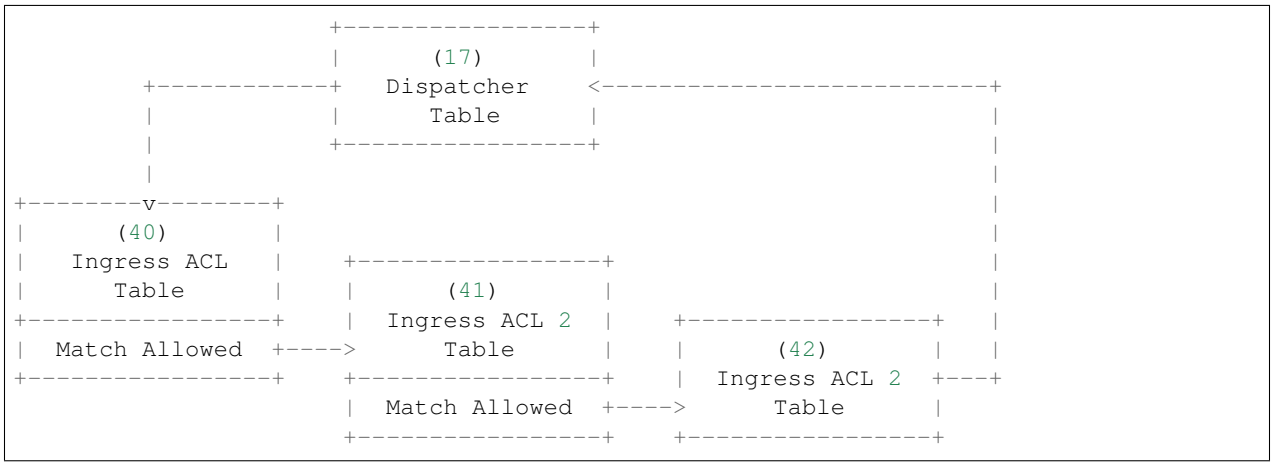


Table
Ing.ACL Service (1)
IPv6 Service (2)
L3 Service (3)
L2 Service (4)
(18,20,38)
Services External
Pipeline
(40 to 42)
Ingress ACL
Pipeline
(17)
(45)
IPv6 Pipeline
(36)
Internal
Tunnel
ODL
(17)
(50 to 55)
L2 Pipeline
(19 to 47)
L3 Pipeline
ODL
(17)
(251 to 253)
Egress ACL
Pipeline
(220) Egress
Dispatcher Table
Out Port

Services Pipelines

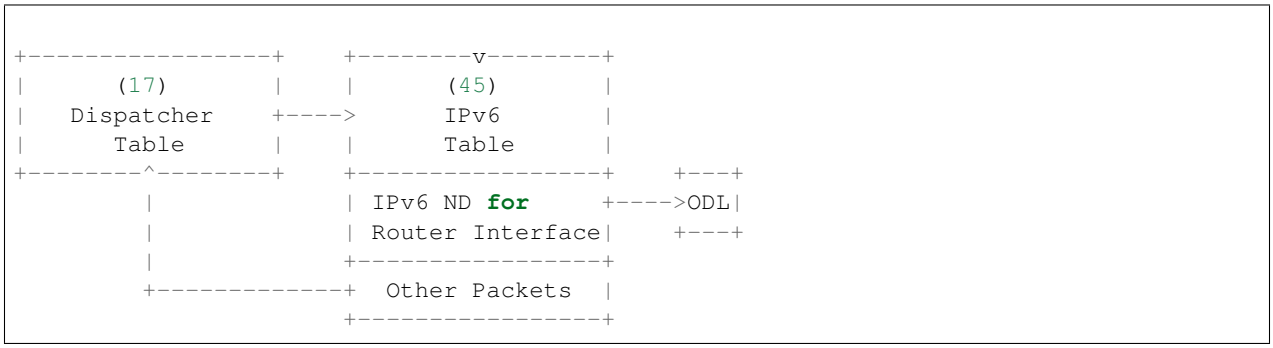
Ingress ACL Pipeline



Owner Project: Netvirt

TBD.

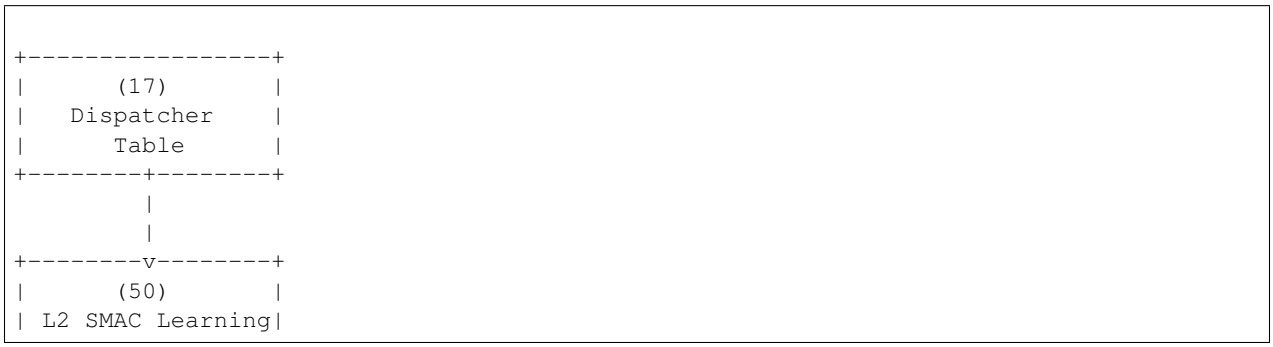
IPv6 Pipeline

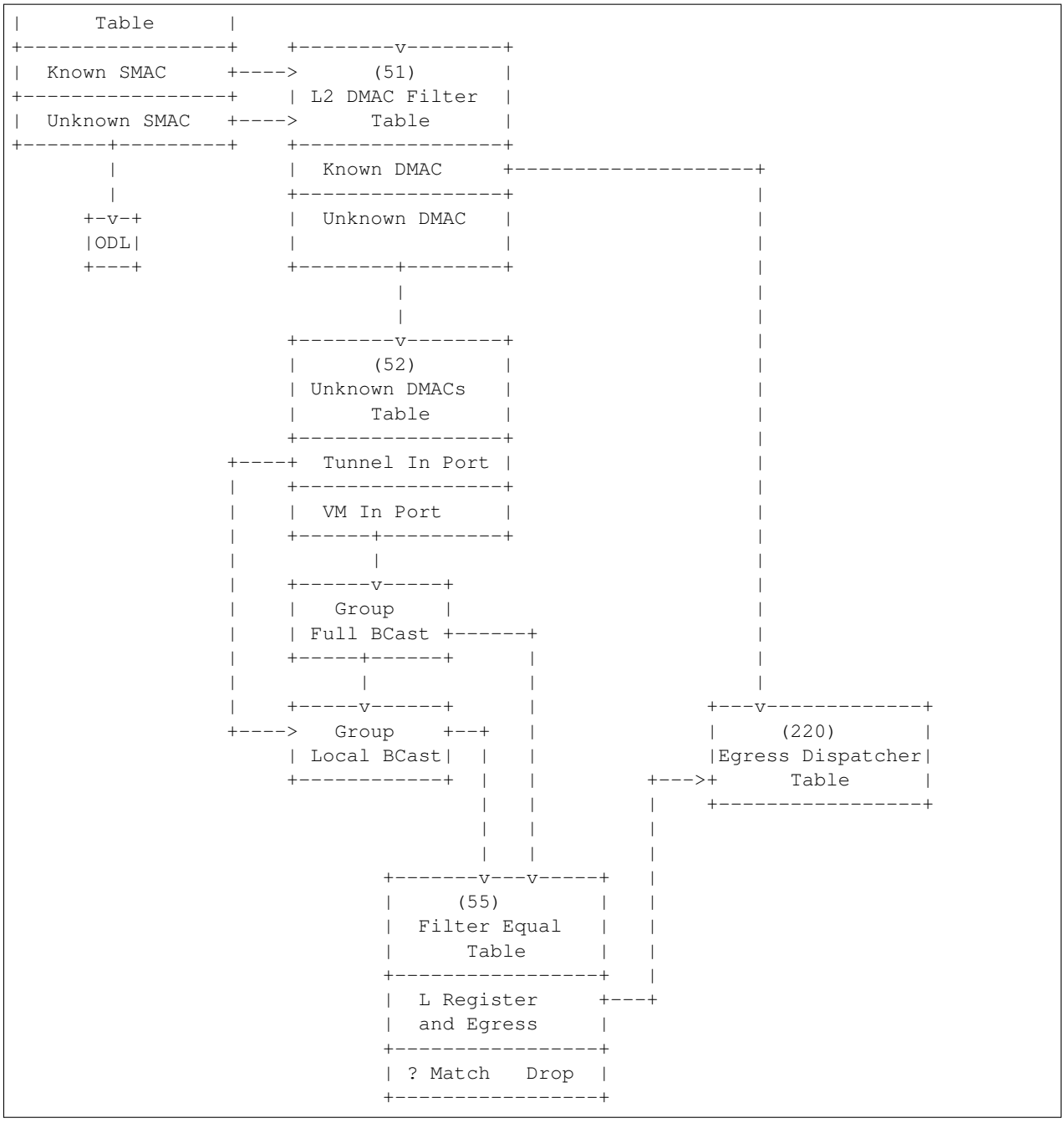


Owner Project: Netvirt

TBD.

L2 Pipeline

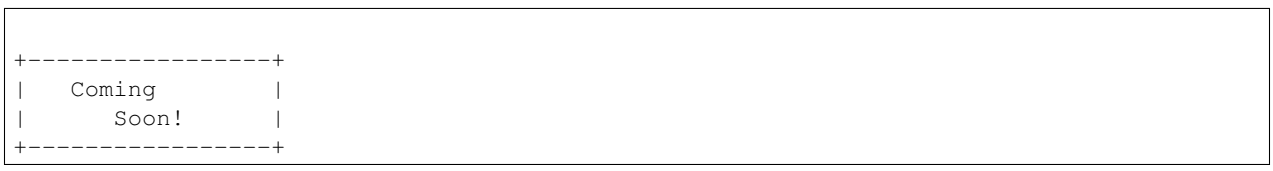




Owner Project: Netvirt

TBD.

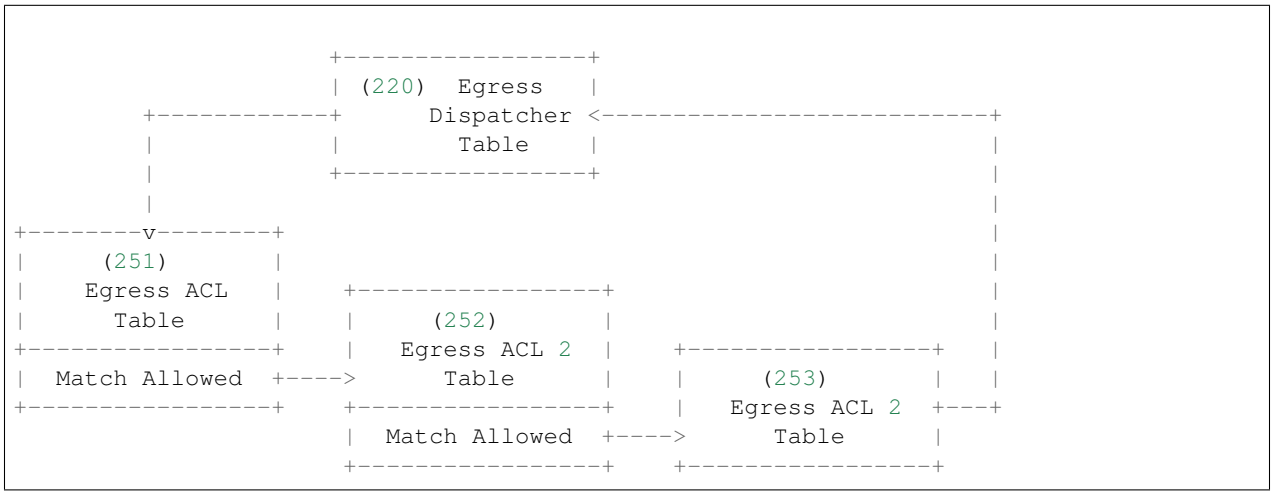
L3 Pipeline



Owner Project: Netvirt

TBD.

Egress ACL Pipeline



Owner Project: Netvirt

TBD.

3.6.2 Genius Design Overview

Genius project provides generic infrastructure services and utilities for integration and co-existance of mulltiple networking services/applications. Following image presents a top level view of Genius framework -

Genius Module Dependencies

Genius modules are developed as karaf features which can be independently installed. However, there is some dependency among these modules. The diagram below provides a dependency relationship of these modules.

All these modules expose Yang based API which can be used to configure/interact with these modules and fetch services provided by these modules. Thus all these modules can be used/configured by other ODL modules and can also be accessed via REST interface.

Genius based packet pipeline

Following picture presents an example of packet pipeline based on Genius framework. It also presents the functions of diffrent genius components -

Following sections provide details about each of these components.

Interface Manager Design

The Interface Manager (IFM) uses MD-SAL based architecture, where different software components operate on, and interact via a set of data-models. Interface manager defines configuration data-stores where other OpenDaylight modules can write interface configurations and register for services. These configuration data-stores can also be accessed by external entities through REST interface. IFM listens to changes in these config data-stores and accordingly programs the data-plane. Data in Configuration data-stores remains persistent across controller restarts.

Operational data like network state and other service specific operational data are stored in operational data-stores. Change in network state is updated in southbound interfaces (OFplugin, OVSDB) data-stores. Interface Manager uses ODL Inventory and Topology datastores to retrive southbound configurations and events. IFM listens to these updates and accordingly updates its own operational data-stores. Operational data stores are cleaned up after a controller restart.

Additionally, a set of RPCs to access IFM data-stores and provide other useful information. Following figure presents different IFM data-stores and its interaction with other modules.

Follwoing diagram provides a toplevel architecture of Interface Manager.

InterfaceManager Dependencies

Interface Manager uses other Genius modules for its operations. It mainly interacts with following other genius modules-

1. **Id Manager** – For allocating dataplane interface-id (if-index)
2. **Aliveness Monitor** - For registering the interfaces for monitoring
3. **MdSalUtil** – For interactions with MD-SAL and other openflow operations

Following picture shows interface manager dependencies

Code structure

Interface manager code is organized in following folders -

1. **interfacemanager-api** contains the interface yang data models and corresponding interface implementation.
2. **interfacemanager-impl** contains the interfacemanager implementation
3. **interface-manager-shell** contains Karaf CLI implementation for interfacemanager

interfacemanager-api

```
---main
   ---java
   |   ---org
   |       ---opendaylight
```

```
|           ---genius
|               ---interfacemanager
|                   ---exceptions
|                   ---globals
|                   ---interfaces
---yang
```

interfacemanager-impl

```
---commons                  <--- contains common utility functions

---listeners                 <--- Contains interfacemanager DCN listenenrs for differnt MD-

---renderer                  <--- Contains different southbound renderers' implementation
|   ---hwvtep                <--- HWVTEP specific renderer
|   |   ---confighelpers
|   |   ---statehelpers
|   |   ---utilities
|   ---ovs                   <--- OVS specific SBI renderer
|       ---confighelpers
|       ---statehelpers
|       ---utilities

---servicebindings           <--- contains interface service binding DCN listener and corre
|   ---flowbased
|       ---confighelpers
|       ---listeners
|       ---statehelpers
|       ---utilities

---rpcservice                <--- Contains interfacemanager RPCs' implementation
---pmcounters                <--- Contains PM counters gathering

---statusanddiag             <--- contains status and diagnostics implementations
```

'interfacemanager-shell

Interfacemanager Data-model

FOllowing picture shows different MD-SAL datastores used by intetrface manager. These datastores are created based on YANG datamodels defined in interfacemanager-api.

Config Datastores

InterfaceManager mainly uses following two datastores to accept configurations.

1. **odl-interface** datamodel () where verious type of interface can be configuted.
2. **service-binding** datamodel () where different applications can bind services to interfaces.

In addition to these datamodels, it also implements several RPCs for accessing interface operational data. Details of these datamodels and RPCs are described in following sections.

Interface Config DS

Interface config datamodel is defined in odl-interface.yang . It is based on 'ietf-interfaces' datamodel (imported in odl_interface.yang) with additional augmentations to it. Common interface configurations are –

- **name (string)** : this is the unique interface name/identifier.
- **type (identityref:iana-if-type)** : this configuration sets the interface type. Interface types are defined in iana-if-types data model. Odl-interfaces.yang data model adds augmentations to iana-if-types to define new interface types. Currently supported interface types are -
  - **l2vlan** (trunk, vlan classified sub-ports/trunk-member)
  - **tunnel** (OVS based VxLAN, GRE, MPLSoverGRE/MPLSoverUDP)
- **enabled (Boolean)** : this configuration sets the administrative state of the interface.
- **parent-refs :** this configuration specifies the parent of the interface, which feeds data/hosts this interface. It can be a physical switch port or a virtual switch port.
  - Parent-interface (string) : is the port name with which a network port in dataplane in that appearing on the southbound interface. E.g. neutron port. this can also be another interface, thus supporting a hierarchy of linked interfaces.
  - Node-identifier (topology_id, node_id) : is used for configuring parent node for HW nodes/VTEPs

Additional configuration parameters are defined for specific interface type. Please see the table below.

Vlan-xparent	**Vlan-trunk**	**Vlan-trunk-member**	**vxlan**	**gre**
Name =uuid	**Name** =uuid	**Name** =uuid	**Name** =uuid	**Name** =uuid
description	description	description	description	description
Type =l2vlan	**Type** =l2valn	**Type** =l2vlan	**Type** =tunnel	**Type** =tunnel
enabled	**enabled**	**enabled**	**enabled**	**enabled**
Parent-if = port-name	**Parent-if** = port-name	**Parent-if** = vlan-trunkIf	Vlan-id	Vlan-id
vlan-mode = transparent	**vlan-mode** = trunk	**vlan-mode** = trunk-member	**tunnel-type** = vxlan	**tunnel-type** = gre
	vlan-list= [trunk-member-list]	**Vlan-Id** = trunk-vlanId	dpn-id	dpn-id
		Parent-if = vlan-trunkIf	Vlan-id	Vlan-id
			local-ip	**local-ip**
			remote-ip	**remote-ip**
			gayeway-ip	**gayeway-ip**

Interface service binding config

Yang Data Model odl-interface-service-bindings.yang contains the service binding configuration daatmodel.

An application can bind services to a particular interface by configuring MD-SAL data node at path /config/interface-service-binding. Binding services on interface allows particular service to pull traffic arriving on that interface, depending upon the a service priority. It is possible to bind services at ingress interface (when packet enters into the packet-pipeline from particular interface) as well as on the egress Interface (before the packet is sent out on particular interafce). Service modules can specify openflow-rules to be applied on the packet belonging to the interface. Usually these rules include sending the packet to specific service table/pipeline. Service modules/applications are responsible for sending the packet back (if not consumed) to service dispatcher table, for next service to process the packet.

Following are the service binding parameters –

- **interface-name** is name of the interface to which service binding is being configured
- **Service-Priority** parameter is used to define the order in which the packet will be delivered to different services bind to the particular interface.
- Service-Name
- **Service-Info** parameter is used to configure flow rule to be applied to the packets as needed by services/applications.
  - (for service-type openflow-based)
  - Flow-priority
  - Instruction-list

When a service is bind to an interface, Interface Manager programs the service dispatcher table with a rule to match on the interface data-plane-id and the service-index (based on priority) and the instruction-set provided by the service/application. Every time when the packet leaves the dispatcher table the service-index (in metadata) is incremented to match the next service rule when the packet is resubmitted back to dispatcher table. Following table gives an example of the service dispatcher flows, where one interface is bind to 2 services.

Service Dispatcher Table	
Match	**Actions**
• if-index = I • ServiceIndex = 1	• Set SI=2 in metadata • service specific actions <e.g., Goto prio 1 Service table>
• if-index = I • ServiceIndex = 2	• Set SI=3 in metadata • service specific actions <e.g., Goto prio 2 Service table>
miss	Drop

Interface Manager programs openflow rules in the service dispatcher table.

Egress Service Binding

There are services that need packet processing on the egress, before sending the packet out to particular port/interface. To accommodate this, interface manager also supports egress service binding. This is achieved by introducing a new "egress dispatcher table" at the egress of packet pipeline before the interface egress groups.

On different application request, Interface Manager returns the egress actions for interfaces. Service modules program use these actions to send the packet to particular interface. Generally, these egress actions include sending packet out to port or appropriate interface egress group. With the inclusion of the egress dispatcher table the **egress actions** for the services would be to

- Update REG6 - Set service_index =0 and egress if_index

- send the packet to Egress Dispatcher table

IFM shall add a default entry in Egress Dispatcher Table for each interface With -

- Match on if_index with REG6
- Send packet to corresponding output port or Egress group.

On Egress Service binding, IFM shall add rules to Egress Dispatcher table with following parameters –

- Match on
  - ServiceIndex=egress Service priority
  - if_index in REG6 = if_index for egress interface
- Actions
  - Increment service_index
  - Actions provided by egress service binding.

Egress Services will be responsible for sending packet back to Egress Dispatcher table, if the packet is not consumed (dropped/ send out). In this case the packet will hit the lowest priority default entry and the packet will be send out.

Operational Datastores

Interface Manager uses ODL Inventory and Topology datastores to retrive southbound configurations and events.

Interface Manager modules

Interface manager is designed in a modular fashion to provide a flexible way to support multiple southbound protocols. North-bound interface/data-model is decoupled from south bound plugins. NBI Data change listeners select and interact with appropriate SBI renderers. The modular design also allows addition of new renderers to support new southbound interfaces, protocols plugins. Following figure shows interface manager modules –

InterfaceManager uses the datastore-job-coordinator module for all its operations.

Datastore Job Coordination framework

The datastore job coordinator framework offers the following benefits :

1. “Datastore Job” is a set of updates to the Config/Operational Datastore.
2. Dependent Jobs (eg. Operations on interfaces on same port) that need to be run one after the other will continue to be run in sequence.
3. Independent Jobs (eg. Operations on interfaces across different Ports) will be allowed to run paralelly.
4. Makes use of ForkJoin Pools that allows for work-stealing across threads. ThreadPool executor flavor is also available... But would be deprecating that soon.
5. Jobs are enqueued and dequeued to/from a two-level Hash structure that ensures point 1 & 2 above are satisfied and are executed using the ForkJoinPool mentioned in point 3.

6. The jobs are enqueued by the application along with an application job-key (type: string). The Coordinator dequeues and schedules the job for execution as appropriate. All jobs enqueued with the same job-key will be executed sequentially.
7. DataStoreJob Coordination to distribute jobs and execute them paralelly within a single node.
8. This will still work in a clustered mode by handling optimistic lock exceptions and retrying of the job.
9. Framework provides the capability to retry and rollback Jobs.
10. Applications can specify how-many retries and provide callbacks for rollback.
11. Aids movement of Application Datastore listeners to "Follower" also listening mode without any change to the business logic of the application.
12. Datastore Job Coordination function gets the list of listenable futures returned from each job.
13. The Job is deemed complete only when the onSuccess callback is invoked and the next enqueued job for that job-key will be dequeued and executed.
14. On Failure, based on application input, retries and/or rollback will be performed. Rollback failures are considered as double-fault and system bails out with error message and moves on to the next job with that Job-Key.

Datastore job coordinator solves the following problems which is observed in the previous Li-based interface manager :

1. The Business Logic for the Interface configuration/state handling is performed in the Actor Thread itself.
2. This will cause the Actor's mailbox to get filled up and may start causing unnecessary back-pressure.
3. Actions that can be executed independently will get unnecessarily serialized.
4. Can cause other unrelated applications starve for chance to execute.
5. Available CPU power may not be utilized fully. (for instance, if 1000 interfaces are created on different ports, all 1000 interfaces creation will happen one after the other.)
6. May depend on external applications to distribute the load across the actors.

IFM Listeners

IFM listeners listen to data change events for different MD-SAL data-stores. On the NBI side it implements data change listeners for interface config data-store and the service-binding data store. On the SBI side IFM implements listeners for Topology and Inventory data-stores in opendaylight.

Interface Config change listener

Interface config change listener listens to ietf-interface/interfaces data node.

service-binding change listener

Interface config change listener listens to ietf-interface/interfaces data node.

Topology state change listener

Interface config change listener listens to ietf-interface/interfaces data node.

inventory state change listener

+++ this page is under construction +++

Dynamic Behavior

when a l2vlan interface is configured

1. Interface ConfigDS is populated
2. Interface DCN in InterfaceManager does the following :
   - Add interface-state entry for the new interface along with if-index generated
   - Add ingress flow entry
   - If it is a trunk VLAN, need to add the interface-state for all child interfaces, and add ingress flows for all child interfaces

when a tunnel interface is configured

1. Interface ConfigDS is populated
2. Interface DCN in InterfaceManager does the following :
   - Creates bridge interface entry in odl-interface-meta Config DS
   - **Add port to Bridge using OVSDB**
     - retrieves the bridge UUID corresponding to the interface and
     - populates the OVSDB Termination Point Datastore with the following information

```
tpAugmentationBuilder.setName(portName);
tpAugmentationBuilder.setInterfaceType(type);
options.put(“key”, “flow”);
options.put(“local_ip”, localIp.getIpv4Address().getValue());
options.put(“remote_ip”, remoteIp.getIpv4Address().getValue());
tpAugmentationBuilder.setOptions(options);
```

OVSDB plugin acts upon this data change and configures the tunnel end points on the switch with the supplied information.

NodeConnector comes up on vSwitch

Inventory DCN Listener in InterfaceManager does the following:

1. Updates interface-state DS.
2. Generate if-index for the interface
3. Update if-index to interface reverse lookup map

4. If interface maps to a vlan trunk entity, operational states of all vlan trunk members are updated
5. If interface maps to tunnel entity, add ingress tunnel flow

Bridge is created on vSWitch

Topology DCN Listener in InterfaceManager does the following:

1. Update odl-interface-meta OperDS to have the dpid to bridge reference
2. Retrieve all pre provisioned bridge Interface Entries for this dpn, and add ports to bridge using ovsdb

ELAN/VPNManager does a bind service

1. Interface service-bindings config DS is populated with service name, priority and lport dispatcher flow instruction details
2. Based on the service priority, the higher priority service flow will go in dispatcher table with match as if-index
3. Lower priority service will go in the same lport dispatcher table with match as if-index and service priority

Interface Manager Sequence Diagrams

Following gallery contains sequence diagrams for different IFM operations -

Removal of Tunnel Interface When OF Switch is Connected

Removal of Tunnel Interfaces in Pre provisioning Mode

Updating of Tunnel Interfaces in Pre provisioning Mode

creation of tunnel-interface when OF switch is connected and PortName already in OperDS

creation of vlan interface in pre provisioning mode

creation of vlan interface when switch is connected

deletion of vlan interface in pre provisioning mode

deletion of vlan interface when switch is connect

Node connector added updated DCN handling

Node connector removed DCN handling

updation of vlan interface in pre provisioning mode

updation of vlan interface when switch is connect

Internal Transport Manager (ITM)

Internal Transport Manager creates and maintains mesh of tunnels of type VXLAN or GRE between Openflow switches forming an overlay transport network. ITM also builds external tunnels towards DC Gateway. ITM does not provide redundant tunnel support.

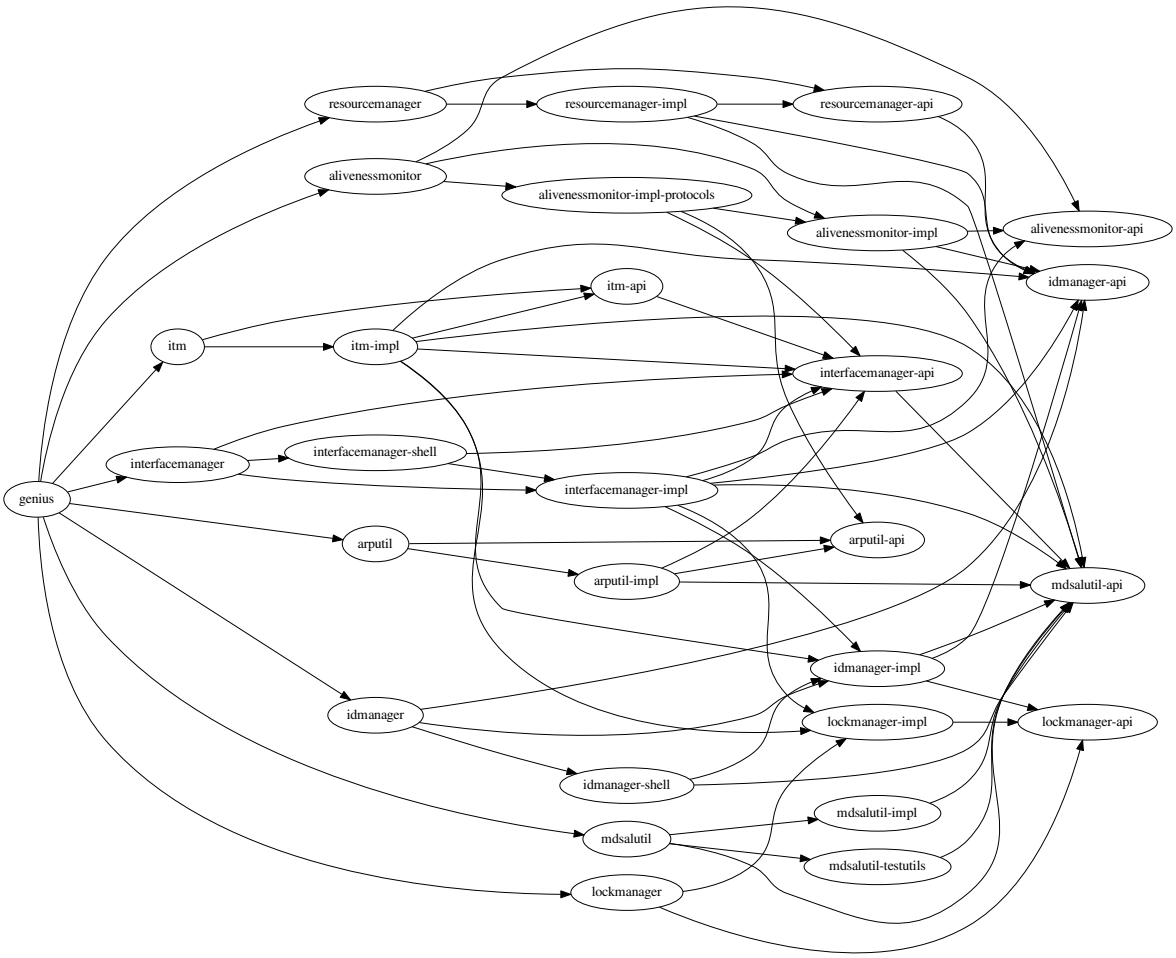
The diagram below gives a pictorial representation of the different modules and data stores and their interactions.

ITM Dependencies

ITM mainly interacts with following other genius modules-

1. **Interface Manager** – For creating tunnel interfaces
2. **Aliveness Monitor** - For monitoring the tunnel interfaces
3. **MdSalUtil** – For openflow operations

Following picture shows interface manager dependencies



Code Structure

As shown in the diagram, ITM has a common placeholder for various datastore listeners, RPC implementation, config helpers. Config helpers are responsible for creating / delete of Internal and external tunnel.

ITM Data Model

ITM uses the following data model to create and manage tunnel interfaces Tunnels interfces are created by writing to Interface Manager's Config DS.

itm.yang

follwoing datamodel is defined in itm.yang This DS stores the transport zone information populated through REST or Karaf CLI

|image33|

Itm-state.yang

This DS stores the tunnel end point information populated through REST or Karaf CLI. The internal and external tunnel interfaces are also stored here.

|image34|

Itm-rpc.yang

This Yang defines all the RPCs provided by ITM.

|image35|

Itm-config.yang

|image36|

ITM Design

ITM uses the datastore job coordinator module for all its operations.

When tunnel end point are configured in ITM datastores by CLI or REST, corresponding DTCNs are fired. ITM TransportZoneListener listens to the . Based on the add/remove end point operation, the transport zone listener queues the approporiate job (ItmInternalTunnelAddWorker or ItmInternalTunnelDeleteWorker) to the DataStoreJob Coordinator. Jobs within transport zones are queued to be executed serially and jobs across transport zones are done parallel.

Tunnel Building Logic

ITM will iterate over all the tunnel end points in each of the transport zones and build the tunnels between every pair of tunnel end points in the given transport zone. The type of the tunnel (GRE/VXLAN) will be indicated in the YANG model as part of the transport zone.

ITM Operations

ITM builds the tunnel infrastructure and maintains them. ITM builds two types of tunnels namely, internal tunnels between openflow switches and external tunnels between openflow switches and an external device such as datacenter gateway. These tunnels can be Vxlan or GRE. The tunnel endpoints are configured using either individual endpoint configuration or scheme based auto configuration method or REST. ITM will iterate over all the tunnel end points in each of the transport zones and build the tunnels between every pair of tunnel end points in the given transport zone.

- ITM creates tunnel interfaces in Interface manager Config DS.
- Stores the tunnel mesh information in tunnel end point format in ITM config DS
- ITM stores the internal and external trunk interface names in itm-state yang
- Creates external tunnels to DC Gateway when VPN manager calls the RPCs for creating tunnels towards DC gateway.

  ITM depends on interface manager for the following functionality.

- Provides interface to create tunnel interfaces

- Provides configuration option to enable monitoring on tunnel interfaces.
- Registers tunnel interfaces with monitoring enabled with alivenessmonitor.

  ITM depends on Aliveness monitor for the following functionality.
- Tunnel states for trunk interfaces are updated by alivenessmonitor. Sets OperState for tunnel interfaces

RPCs

The following are the RPCs supported by ITM

Get-tunnel-interface-id RPC

|image37|

Get-internal-or-external-interface-name

|image38|

Get-external-tunnel-interface-name

|image39|

Build-external-tunnel-from-dpns

|image40|

Add-external-tunnel-endpoint

|image41|

Remove-external-tunnel-from-dpns

|image42|

Remove-external-tunnel-endpoint

|image43|

Create-terminating-service-actions

|image44|

Remove-terminating-service-actions

|image45|

1. Aliveness Monitor
2. ID-Manager
3. MDSAL Utils
4. Resource Manager
5. FCAPS manager

3.6.3 Genius Design Specifications

Starting from Carbon, Genius uses RST format Design Specification document for all new features. These specifications are perfect way to understand various Genius features.

Contents:

Table of Contents

Title of the feature

[link to gerrit patch]

Brief introduction of the feature.

Problem description

Detailed description of the problem being solved by this feature

Use Cases

Use cases addressed by this feature.

Proposed change

Details of the proposed change.

Pipeline changes

Any changes to pipeline must be captured explicitly in this section.

Yang changes

This should detail any changes to yang models.

Configuration impact

Any configuration parameters being added/deprecated for this feature? What will be defaults for these? How will it impact existing deployments?

Note that outright deletion/modification of existing configuration is not allowed due to backward compatibility. They can only be deprecated and deleted in later release(s).

Clustering considerations

This should capture how clustering will be supported. This can include but not limited to use of CDTCL, EOS, Cluster Singleton etc.

Other Infra considerations

This should capture impact from/to different infra components like MDSAL Datastore, karaf, AAA etc.

Security considerations

Document any security related issues impacted by this feature.

Scale and Performance Impact

What are the potential scale and performance impacts of this change? Does it help improve scale and performance or make it worse?

Targeted Release

What release is this feature targeted for?

Alternatives

Alternatives considered and why they were not selected.

Usage

How will end user use this feature? Primary focus here is how this feature will be used in an actual deployment.

For most Genius features users will be other projects but this should still capture any user visible CLI/API etc. e.g. ITM configuration.

This section will be primary input for Test and Documentation teams. Along with above this should also capture REST API and CLI.

Features to Install

odl-genius-ui

Identify existing karaf feature to which this change applies and/or new karaf features being introduced. These can be user facing features which are added to integration/distribution or internal features to be used by other projects.

REST API

Sample JSONS/URIs. These will be an offshoot of yang changes. Capture these for User Guide, CSIT, etc.

CLI

Any CLI if being added.

Implementation

Assignee(s)

Who is implementing this feature? In case of multiple authors, designate a primary assignee and other contributors.

Primary assignee: <developer-a>

Other contributors: <developer-b> <developer-c>

Work Items

Break up work into individual items. This should be a checklist on Trello card for this feature. Give link to trello card or duplicate it.

Dependencies

Any dependencies being added/removed? Dependencies here refers to internal [other ODL projects] as well as external [OVS, karaf, JDK etc.] This should also capture specific versions if any of these dependencies. e.g. OVS version, Linux kernel version, JDK etc.

This should also capture impacts on existing project that depend on Genius. Following projects currently depend on Genius: * Netvirt * SFC

Testing

Capture details of testing that will need to be added.

Unit Tests

Integration Tests

CSIT

Documentation Impact

What is impact on documentation for this change? If documentation change is needed call out one of the <contributors> who will work with Project Documentation Lead to get the changes done.

Don't repeat details already discussed but do reference and call them out.

References

Add any useful references. Some examples:

- Links to Summit presentation, discussion etc.
- Links to mail list discussions
- Links to patches in other projects
- Links to external documentation

[1] OpenDaylight Documentation Guide

[2] https://specs.openstack.org/openstack/nova-specs/specs/kilo/template.html

---

Note: This template was derived from [2], and has been modified to support our project.

ITM Tunnel Auto-Configuration

https://git.opendaylight.org/gerrit/#/q/topic:itm-auto-config

Internal Transport Manager (ITM) Tunnel Auto configuration feature proposes a solution to migrate from REST/CLI based Tunnel End Point (TEP) configuration to automatic learning of Openvswitch (OVS) TEPs from the switches, thereby triggering automatic configuration of tunnels.

Problem description

User has to use ITM REST APIs for addition/deletion of TEPs into/from Transport zone. But, OVS and other TOR switches that support OVSDB can be configured for TEP without requring TEP configuration through REST API, which leads to redundancy and makes the process cumbersome and error-prone.

Use Cases

This feature will support following use cases:

- Use case 1: Add tep to existing transport-zone from southbound interface(SBI).
- Use case 2: Delete tep from SBI.
- Use case 3: Move the tep from one transport zone to another from SBI.
- Use case 4: User can specify the Datapath Node (DPN) bridge for tep other than `br-int` from SBI.
- Use case 5: Allow user to configure a tep from SBI if they want to use flow based tunnels.
- Use case 6: TEP-IP, Port, vlan, subnet, gateway IP are optional parameters for creating a transport zone from REST.
- Use case 7: User must configure Transport zone name and tunnel type parameters while creating a transport zone from REST, as both are mandatory parameters.
- Use case 8: Store teps received on OVS connect for transport-zone which is not yet created and also allow to move such teps into transport-zone when it gets created from northbound.

- Use case 9: Allow user to control creation of default transport zone through start-up configurable parameter `def-tz-enabled` in config file.
- Use case 10: Tunnel-type for default transport zone should be configurable through configurable parameter `def-tz-tunnel-type` in config file.
- Use case 11: Allow user to change `def-tz-enabled` configurable parameter from OFF to ON during OpenDaylight controller restart.
- Use case 12: Allow user to change `def-tz-enabled` configurable parameter from ON to OFF during OpenDaylight controller restart.
- Use case 13: Default value for configurable parameter `def-tz-enabled` is OFF and if it is not changed by user, then it will be OFF after OpenDaylight controller restart as well.

Following use cases will not be supported:

- If a switch gets disconnected, the corresponding TEP entries will not get cleared off from the ITM config datastore (DS) and operator must explicitly clean it up.
- Operator is not supposed to delete `default-transport-zone` from REST, such scenario will be taken as incorrect configuration.
- Dynamic change in the bridge for tunnel creation via change in Openvswitch table's external_ids parameter `br-name` is not supported.
- Dynamic change for `of-tunnel` tep configuration via change in Openvswitch table's external_ids parameter `of-tunnel` is not supported.
- Dynamic change for configurable parameters `def-tz-enabled` and `def-tz-tunnel-type` is not supported.

Proposed change

ITM will create a default transport zone on OpenDaylight start-up if configurable parameter `def-tz-enabled` is `true` in `genius-itm-config.xml` file (by default, this flag is false). When the flag is true, default transport zone is created and configured with:

- Default transport zone will be created with name `default-transport-zone`.
- Tunnel type: This would be configurable parameter via config file. ITM will take tunnel type value from config file for `default-transport-zone`. Tunnel-type value cannot be changed dynamically. It will take value of `def-tz-tunnel-type` parameter from config file `genius-itm-config.xml` on startup.
  - If `def-tz-tunnel-type` parameter is changed and `def-tz-enabled` remains `true` during OpenDaylight restart, then `default-transport-zone` with previous value of tunnel-type would be first removed and then `default-transport-zone` would be created with newer value of tunnel-type.

If `def-tz-enabled` is configured as `false`, then ITM will delete `default-transport-zone` if it is present already.

When transport-zone is added from northbound i.e. REST interface. Few of the transport-zone parameters are mandatory and fewer are optional now.

Status	Transport zone parameters
Mandatory	transport-zone name, tunnel-type
Optional	TEP IP-Address, Subnet prefix, Dpn-id, Gateway-ip, Vlan-id, Portname

When a new transport zone is created, check for any TEPs if present in `tepsNotHostedInTransportZone` for that transport zone. If present, remove from `tepsNotHostedInTransportZone` and add them under the transport zone and include the TEP in the tunnel mesh.

ITM will register listeners to the Node of network topology Operational DS to receive Data Tree Change Notification (DTCN) for add/update/delete notification in the OVSDB node so that such DTCN can be parsed and changes in the `external_ids` for TEP parameters can be determined to perform TEP add/update/delete operations.

URL: restconf/operational/network-topology:network-topology/topology/ovsdb:1

Sample JSON output


```
{
  "topology": [
    {
      "topology-id": "ovsdb:1",
      "node": [
      {
      "node-id": "ovsdb://uuid/83192e6c-488a-4f34-9197-d5a88676f04f",
      "ovsdb:db-version": "7.12.1",
      "ovsdb:ovs-version": "2.5.0",
      "ovsdb:openvswitch-external-ids": [
        {
          "external-id-key": "system-id",
          "external-id-value": "e93a266a-9399-4881-83ff-27094a648e2b"
        },
        {
          "external-id-key": "tep-ip",
          "external-id-value": "20.0.0.1"
        },
        {
          "external-id-key": "tzname",
          "external-id-value": "TZA"
        },
        {
          "external-id-key": "of-tunnel",
          "external-id-value": "true"
        }
      ],
      "ovsdb:datapath-type-entry": [
        {
          "datapath-type": "ovsdb:datapath-type-system"
        },
        {
          "datapath-type": "ovsdb:datapath-type-netdev"
        }
      ],
      "ovsdb:connection-info": {
        "remote-port": 45230,
        "local-ip": "10.111.222.10",
        "local-port": 6640,
        "remote-ip": "10.111.222.20"
      }

      ...
      ...

     }
    ]
   }
  ]
}
```

OVSDB changes

Below table covers how ITM TEP parameter are mapped with OVSDB and which fields of OVSDB would provide ITM TEP parameter values.

ITM TEP parameter	OVSDB field
DPN-ID	`ovsdb:datapath-id` from bridge whose name is pre-configured with openvswitch:external_ids:br-name:value
IP-Address	`openvswitch:external_ids:tep-ip`:value
Transport Zone Name	`openvswitch:external_ids:tzname`:value
of-tunnel	`openvswitch:external_ids:of-tunnel`:value

NOTE: If `openvswitch:external_ids:br-name` is not configured, then by default `br-int` will be considered to fetch DPN-ID which in turn would be used for tunnel creation.

MDSALUtil changes

`getDpnId()` method is added into MDSALUtil.java.

```
/**
 * This method will be utility method to convert bridge datapath ID from
 * string format to BigInteger format.
 *
 * @param datapathId datapath ID of bridge in string format
 *
 * @return the datapathId datapath ID of bridge in BigInteger format
 */
public static BigInteger getDpnId(String datapathId);
```

Pipeline changes

N.A.

Yang changes

Changes are needed in `itm.yang` and `itm-config.yang` which are described in below sub-sections.

itm.yang changes

Following changes are done in `itm.yang` file.

1. A new list `tepsNotHostedInTransportZone` will be added to container `transport-zones` for storing details of TEP received from southbound having transport zone which is not yet hosted from northbound.
2. Existing list `transport-zone` would be modified for leaf `zone-name` and `tunnel-type` to make them mandatory parameters.

Listing 3.1: itm.yang

```
list transport-zone {
    ordered-by user;
    key zone-name;
    leaf zone-name {
        type string;
        mandatory true;
    }
    leaf tunnel-type {
        type identityref {
            base odlif:tunnel-type-base;
        }
        mandatory true;
    }
}

list tepsNotHostedInTransportZone {
    key zone-name;
    leaf zone-name {
        type string;
    }
    list unknown-vteps {
        key "dpn-id";
        leaf dpn-id {
            type uint64;
        }
        leaf ip-address {
            type inet:ip-address;
        }
        leaf of-tunnel {
            description "Use flow based tunnels for remote-ip";
            type boolean;
            default false;
        }
    }
}
```

itm-config.yang changes

`itm-config.yang` file is modified to add new container to contain following parameters which can be configured in `genius-itm-config.xml` on OpenDaylight controller startup.

- `def-tz-enabled`: this is boolean type parameter which would create or delete `default-transport-zone` if it is configured true or false respectively. By default, value is `false`.
- `def-tz-tunnel-type`: this is string type parameter which would allow user to configure tunnel-type for `default-transport-zone`. By default, value is `vxlan`.

Listing 3.2: itm-config.yang

```
container itm-config {
   config true;
   leaf def-tz-enabled {
      type boolean;
      default false;
   }
```

```
    leaf def-tz-tunnel-type {
       type string;
       default "vxlan";
    }
 }
```

Workflow

TEP Addition

When TEP IP `external_ids:tep-ip` and `external_ids:tzname` are configured at OVS side using `ovs-vsctl` commands to add TEP, then TEP parameters details are passed to the OVSDB plugin via OVSDB connection which in turn, is updated into Network Topology Operational DS. ITM listens for change in Network Topology Node.

When TEP parameters (like `tep-ip`, `tzname`, `br-name`, `of-tunnel`) are received in add notification of OVSDB Node, then TEP is added.

For TEP addition, TEP-IP and DPN-ID are mandatory. TEP-IP is obtained from `tep-ip` TEP parameter and DPN-ID is fetched from OVSDB node based on `br-name` TEP parameter:

- if bridge name is specified, then datapath ID of the specified bridge is fetched.
- if bridge name is not specified, then datapath ID of the `br-int` bridge is fetched.

TEP-IP and fetched DPN-ID would be needed to add TEP in the transport-zone. Once TEP is added in config datastore, transport-zone listener of ITM would internally take care of creating tunnels on the bridge whose DPN-ID is passed for TEP addition. It is noted that TEP parameter `of-tunnel` would be checked if it is true, then `of-tunnel` flag would be set for vtep to be added under transport-zone or `tepsNotHostedInTransportZone`.

TEP would be added under transport zone with following conditions:

- TEPs not configured with `external_ids:tzname` i.e. without transport zone will be placed under the `default-transport-zone` if `def-tz-enabled` parameter is configured to true in `genius-itm-config.xml`. This will fire a DTCN to transport zone yang listener and ITM tunnels gets built.
- TEPs configured with `external_ids:tzname` i.e. with transport zone and if the specified transport zone exists in the ITM Config DS, then TEP will be placed under the specified transport zone. This will fire a DTCN to transport zone yang listener and the ITM tunnels gets built.
- TEPs configured with `external_ids:tzname` i.e. with transport zone and if the specified transport zone does not exist in the ITM Config DS, then TEP will be placed under the `tepsNotHostedInTransportZone` under ITM config DS.

TEP Movement

When transport zone which was not configured earlier, is created through REST, then it is checked whether any "orphan" TEPs already exists in the `tepsNotHostedInTransportZone` for the newly created transport zone, if present, then such TEPs are removed from `tepsNotHostedInTransportZone`, and then added under the newly created transport zone in ITM config DS and then TEPs are added to the tunnel mesh of that transport zone.

TEP Updation

- TEP updation for IP address is considered as TEP deletion followed by TEP addition. Remove existing TEP-IP `external_ids:tep-ip` and then add new TEP-IP using `ovs-vsctl` commands. TEP with old TEP-IP is deleted and then TEP with new TEP-IP gets added.
- TEP updation for transport zone can be done dynamically. When `external_ids:tzname` is updated at OVS side, then such change will be notified to OVSDB plugin via OVSDB protocol, which in turn is reflected in Network topology Operational DS. ITM gets DTCN for Node update. Parsing Node update notification for `external_ids:tzname` parameter in old and new node can determine change in transport zone for TEP. If it is updated, then TEP is deleted from old transport zone and added into new transport zone. This will fire a DTCN to transport zone yang listener and the ITM tunnels gets updated.

TEP Deletion

When an `openvswitch:external_ids:tep-ip` parameter gets deleted through *ovs-vsctl* command, then network topology Operational DS gets updated via OVSB update notification. ITM which has registered for the network-topology DTCNs, gets notified and this deletes the TEP from Transport zone or `tepsNotHostedInTransportZone` stored in ITM config DS based on `external_ids:tzname` parameter configured for TEP.

- If `external_ids:tzname` is configured and corresponding transport zone exists in Configuration DS, then remove TEP from transport zone. This will fire a DTCN to transport zone yang listener and the ITM tunnels of that TEP gets deleted.
- If `external_ids:tzname` is configured and corresponding transport zone does not exist in Configuration DS, then check if TEP exists in `tepsNotHostedInTransportZone`, if present, then remove TEP from `tepsNotHostedInTransportZone`.
- If `external_ids:tzname` is not configured, then check if TEP exists in the default transport zone in Configuration DS, if and only if `def-tz-enabled` parameter is configured to true in `genius-itm-config.xml`. In case, TEP is present, then remove TEP from `default-transport-zone`. This will fire a DTCN to transport zone yang listener and ITM tunnels of that TEP gets deleted.

Configuration impact

Following are the configuation changes and impact in the OpenDaylight.

- `genius-itm-config.xml` configuation file is introduced newly into ITM in which following parameters are added:
  - `def-tz-enabled`: this is boolean type parameter which would create or delete `default-transport-zone` if it is configured true or false respectively. Default value is `false`.
  - `def-tz-tunnel-type`: this is string type parameter which would allow user to configure tunnel-type for `default-transport-zone`. Default value is `vxlan`.

Listing 3.3: genius-itm-config.xml

```
<itm-config xmlns="urn:opendaylight:genius:itm:config">
    <def-tz-enabled>false</def-tz-enabled>
    <def-tz-tunnel-type>vxlan</def-tz-tunnel-type>
</itm-config>
```

Runtime changes to the parameters of this config file would not be taken into consideration.

Clustering considerations

Any clustering requirements are already addressed in ITM, no new requirements added as part of this feature.

Other Infra considerations

N.A.

Security considerations

N.A.

Scale and Performance Impact

This feature would not introduce any significant scale and performance issues in the OpenDaylight.

Targeted Release

OpenDaylight Carbon

Known Limitations

- Dummy Subnet prefix `255.255.255.255/32` under transport-zone is used to store the TEPs listened from southbound.

Alternatives

N.A.

Usage

Features to Install

This feature doesn't add any new karaf feature. This feature would be available in already existing `odl-genius` karaf feature.

REST API

Creating transport zone

As per this feature, the TEP addition is based on the southbound configuation and respective transport zone should be created on the controller to form the tunnel for the same. The REST API to create the transport zone with mandatory parameters.

URL: restconf/config/itm:transport-zones/

Sample JSON data

```
{
    "transport-zone": [
        {
            "zone-name": "TZA",
             "tunnel-type": "odl-interface:tunnel-type-vxlan"
        }
    ]
}
```

Retrieving transport zone

To retrieve the TEP configuations from all the transport zones.

URL: restconf/config/itm:transport-zones/

Sample JSON output

```
{
    "transport-zones": {
       "transport-zone": [
          {
            "zone-name": "default-transport-zone",
            "tunnel-type": "odl-interface:tunnel-type-vxlan"
          },
          {
            "zone-name": "TZA",
            "tunnel-type": "odl-interface:tunnel-type-vxlan",
            "subnets": [
              {
                "prefix": "255.255.255.255/32",
                "vteps": [
                  {
                    "dpn-id": 1,
                    "portname": "",
                    "ip-address": "10.0.0.1"
                  },
                  {
                    "dpn-id": 2,
                    "portname": "",
                    "ip-address": "10.0.0.2"
                  }
                ],
                "gateway-ip": "0.0.0.0",
                "vlan-id": 0
              }
            ]
          }
        ]
    }
}
```

CLI

No CLI is added into OpenDaylight for this feature.

OVS CLI

ITM TEP parameters can be added/removed to/from the OVS switch using the `ovs-vsctl` command:

```
DESCRIPTION
  ovs-vsctl
  Command for querying and configuring ovs-vswitchd by providing a
  high-level interface to its configuration database.
  Here, this command usage is shown to store TEP parameters into
  ``openvswitch`` table of OVS database.

SYNTAX
  ovs-vsctl  set O . [column]:[key]=[value]

* To set TEP params on OVS table:

ovs-vsctl    set O . external_ids:tep-ip=192.168.56.102
ovs-vsctl    set O . external_ids:tzname=TZA
ovs-vsctl    set O . external_ids:br-name=br0
ovs-vsctl    set O . external_ids:of-tunnel=true

* To clear TEP params in one go by clearing external_ids column from
  OVS table:

ovs-vsctl clear O . external_ids

* To clear specific TEP paramter from external_ids column in OVS table:

ovs-vsctl remove O . external_ids tep-ip
ovs-vsctl remove O . external_ids tzname

* To check TEP params are set or cleared on OVS table:

ovsdb-client dump -f list  Open_vSwitch
```

Implementation

Assignee(s)

Primary assignee:

- Tarun Thakur

Other contributors:

- Sathish Kumar B T
- Nishchya Gupta
- Jogeswar Reddy

Work Items

1. YANG changes
2. Add code to create xml config file for ITM to configure flag which would control creation of `default-transport-zone` during bootup and configure `tunnel-type` for default transport zone.

3. Add code to handle changes in the `def-tz-enabled` configurable parameter during OpenDaylight restart.
4. Add code to handle changes in the `def-tz-tunnel-type` configurable parameter during OpenDaylight restart.
5. Add code to create listener for OVSDB to receive TEP-specific parameters configured at OVS.
6. Add code to update configuation datastore to add/delete TEP received from southbound into transport-zone.
7. Check tunnel mesh for transport-zone is updated correctly for TEP add/delete into transport-zone.
8. Add code to update configuation datastore for handling update in TEP-IP.
9. Add code to update configuation datastore for handling update in TEP's transport-zone.
10. Check tunnel mesh is updated correctly against TEP update.
11. Add code to create `tepsNotHostedInTransportZone` list in configuation datastore to store TEP received with not-configured transport-zone.
12. Add code to move TEP from `tepsNotHostedInTransportZone` list to transport-zone configured from REST.
13. Check tunnel mesh is formed for TEPs after their movement from `tepsNotHostedInTransportZone` list to transport-zone.
14. Add UTs.
15. Add ITs.
16. Add CSIT.
17. Add Documentation.

Dependencies

This feature should be used when configuration flag i.e. `use-transport-zone` in `netvirt-neutronvpn-config.xml` for automatic tunnel configuration in transport-zone is disabled in Netvirt's NeutronVpn, otherwise netvirt feature of dynamic tunnel creation may duplicate tunnel for TEPs in the tunnel mesh.

Testing

Unit Tests

Appropriate UTs will be added for the new code coming in, once UT framework is in place.

Integration Tests

Integration tests will be added, once IT framework for ITM is ready.

CSIT

Following test cases will need to be added/expanded in Genius CSIT:

1. Verify `default-transport-zone` is not created when `def-tz-enabled` flag is false.
2. Verify tunnel-type change is considered while creation of `default-transport-zone`.

3. Verify ITM tunnel creation on default-transport-zone when TEPs are configured without transport zone or with `default-transport-zone` on switch when `def-tz-enabled` flag is true.
4. Verify `default-transport-zone` is deleted when `def-tz-enabled flag` is changed from true to false during OpenDaylight controller restart.
5. Verify ITM tunnel creation by TEPs configured with transport zone on switch and respective transport zone should be pre-configured on OpenDaylight controller.
6. Verify auto-mapping of TEPs to corresponding transport zone group.
7. Verify ITM tunnel deletion by deleting TEP from switch.
8. Verify TEP transport zone change from OVS will move the TEP to corresponding transport-zone in OpenDaylight controller.
9. Verify TEPs movement from `tepsNotHostedInTransportZone` to transport-zone when transport-zone is configured from northbound.
10. Verify ITM tunnel details persist after OpenDaylight controller restart, switch restart.

Documentation Impact

This will require changes to User Guide and Developer Guide.

User Guide will need to add information for below details:

- TEPs parameters to be configured from OVS side to use this feature.
- TEPs added from southbound can be viewed from REST APIs.
- TEPs added from southbound will be added under dummy subnet (255.255.255.255/32) in transport-zone.
- Usage details of genius-itm-config.xml config file for ITM to configure `def-tz-enabled` flag and `def-tz-tunnel-type` to create/delete `default-transport-zone` and its `tunnel-type` respectively.
- User is explicitly required to configure `def-tz-enabled` as true if TEPs needed to be added into `default-transport-zone` from northbound.

Developer Guide will need to capture how to use changes in ITM to create tunnel automatically for TEPs configured from southbound.

References

- Genius: Carbon Release Plan

Table of Contents

Load balancing and high availability of multiple VxLAN tunnels

https://git.opendaylight.org/gerrit/#/q/topic:vxlan-tunnel-aggregation

The purpose of this feature is to enable resiliency and load balancing of VxLAN encapsulated traffic between pair of OVS nodes.

Additionally, the feature will provide infrastructure to support more complex use cases such as policy-based path selection. The exact implementation of policy-based path selection is out of the scope of this document and will be described in a different spec [2].

Problem description

The current ITM implementation enables creation of a single VxLAN tunnel between each pair of hypervisors.

If the hypervisor is connected to the network using multiple links with different capacity or connected to different L2 networks in different subnets, it is not possible to utilize all the available network resources to increase the throughput of traffic to remote hypervisors.

In addition, link failure of the network card forwarding the VxLAN traffic will result in complete traffic loss to/from the remote hypervisor if the network card is not part of a bonded interface.

Use Cases

- Forwarding of VxLAN traffic between hypervisors with multiple network cards connected to L2 switches in different networks.
- Forwarding of VxLAN traffic between hypervisors with multiple network cards connected to the same L2 switch.

Proposed change

ITM Changes

The ITM will continue to create tunnels based on transport-zone configuration similarly to the current implementation - TEP IP per DPN per transport zone. When ITM creates TEP interfaces, in addition to creating the actual tunnels, it will create logical tunnel interface for each pair of DPNs in the `ietf-interface` config data-store representing the tunnel aggregation group between the DPNs. The logical tunnel interface be created only when the first tunnel interface on each OVS is created. In addition, this feature will be guarded by a global configuration option in the ITM and will be turned off by default. Only when the feature is enabled, the logical tunnel interfaces will be created.

Creation of transport-zone with multiple IPs per DPN is out of the scope of this document and will be described in [2] However, the limitation of configuring no more than one TEP ip per transport zone will remain.

The logical tunnel will reference all member tunnel interfaces in the group using `interface-child-info` model. In addition, it would be possible to add weight to each member of the group to support unequal load-sharing of traffic.

The proposed feature depends on egress tunnel service binding functionality detailed in [3].

When the logical tunnel interface is created, a default egress service would be bound to it. The egress service will create an OF select group based on the actual list of tunnel members in the logical group. Each tunnel member can be assigned a weight field that will be applied on it's corresponding bucket in the OF select group. If weight was not defined, the bucket weight will be configured with a default value of 1 resulting in uniform distribution if weight was not configured for any of the buckets. Each bucket in the select group will route the egress traffic to one of the tunnel members in the group by loading the lport-tag of the tunnel member interface to NXM `register6`.

Logical tunnel egress service pipeline example:

```
cookie=0x6900000, duration=0.802s, table=220, n_packets=0, n_bytes=0, priority=6,
 ↪reg6=0x500
actions=load:0xe000500->NXM_NX_REG6[],write_metadata:0xe000500000000000/
 ↪0xfffffffffffffffe,group:80000
cookie=0x8000007, duration=0.546s, table=220, n_packets=0, n_bytes=0, priority=7,
 ↪reg6=0x600 actions=output:3
cookie=0x8000007, duration=0.546s, table=220, n_packets=0, n_bytes=0, priority=7,
 ↪reg6=0x700 actions=output:4
cookie=0x8000007, duration=0.546s, table=220, n_packets=0, n_bytes=0, priority=7,
 ↪reg6=0x800 actions=output:5
group_id=800000,type=select,
bucket=weight:50,watch_port=3,actions=load:0x600->NXM_NX_REG6[],resubmit(,220),
```

```
bucket=weight:25,watch_port=4,actions=load:0x700->NXM_NX_REG6[],resubmit(,220),
bucket=weight:25,watch_port=5,actions=load:0x800->NXM_NX_REG6[],resubmit(,220)
```

Each bucket of the LB group will set the `watch_port` property to be the tunnel member OF port number. This will allow the OVS to monitor the bucket liveness and route egress traffic only to live buckets.

BFD monitoring is required to probe the tunnel state and update the OF select group accordingly. Using OF tunnels [4] or turning off BFD monitoring will not allow the logical group service to respond to tunnel state changes.

OF select group for logical tunnel can contain a mix of IPv4 and IPv6 tunnels, depending on the transport-zone configuration.

A new pool will be allocated to generate OF group ids of the default select group and the policy groups described in [2]. The pool name `VXLAN_GROUP_POOL` will allocate ids from the id-manager in the range 300,000-310,000. ITM RPC calls to get internal tunnel interface between source and destination DPNs will return the logical tunnel interface group name if such exits, otherwise the lower layer tunnel will be returned.

IFM Changes

The logical tunnel group is an `ietf-interface` thus it has an allocated lport-tag. RPC call to `getEgressActionsForInterface` for the logical tunnel will load `register6` with its corresponding lport-tag and resubmit the traffic to the egress dispatcher table.

The state of the logical tunnel group is affected by the states of the group members. If at least one of the tunnels is in `oper-status` UP, the logical group is considered UP.

If the logical tunnel was set as `admin-status` DOWN, all the tunnel members will be set accordingly.

Ingress traffic from VxLAN tunnels would not be bounded to any logical group service as part of this feature and it will continue to use the same workflow while traversing the ingress services pipeline.

Other applications would be able to utilize this infrastructure to introduce new services over logical tunnel group interface e.g. policy-based path selection. These services will take precedence over the default egress service for logical tunnel.

Netvirt Changes

L3 models map each combination of VRF id and destination prefix to a list of nexthop ip addresses. When calling `getInternalOrExternalInterfaceName` RPC from the FIB manager, if the DPN id of the remote nexthop is known it will be sent along with the nexthop ip. If logical tunnel exists between the source and destination DPNs it will be set as the lport-tag of `register6` in the remote nexthop actions.

Pipeline changes

For the flows below it is assumed that a logical tunnel group was configured for both ingress and egress DPNs. The logical tunnel group is composed of { `tunnnel1`, `tunnel2` } and bound to the default logical tunnel egress service.

Traffic between VMs on the same DPN

No pipeline changes required

L3 traffic between VMs on different DPNs

VM originating the traffic (Ingress DPN):

- Remote next hop group in the FIB table references the logical tunnel group.
- The default logical group service uses OF select group to load balance traffic between the tunnels.

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id,dst-ip=vm2-ip set dst-mac=vm2-mac tun-id=vm2-label reg6=logical-tun-lport-tag` =>
Egress table (220) `match: reg6=logical-tun-lport-tag` =>
Logical tunnel LB select group `set reg6=tun1-lport-tag` =>
Egress table (220) `match: reg6=tun1-lport-tag` output to `tunnel1`

VM receiving the traffic (Ingress DPN):

- No pipeline changes required

Classifier table (0) =>
Internal tunnel Table (36) `match:tun-id=vm2-label` =>
Local Next-Hop group: `set dst-mac=vm2-mac,reg6=vm2-lport-tag` =>
Egress table (220) `match: reg6=vm2-lport-tag` output to VM 2

SNAT traffic from non-NAPT switch

VM originating the traffic is non-NAPT switch:

- NAPT group references the logical tunnel group.

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id` =>
Pre SNAT table (26) `match: vpn-id=router-id` =>
NAPT Group `set tun-id=router-id reg6=logical-tun-lport-tag` =>
Egress table (220) `match: reg6=logical-tun-lport-tag` =>
Logical tunnel LB select group `set reg6=tun1-lport-tag` =>

Egress table (220) `match: reg6=tun1-lport-tag` output to `tunnel1`

Traffic from NAPT switch punted to controller:

- No explicit pipeline changes required

Classifier table (0) =>
Internal tunnel Table (36) `match:tun-id=router-id` =>
Outbound NAPT table (46) `set vpn-id=router-id, punt-to-controller`

L2 unicast traffic between VMs in different DPNs

VM originating the traffic (Ingress DPN):

- ELAN DMAC table references the logical tunnel group

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) =>
Dispatcher table (17) `l2vpn service: set elan-tag=vxlan-net-tag` =>
ELAN base table (48) =>
ELAN SMAC table (50) `match: elan-tag=vxlan-net-tag,src-mac=vm1-mac` =>
ELAN DMAC table (51) `match: elan-tag=vxlan-net-tag,dst-mac=vm2-mac set tun-id=vm2-lport-tag reg6=logical-tun-lport-tag` =>
Egress table (220) `match: reg6=logical-tun-lport-tag` =>
Logical tunnel LB select group `set reg6=tun2-lport-tag` =>
Egress table (220) `match: reg6=tun2-lport-tag` output to `tunnel2`

VM receiving the traffic (Ingress DPN):

- No explicit pipeline changes required

Classifier table (0) =>
Internal tunnel Table (36) `match:tun-id=vm2-lport-tag set reg6=vm2-lport-tag` =>
Egress table (220) `match: reg6=vm2-lport-tag` output to VM 2

L2 multicast traffic between VMs in different DPNs

VM originating the traffic (Ingress DPN):

- ELAN broadcast group references the logical tunnel group.

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) =>
Dispatcher table (17) `l2vpn service: set elan-tag=vxlan-net-tag` =>
ELAN base table (48) =>
ELAN SMAC table (50) `match: elan-tag=vxlan-net-tag,src-mac=vm1-mac` =>
ELAN DMAC table (51) =>
ELAN DMAC table (52) `match: elan-tag=vxlan-net-tag` =>
ELAN BC group `goto_group=elan-local-group, set tun-id=vxlan-net-tag reg6=logical-tun-lport-tag` =>
Egress table (220) `match: reg6=logical-tun-lport-tag` =>
Logical tunnel LB select group `set reg6=tun1-lport-tag` =>
Egress table (220) `match: reg6=tun1-lport-tag` output to `tunnel1`

VM receiving the traffic (Ingress DPN):

- No explicit pipeline changes required

Classifier table (0) =>
Internal tunnel Table (36) `match:tun-id=vxlan-net-tag` =>
ELAN local BC group `set tun-id=vm2-lport-tag` =>
ELAN filter equal table (55) `match: tun-id=vm2-lport-tag set reg6=vm2-lport-tag` =>
Egress table (220) `match: reg6=vm2-lport-tag` output to VM 2

Yang changes

The following changes would be required to support configuration of logical tunnel group:

IFM Yang Changes

Add a new tunnel type to represent the logical group in `odl-interface.yang`.

```
identity tunnel-type-logical-group {
    description "Aggregation of multiple tunnel endpoints between two DPNs";
    base tunnel-type-base;
}
```

Each tunnel member in the logical group can have an assigned weight as part of `tunnel-optional-params` in `odl-interface:if-tunnel` augment to support unequal load sharing.

```
 grouping tunnel-optional-params {
     leaf tunnel-source-ip-flow {
         type boolean;
         default false;
     }

     leaf tunnel-remote-ip-flow {
         type boolean;
```

```
         default false;
     }

     leaf weight {
        type uint16;
     }

     ...
 }
```

ITM Yang Changes

Each tunnel endpoint in `itm:transport-zones/transport-zone` can be configured with optional weight parameter. Weight configuration will be propagated to `tunnel-optional-params`.

```
 list vteps {
      key "dpn-id portname";
      leaf dpn-id {
          type uint64;
      }

      leaf portname {
           type string;
      }

      leaf ip-address {
           type inet:ip-address;
      }

      leaf weight {
           type unit16;
           default 1;
      }

      leaf option-of-tunnel {
           type boolean;
           default false;
      }
 }
```

The internal tunnel will be enhanced to contain multiple tunnel interfaces

```
container tunnel-list {
    list internal-tunnel {
        key "source-DPN destination-DPN transport-type";
        leaf source-DPN {
            type uint64;
        }

        leaf destination-DPN {
            type uint64;
        }

        leaf transport-type {
            type identityref {
                base odlif:tunnel-type-base;
```

```
            }
        }

        leaf-list tunnel-interface-name {
             type string;
        }
    }
}
```

The RPC call `itm-rpc:get-internal-or-external-interface-name` will be enhanced to contain the destination dp-id as an optional input parameter

```
rpc get-internal-or-external-interface-name {
    input {
         leaf source-dpid {
              type uint64;
         }

         leaf destination-dpid {
              type uint64;
         }

         leaf destination-ip {
              type inet:ip-address;
         }

         leaf tunnel-type {
             type identityref {
                  base odlif:tunnel-type-base;
             }
         }
    }

    output {
         leaf interface-name {
              type string;
         }
    }
}
```

Configuration impact

Creation of logical tunnel group will be guarded by configuration in `itm-config` per tunnel-type

```
container itm-config {
   config true;
   leaf def-tz-enabled {
      type boolean;
      default false;
   }

   leaf def-tz-tunnel-type {
      type string;
      default "vxlan";
   }
```

```
    list tunnel-aggregation {
       key "tunnel-type";
       leaf tunnel-type {
           type string;
       }

       leaf enabled {
           type boolean;
           default false;
       }
    }
}
```

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

This feature is expected to increase the datapath throughput by utilizing all available network resources.

Targeted Release

Carbon

Alternatives

There are certain use cases where it would be possible to add the network cards to a separate bridge with LACP enabled and patch it to br-int but this alternative was rejected since it imposes limitations on the type of links and the overall capacity.

Usage

Features to Install

This feature doesn't add any new karaf feature.

REST API

Create multiple uplinks between pair of OVS nodes

URL: restconf/config/itm:transport-zones/

Sample JSON data

The following REST will create 3 bi-directional tunnels between two OVS nodes.

```
{
   "transport-zone": [
    {
        "zone-name": "underlay-net1",
        "subnets": [
        {
          "prefix": "0.0.0.0/0",
          "vteps": [
            {
              "dpn-id": 273348439543366,
              "portname": "tunnel_port",
              "ip-address": "20.2.1.2",
              "option-of-tunnel": false
            },
            {
              "dpn-id": 110400932149974,
              "portname": "tunnel_port",
              "ip-address": "20.2.1.3",
              "option-of-tunnel": false
            }
          ],
          "gateway-ip": "0.0.0.0",
          "vlan-id": 0
        }
       ],
      "tunnel-type": "odl-interface:tunnel-type-vxlan"
    },
    {
        "zone-name": "underlay-net2",
        "subnets": [
        {
          "prefix": "0.0.0.0/0",
          "vteps": [
            {
              "dpn-id": 273348439543366,
              "portname": "tunnel_port",
              "ip-address": "30.3.1.2",
              "option-of-tunnel": false
            },
            {
              "dpn-id": 110400932149974,
              "portname": "tunnel_port",
              "ip-address": "30.3.1.3",
              "option-of-tunnel": false
            }
          ],
          "gateway-ip": "0.0.0.0",
          "vlan-id": 0
```

```
        }
       ],
      "tunnel-type": "odl-interface:tunnel-type-vxlan"
    },
   {
        "zone-name": "underlay-net3",
        "subnets": [
        {
          "prefix": "0.0.0.0/0",
          "vteps": [
            {
              "dpn-id": 273348439543366,
              "portname": "tunnel_port",
              "ip-address": "40.4.1.2",
              "option-of-tunnel": false
            },
            {
              "dpn-id": 110400932149974,
              "portname": "tunnel_port",
              "ip-address": "40.4.1.3",
              "option-of-tunnel": false
            }
          ],
          "gateway-ip": "0.0.0.0",
          "vlan-id": 0
        }
       ],
      "tunnel-type": "odl-interface:tunnel-type-vxlan"
    }
  ]
 }
```

ITM RPCs

URL: restconf/operations/itm-rpc:get-tunnel-interface-name

```
{
   "input": {
       "source-dpid": "40146672641571",
       "destination-dpid": "102093507130250",
       "tunnel-type": "odl-interface:tunnel-type-vxlan"
   }
}
```

URL: restconf/operations/itm-rpc:get-internal-or-external-interface-name

```
{
   "input": {
       "source-dpid": "40146672641571",
       "destination-dpid": "102093507130250",
       "tunnel-type": "odl-interface:tunnel-type-vxlan"
   }
}
```

CLI

`tep:show-state` will be enhanced to extract the state of the logical tunnel interface in addition to the actual TEP state.

Implementation

Assignee(s)

Primary assignee: Olga Schukin <olga.schukin@hpe.com>

Other contributors: Tali Ben-Meir <tali@hpe.com>

Work Items

Trello card: https://trello.com/c/Q7LgiHH7/92-multiple-vxlan-endpoints-for-compute

- Add support to ITM for creation of multiple tunnels between pair of DPNs
- Create logical tunnel group in `ietf-interface` if more than one tunnel exist between two DPNs. Update the `interface-child-info` model with the list of individual tunnel members
- Bind a default service for the logical tunnel interface to create OF select group based on the tunnel members
- Change ITM RPC calls to `getTunnelInterfaceName` and `getInternalOrExternalInterfaceName` to prefer the logical tunnel group over the tunnel members
- Support OF weighted select group

Dependencies

None

Testing

Unit Tests

- ITM unitests will be enhanced with test cases of multiple tunnels
- IFM unitests will be enhanced to handle CRUD operations on logical tunnel group

Integration Tests

CSIT

Transport zone creation with multiple tunnels

- Verify tunnel endpoint creation
- Verify logical tunnel group creation
- Verify logical tunnel service binding flows/group

Transport zone removal with multiple tunnels

- Verify tunnel endpoint removal
- Verify logical tunnel group removal
- Verify logical tunnel service binding flows/group removal

Transport zone updates to single/multiple tunnels

- Verify tunnel endpoint creation/removal
- Verify logical tunnel group creation/removal
- Verify logical tunnel service binding flows/group creation/removal

Transport zone creation with multiple OF tunnels

- Verify tunnel endpoint creation
- Verify logical tunnel group creation
- Verify logical tunnel service binding flows/group

Documentation Impact

None

References

[1] OpenDaylight Documentation Guide

[2] Policy based path selection

[3] Service Binding On Tunnels

[4] OF tunnels

Table of Contents

OF Tunnels

https://git.opendaylight.org/gerrit/#/q/topic:of-tunnels

OF Tunnels feature adds support for flow based tunnels to allow scalable overlay tunnels.

Problem description

Today when tunnel interfaces are created, InterFaceManager [IFM] creates one OVS port for each tunnel interface i.e. source-destination pair. For N devices in a TransportZone this translates to N*(N-1) tunnel ports created across all devices and N-1 ports in each device. This has obvious scale limitations.

Use Cases

This feature will support following use cases:

- Use case 1: Allow user to specify if they want to use flow based tunnels at the time of configuration.

- Use case 2: Create single OVS Tunnel Interface if flow based tunnels are configured and this is the first tunnel on this device/tep.
- Use case 3: Flow based and non flow based tunnels should be able to exist in a given transport zone.
- Use case 4: On tep delete, if this is the last tunnel interface on this tep/device and it is flow based tunnel, delete the OVS Tunnel Interface.

Following use cases will not be supported:

- Configuration of flow based and non-flow based tunnels of same type on the same device. OVS requires one of the following: `remote_ip`, `local_ip`, `type` and `key` to be unique. Currently we don't support multiple local_ip and key is always set to flow. So `remote_ip` and `type` are the only unique identifiers. `remote_ip=flow` is a super set of `remote_ip=<fixed-ip>` and we can't have two interfaces with all other fields same except this.
- Changing tunnel from one flow based to non-flow based at runtime. Such a change will require deletion and addition of tep. This is inline with existing model where tunnel-type cannot be changed at runtime.
- Configuration of Source IP for tunnel through flow. It will still be fixed. Though we're adding option in IFM YANG for this, implementation for it won't be done till we get use case(s) for it.

Proposed change

`OVS 2.0.0` onwards allows configuration of flow based tunnels through interface `option:remote_ip=flow`. Currently this field is set to IP address of the destination endpoint.

`remote_ip=flow` means tunnel destination IP will be set by an OpenFlow action. This allows us to add different actions for different destinations using the single OVS/OF port.

This change will add optional parameters to ITM and IFM YANG files to allow OF Tunnels. Based on this option, ITM will configure IFM which in turn will create tunnel ports in OVSDB.

Using OVSDB Plugin

OVSDB Plugin provides following field in Interface to configure options:

Listing 3.4: ovsdb.yang

```
list options {
    description "Port/Interface related optional input values";
    key "option";
    leaf option {
        description "Option name";
        type string;
    }
    leaf value {
        description "Option value";
        type string;
    }
```

For flow based tunnels we will set option name `remote_ip` to value `flow`.

MDSALUtil changes

Following new actions will be added to `mdsalutil/ActionType.java`

- set_tunnel_src_ip
- set_tunnel_dest_ip

Following new matches will be added to `mdsalutil/NxMatchFieldType.java`

- tun_src_ip
- tun_dest_ip

Pipeline changes

This change adds a new match in **Table0**. Today we match in `in_port` to determine which tunnel interface this pkt came in on. Since currently each tunnel maps to a source-destination pair it tells us about source device. For interfaces configured to use flow based tunnels this will add an additional match for `tun_src_ip`. So, `in_port+tunnel_src_ip` will give us which tunnel interface this pkt belongs to.

When services call `getEgressActions(), they will get one additional action, ``set_tunnel_dest_ip` before the `output:ofport` action.

YANG changes

Changes will be needed in `itm.yang` and `odl-interface.yang` to allow configuring a tunnel as flow based or not.

ITM YANG changes

A new parameter `option-of-tunnel` will be added to `list-vteps`

Listing 3.5: itm.yang

```
list vteps {
    key "dpn-id portname";
    leaf dpn-id {
        type uint64;
    }
    leaf portname {
        type string;
    }
    leaf ip-address {
        type inet:ip-address;
    }
    leaf option-of-tunnel {
        type boolean;
        default false;
    }
}
```

Same parameter will also be added to `tunnel-end-points` in `itm-state.yang`. This will help eliminate need to retrieve information from TransportZones when configuring tunnel interfaces.

Listing 3.6: itm-state.yang

```
list tunnel-end-points {
    ordered-by user;
    key "portname VLAN-ID ip-address tunnel-type";
```

```
    /* Multiple tunnels on the same physical port but on different VLAN can be␣
↪supported */

    leaf portname {
        type string;
    }
    ...
    ...
    leaf option-of-tunnel {
        type boolean;
        default false;
    }
}
```

This will allow to set OF Tunnels on per VTEP basis. So in a transport-zone we can have some VTEPs (devices) that use OF Tunnels and others that don't. Default of false means it will not impact existing behavior and will need to be explicitly configured. Going forward we can choose to set default true.

IFM YANG changes

We'll add a new `tunnel-optional-params` and add them to `iftunnel`

Listing 3.7: odl-interface.yang

```
grouping tunnel-optional-params {
    leaf tunnel-source-ip-flow {
        type boolean;
        default false;
    }

    leaf tunnel-remote-ip-flow {
        type boolean;
        default false;
    }

    list tunnel-options {
        key "tunnel-option";
        leaf tunnel-option {
            description "Tunnel Option name";
            type string;
        }
        leaf value {
            description "Option value";
            type string;
        }
    }
}
```

The `list tunnel-options` is a list of key-value pairs of strings, similar to options in OVSDB Plugin. These are not needed for OF Tunnels but is being added to allow user to configure any other Interface options that OVS supports. Aim is to enable developers and users try out newer options supported by OVS without needing to add explicit support for it. Note that there is no counterpart for this option in `itm.yang`. Any options that we want to explicitly support will be added as a separate option. This will allow us to do better validations for options that are needed for our specific use cases.

```
augment "/if:interfaces/if:interface" {
    ext:augment-identifier "if-tunnel";
    when "if:type = 'ianaift:tunnel'";
    ...
    ...
    uses tunnel-optional-params;
    uses monitor-params;
}
```

Workflow

Adding tep

1. User: While adding tep user gives `option-of-tunnel:true` for tep being added.
2. ITM: When creating tunnel interfaces for this tep, if `option-of-tunnel:true`, set `tunnel-remote-ip:true` for the tunnel interface.
3. IFM: If `option-of-tunnel:true` and this is first tunne on this device, set `option:remote_ip=flow` when creating tunnel interface in OVSDB. Else, set `option:remote_ip=<destination-ip>`.

Deleting tep

1. If `tunnel-remote-ip:true` and this is *last* tunnel on this device, delete tunnel port in OVSDB. Else, do nothing.
2. If `tunnel-remote-ip:false`, follow existing logic.

Configuration impact

This change doesn't add or modify any configuration parameters.

Clustering considerations

Any clustering requirements are already addressed in ITM and IFM, no new requirements added as part of this feature.

Other Infra considerations

N.A.

Security considerations

N.A.

Scale and Performance Impact

This solution will help improve scale numbers by reducing no. of interfaces created on devices as well as no. of interfaces and ports present in `inventory` and `network-topology`.

Targeted Release(s)

Carbon. Boron-SR3.

Known Limitations

BFD monitoring will not work when OF Tunnels are used. Today BFD monitoring in OVS relies on destination_ip configured in remote_ip when creating tunnel port to determine target IP for BFD packets. If we use `flow` it won't know where to send BFD packets. Unless OVS allows adding destination IP for BFD monitoring on such tunnels, monitoring cannot be enabled.

Alternatives

LLDP/ARP based monitoring was considered for OF tunnels to overcome lack of BFD monitoring but was rejected because LLDP/ARP based monitoring doesn't scale well. Since driving requirement for this feature is scale setups, it didn't make sense to use an unscalable solution for monitoring.

XML/CFG file based global knob to enable OF tunnels for all tunnel interfaces was rejected due to inflexible nature of such a solution. Current solution allows a more fine grained and device based configuration at runtime. Also, wanted to avoid adding yet another global configuration knob.

Usage

Features to Install

This feature doesn't add any new karaf feature.

REST API

Adding TEPs to transport zone

For most users TEP Addition is the only configuration they need to do to create tunnels using genius. The REST API to add TEPs with OF Tunnels is same as earlier with one small addition.

URL: restconf/config/itm:transport-zones/

Sample JSON data

```
{
 "transport-zone": [
     {
         "zone-name": "TZA",
         "subnets": [
             {
                 "prefix": "192.168.56.0/24",
                 "vlan-id": 0,
                 "vteps": [
                     {
                         "dpn-id": "1",
                         "portname": "eth2",
                         "ip-address": "192.168.56.101",
                         "option-of-tunnel":"true"
```

```
                    }
                ],
                "gateway-ip": "0.0.0.0"
            }
        ],
        "tunnel-type": "odl-interface:tunnel-type-vxlan"
    }
 ]
}
```

Creating tunnel-interface directly in IFM

This use case is mainly for those who want to write applications using Genius and/or want to create individual tunnel interfaces. Note that this is a simpler easy way to create tunnels without needing to delve into how OVSDB Plugin creates tunnels.

Refer Genius User Guide for more details on this.

URL: restconf/config/ietf-interfaces:interfaces

Sample JSON data

```
{
 "interfaces": {
 "interface": [
     {
         "name": "vxlan_tunnel",
         "type": "iana-if-type:tunnel",
         "odl-interface:tunnel-interface-type": "odl-interface:tunnel-type-vxlan",
         "odl-interface:datapath-node-identifier": "1",
         "odl-interface:tunnel-source": "192.168.56.101",
         "odl-interface:tunnel-destination": "192.168.56.102",
         "odl-interface:tunnel-remote-ip-flow": "true",
         "odl-interface:monitor-enabled": false,
         "odl-interface:monitor-interval": 10000,
         "enabled": true
     }
  ]
 }
}
```

CLI

A new boolean option, `remoteIpFlow` will be added to `tep:add` command.

```
DESCRIPTION
  tep:add
  adding a tunnel end point

SYNTAX
  tep:add [dpnId] [portNo] [vlanId] [ipAddress] [subnetMask] [gatewayIp]␣
↪[transportZone]
  [remoteIpFlow]

ARGUMENTS
```

```
dpnId
        DPN-ID
portNo
        port-name
vlanId
        vlan-id
ipAddress
        ip-address
subnetMask
        subnet-Mask
gatewayIp
        gateway-ip
transportZone
        transport_zone
remoteIpFlow
        Use flow for remote ip
```

Implementation

Assignee(s)

Primary assignee: <Vishal Thapar>

Other contributors: <Vacancies available>

Work Items

1. YANG changes
2. Add relevant match and actions to MDSALUtil
3. Add `set_tunnel_dest_ip` action to actions returned in `getEgressActions()` for OF Tunnels.
4. Add match on `tun_src_ip` in **Table0** for OF Tunnels.
5. Add CLI.
6. Add UTs.
7. Add ITs.
8. Add CSIT.
9. Add Documentation

Dependencies

This doesn't add any new dependencies. This requires minimum of `OVS 2.0.0` which is already lower than required by some of other features.

This change is backwards compatible, so no impact on dependent projects. Projects can choose to start using this when they want. However, there is a known limitation with monitoring, refer Limitations section for details.

Following projects currently depend on Genius:

- Netvirt
- SFC

Testing

Unit Tests

Appropriate UTs will be added for the new code coming in once framework is in place.

Integration Tests

Integration tests will be added once IT framework for ITM and IFM is ready.

CSIT

CSIT already has test cases for tunnels which test with non OF Tunnels. Similar test cases will be added for OF Tunnels. Alternatively, some of the existing test cases that use multiple teps can be tweaked to use OF Tunnels for one of them.

Following test cases will need to be added/expanded in Genius CSIT:

1. Create a TZ with more than one TEPs set to use OF Tunnels and test datapath.
2. Create a TZ with mix of OF and non OF Tunnels and test datapath.
3. Delete a TEP using OF Tunnels and add it again with non OF tunnels and test the datapath.
4. Delete a TEP using non OF Tunnels and add it again with OF Tunnels and test datapath.

Documentation Impact

This will require changes to User Guide and Developer Guide.

User Guide will need to add information on how to add TEPs with flow based tunnels.

Developer Guide will need to capture how to use changes in IFM to create individual tunnel interfaces.

References

- https://wiki.opendaylight.org/view/Genius:Carbon_Release_Plan

Table of Contents

Traffic shaping with Ovsdb QoS queues

QoS patches: https://git.opendaylight.org/gerrit/#/q/topic:qos-shaping

The current Boron implementation provides support for ingress rate limiting configuration of OVS. The Carbon release will add egress traffic shaping to QoS feature set. (Note, the direction of traffic flow (ingress, egress) is from the perspective of the OpenSwitch)

Problem description

OVS supports traffic shaping for traffic that egresses from a switch. To utilize this functionality, Genius implementation should be able to create 'set queue' output action upon connection of new OpenFlow node.

Use Cases

Use case 1: Allow Unimgr to shape egress traffic from UNI

Proposed change

Unimgr or Neutron VPN creates ietf vlan interface for each port connected to particular service. The Ovsdb provides a possibility to create QoS and mapped Queue with egress rate limits for lower level port. Such queue should be created on parent physical interface of vlan or trunk member port if service has definition of limits. The ovsdb southbound provides interface for creation of ovs QoS and Queues. This functionality may be utilized by netvirt qos service. Below is the dump from ovsdb with queues created for one of the ports.

```
Port table
   _uuid : a6cf4ca9-b15c-4090-aefe-23af2d5ce4f2
   name                : "ens5"
   qos                 : 9779ce41-4347-4383-b308-75f46d6a258c
QoS table
   _uuid               : 9779ce41-4347-4383-b308-75f46d6a258c
   other_config        : {max-rate="50000"}
   queues              : {1=3cc34bb7-7df8-4538-9fd7-4a6c6c467c69}
   type                : linux-htb
Queue table
   _uuid               : 3cc34bb7-7df8-4538-9fd7-4a6c6c467c69
   dscp                : []
   other_config        : {max-rate="50000", min-rate="5000"}
```

The queues creation is out of scope of this document. The definition of vlan or trunk member port will be augmented with relevant queue reference and number if queue was created successful. That will allow to create openflow 'set_queue' output action during service binding.

Pipeline changes

New 'set_queue' action will be supported in Egress Dispatcher table

Table	Match	Action
Egress Dispatcher [220]	no changes	Set queue id (optional) and output to port

Yang changes

A new augment "ovs-qos" is added to if:interface in odl-interface.yang

```
/* vlan port to qos queue */
 augment "/if:interfaces/if:interface" {
     ext:augment-identifier "ovs-qos";
     when "if:type = 'ianaift:l2vlan'";

     leaf ovs-qos-ref {
         type instance-identifier;
         description
           "represents whether service port has associated qos. A reference to a␣
↪ovsdb QoS entry";
     }
     leaf service-queue-number {
         type uint32;
         description
           "specific queue number within the list of queues in the qos entry";
     }
 }
```

Configuration impact

None

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

Additional OpenFlow action will be performed on part of the packages. Egress packages will be processed via linux-htp if service configured accordanly.

Targeted Release

Carbon

Alternatives

The unified REST API for ovsdb port adjustment could be created if future release. The QoS engress queues and ingress rate limiting should be a part of this API. Usage ===== User will configure unimgr service with egress rate limits. That will follow to process described above.

Features to Install

- odl-genius (unimgr using genius feature for flows creation)

REST API

None

CLI

None

Implementation

Assignee(s)

Primary assignee: konsta.pozdeev@hpe.com

Work Items

Dependencies

Minimum OVS version 1.8.0 is required.

Testing

Unimgr test cases with configured egress rate limits will cover this functionality.

Unit Tests

Integration Tests

CSIT

References

[1] *OpenDaylight Documentation Guide <http://docs.opendaylight.org/en/latest/documentation.html>*

[2] https://specs.openstack.org/openstack/nova-specs/specs/kilo/template.html

Table of Contents

Service Binding On Tunnels

https://git.opendaylight.org/gerrit/#/q/topic:service-binding-on-tunnels

Service Binding On Tunnels Feature enables applications to bind multiple services on an ingress/egress tunnel.

Problem description

Currently GENIUS does not provide a generic mechanism to support binding services on all interfaces.Ingress service binding pipeline is different for l2vlan interfaces and tunnel interfaces.Similarly, egress Service Binding is only supported for l2vlan interfaces.

Today when ingress services are bound on a tunnel, the highest priority service gets bound in `INTERFACE INGRESS TABLE(0)` itself, and remaining service entries get populated in `LPORT DISPATCHER TABLE(17)`, which is not in alignment with the service binding logic for VM ports. As part of this feature, we enable ingress/egress service binding support for tunnels in the same way as for VM interfaces. This feature also enables service-binding based on a tunnel-type which is basically meant for optimizing the number of flow entries in dispatcher tables.

Use Cases

This feature will support following use cases:

- Use case 1: IFM should support binding services based on tunnel type.
- Use case 2: All application traffic ingressing on a tunnel should go through the `LPORT DISPATCHER TABLE(17)`.
- Use case 3: IFM should support binding multiple ingress services on tunnels.
- Use case 4: IFM should support priority based ingress service handling for tunnels.
- Use case 5: IFM should support unbinding ingress services on tunnels.
- Use case 6: IFM should support binding multiple egress services on tunnels.
- Use case 7: IFM should support priority based egress service handling for tunnels.

- Use case 8: All application traffic egressing on a tunnel should go through the egress dispatcher table(220).
- Use case 9: Datapath should be intact even if there is no egress service bound on the tunnel.
- Use case 10: IFM should support unbinding egress services on tunnels.
- Use case 11: IFM should support handling of lower layer interface deletions gracefully.
- Use case 12: IFM should support binding services based on tunnel type and lport-tag on the same tunnel interface on a priority basis.
- Use case 13: Applications should bind on specific tunnel types on module startup
- Use case 13: IFM should take care of programming the tunnel type based binding flows on each DPN.

Following use cases will not be supported:

- Use case 1 : Update of service binding on tunnels. Any update should be done as delete and re-create

Proposed change

The proposed change extends the current l2vlan service binding functionality to tunnel interfaces. With this feature, multiple applications can bind their services on the same tunnel interface, and traffic will be processed on an application priority basis. Applications are given the flexibility to provide service specific actions while they bind their services. Normally service binding actions include *go-to-service-pipeline-entry-table*. Packets will enter a particular service based on the service priority, and if the packet is not consumed by the service, it is the application's responsibility to resubmit the packet back to the `egress/ingress dispatcher table` for further processing by next priority service. Egress Dispatcher Table will have a default service priority entry per tunnel interface to egress the packet on the tunnel port.So, if there are no egress services bound on a tunnel interface, this default entry will take care of taking the packet out of the switch.

The feature also enables service binding based on tunnel type. This way number of entries in Dispatcher Tables can be optimized if all the packets entering on tunnel of a particular type needs to be handled in the same way.

Pipeline changes

There is a pipeline change introduced as part of this feature for tunnel egress as well as ingress, and is captured in genius pipeline document patch[2].

With this feature, all traffic from INTERFACE_INGRESS_TABLE(0) will be dispatched to LPORT_DISPATCHER_TABLE(17), from where the packets will be dispatched to the respective applications on a priority basis.

Register6 will be used to set the ingress tunnel-type in Table0, and this can be used to match in Table17 to identify the respective applications bound on the tunnel-type. Remaining logic of ingress service binding will remain as is, and service-priority and interface-tag will be set in metadata as usual. The bits from 25-28 of Register6 will be used to indicate tunnel-type.

After the ingress service processing, packets which are identified to be egressed on tunnel interfaces, currently directly go to the tunnel port. With this feature, these packets will goto Egress Dispatcher Table[Table 220] first, where the packet will be processed by Egress Services on the tunnel interface one by one, and finally will egress the switch.

Register6 will be used to indicate service priority as well as interface tag for the egress tunnel interface, in Egress Dispatcher Table, and when there are N services bound on a tunnel interface, there will be N+1 entries in Egress Dispatcher Table, the additional one for the default tunnel entry. The first 4 bits of Register6 will be used to indicate the service priority and the next 20 bits for interface Tag, and this will be the match criteria for packet redirection to service pipeline in Egress Dispatcher Table. Before sending the packet to the service, Egress Dispatcher Table will set

[2] Netvirt Pipeline Diagram http://docs.opendaylight.org/en/latest/submodules/genius/docs/pipeline.html

the service index to the next service' priority. Same as ingress, Register6 will be used for egress tunnel-type matching, if there are services bound on tunnel-type.

TABLE	MATCH	ACTION
INTER-FACE_INGRESS_TABLE	in_port	SI=0,reg6=interface_type, metadata=lport tag, goto table 17
LPORT_DISPATCHER_TABLE	metadata=service priority && lport-tag(priority=10)	increment SI, apply service specific actions, goto ingress service
	reg6=tunnel-type priority=5	increment SI, apply service specific actions, goto ingress service
EGRESS_DISPATCHER_TABLE	Reg6==service Priority && lport-tag(priority=10)	increment SI, apply service specific actions, goto egress service
	reg6=tunnel-type priority=5	increment SI, apply service specific actions, goto egress service

RPC Changes

`GetEgressActionsForInterface` RPC in interface-manager currently returns the output:port action for tunnel interfaces. This will be changed to return set_field_reg6(default-service-index + interface-tag) and resubmit(egress_dispatcher_table).

Yang changes

No yang changes are needed, as binding on tunnel-type is enabled by having reserved keywords for interface-names

Workflow

Create Tunnel

1. User: User created a tunnel end point
2. IFM: When tunnel port is created on OVS, and the respective OpenFlow port Notification comes, IFM binds a default service in Egress Dispatcher Table for the tunnel interface, which will be the least priority service, and the action will be to take the packet out on the tunnel port.

Bind Service on Tunnel Interface

1. User: While binding service on tunnels user gives `service-priority`, `service-mode` and `instructions` for service being bound on the tunnel interface.
2. IFM: When binding the service for the tunnel, if this is the first service being bound, program flow rules in Dispatcher Table(ingress/egress based on service mode) to match on `service-priority` and `interface-tag` value with actions pointing to the service specific actions supplied by the application.
3. IFM: When binding a second service, based on the service priority one more flow will be created in Dispatcher Table with matches specific to the new service priority.

Unbind Service on Tunnel Interface

1. User: While unbinding service on tunnels user gives `service-priority` and `service-mode` for service being unbound on the tunnel interface.

2. IFM: When unbinding the service for the tunnel, IFM removes the entry in Dispatcher Tables for the service. IFM also rearranges the remaining flows for the same tunnel interface to adjust the missing service priority

Bind Service on Tunnel Type

1. Application: While binding service on tunnel type user gives a reserved keyword indicating the tunnel-type apart from‘‘service-priority‘‘, `service-mode` and `instructions` for service being bound. The reserved keywords will be `ALL_VXLAN_INTERNAL`, `ALL_VXLAN_EXTERNAL`, and `ALL_MPLS_OVER_GRE`.
2. IFM: When binding the service for the tunnel-type,program flow rules in Dispatcher Table(ingress/egress based on service mode) to match on `service-priority` and `tunnel-type` value with actions pointing to the service specific actions supplied by the application will be created on each DPN.
3. IFM: When binding a second service, based on the service priority one more flow will be created in Dispatcher Table with matches specific to the new service priority will be created on each DPN..

Unbind Service on Tunnel Type

1. User: While unbinding service on tunnels user gives a reserved keyword indicating the tunnel-type ,‘‘service-priority‘‘ and `service-mode` for service being unbound on all connected DPNs.
2. IFM: When unbinding the service for the tunnel-type, IFM removes the entry in Dispatcher Tables for the service. IFM also rearranges the remaining flows for the same tunnel type to adjust the missing service priority

Delete Tunnel

1. User: User deleted a tunnel end point
2. IFM: When tunnel port is deleted on OVS, and the respective OpenFlow Port Notification comes, IFM unbinds the default service in Egress Dispatcher Table for the tunnel interface.
3. IFM: If there are any outstanding services bound on the tunnel interface, all the Dispatcher Table Entries for this Tunnel will be deleted by IFM.

Application Module Startup

1. Applications: When Application bundle comes up, they can bind respective applications on the tunnel types they are interested in, with their respective service priorities.

Configuration impact

This change doesn’t add or modify any configuration parameters.

Clustering considerations

The solution is supported on a 3-node cluster.

Other Infra considerations

N.A.

Security considerations

N.A.

Scale and Performance Impact

- The feature adds one extra transaction during tunnel port creation, since the default Egress Dispatcher Table entry has to be programmed for each tunnel.
- The feature provides support for service-binding on tunnel type with the primary purpose of minimizing the number of flow entries in ingress/egress dispatcher tables.

Targeted Release

Carbon.

Alternatives

N/A

Usage

Features to Install

This feature doesn't add any new karaf feature.Installing any of the below features can enable the service:

odl-genius-ui odl-genius-rest odl-genius

REST API

Creating tunnel-interface directly in IFM

This use case is mainly for those who want to write applications using Genius and/or want to create individual tunnel interfaces. Note that this is a simpler easy way to create tunnels without needing to delve into how OVSDB Plugin creates tunnels.

Refer *Genius User Guide [4]_* for more details on this.

URL: restconf/config/ietf-interfaces:interfaces

Sample JSON data

```
{
 "interfaces": {
 "interface": [
     {
         "name": "vxlan_tunnel",
         "type": "iana-if-type:tunnel",
         "odl-interface:tunnel-interface-type": "odl-interface:tunnel-type-vxlan",
         "odl-interface:datapath-node-identifier": "1",
         "odl-interface:tunnel-source": "192.168.56.101",
```

```
          "odl-interface:tunnel-destination": "192.168.56.102",
          "odl-interface:monitor-enabled": false,
          "odl-interface:monitor-interval": 10000,
          "enabled": true
      }
  ]
 }
}
```

Binding Egress Service On Tunnels

URL: http://localhost:8181/restconf/config/interface-service-bindings:service-bindings/services-info/{tunnel-interface-name}/interface-service-bindings:service-mode-egress

Sample JSON data

```
{
   "bound-services": [
     {
       "service-name": "service1",
       "flow-priority": "5",
       "service-type": "service-type-flow-based",
       "instruction": [
        {
         "order": 1,
         "go-to-table": {
            "table_id": 88
          }
        }],
       "service-priority": "2",
       "flow-cookie": "1"
     }
   ]
}
```

CLI

N.A.

Implementation

Assignee(s)

Primary assignee: Faseela K

Work Items

1. Create Table 0 tunnel entries to set tunnel-type and lport_tag and point to `LPORT_DISPATCHER_TABLE`
2. Support of reserved keyword in interface-names for tunnel type based service binding.
3. Program tunnel-type based service binding flows on DPN connect events.

4. Program Lport Dispatcher Flows(17) on bind service
5. Remove Lport Dispatcher Flows(17) on unbind service
6. Handle multiple service bind/unbind on tunnel interface
7. Create default Egress Service for Tunnel on Tunnel Creation
8. Add `set_field_reg_6` and `resubmit(220)` action to actions returned in `getEgressActionsForInterface()` for Tunnels.
9. Program Egress Dispatcher Table(220) Flows on bind service
10. Remove Egress Dispatcher Table(220) Flows on unbind service
11. Handle multiple egress service bind/unbind on tunnel interface
12. Delete default Egress Service for Tunnel on Tunnel Deletion
13. Add UTs.
14. Add CSIT.
15. Add Documentation
16. Trello Card : https://trello.com/c/S8lNGd9S/6-service-binding-on-tunnel-interfaces

Dependencies

Genius, Netvirt

There will be several impacts on netvirt pipeline with this change. A brief overview is given in the table below:

Testing

Capture details of testing that will need to be added.

Unit Tests

New junits will be added to InterfaceManagerConfigurationTest to cover the following :

1. Bind/Unbind single ingress service on tunnel-type
2. Bind/Unbind single egress service on tunnel-type
3. Bind single ingress service on tunnel-interface
4. Unbind single ingress service on tunnel-interface
5. Bind multiple ingress services on tunnel in priority order
6. Unbind multiple ingress services on tunnel in priority order
7. Bind multiple ingress services out of priority order
8. Unbind multiple ingress services out of priority order
9. Delete tunnel port to check if ingress dispatcher flows for bound services get deleted
10. Add tunnel port back to check if ingress dispatcher flows for bound services get added back
11. Bind single egress service on tunnel

12. Unbind single egress service on tunnel
13. Bind multiple egress services on tunnel in priority order
14. Unbind multiple egress services on tunnel in priority order
15. Bind multiple egress services out of priority order
16. Unbind multiple egress services out of priority order
17. Delete tunnel port to check if egress dispatcher flows for bound services get deleted
18. Add tunnel port back to check if egress dispatcher flows for bound services get added back

Integration Tests

CSIT

The following TCs should be added to CSIT to cover this feature:

1. Bind/Unbind single ingress/egress service on tunnel-type to see the corresponding table entries are created in switch.
2. Bind single ingress service on tunnel to see the corresponding table entries are created in switch.
3. Unbind single ingress service on tunnel to see the corresponding table entries are deleted in switch.
4. Bind multiple ingress services on tunnel in priority order to see if metadata changes are proper on the flow table.
5. Unbind multiple ingress services on tunnel in priority order to see if metadata changes are proper on the flow table on each unbind.
6. Bind multiple ingress services out of priority order to see if metadata changes are proper on the flow table.
7. Unbind multiple ingress services out of priority order.
8. Delete tunnel port to check if ingress dispatcher flows for bound services get deleted.
9. Add tunnel port back to check if ingress dispatcher flows for bound services get added back.
10. Bind single egress service on tunnel to see the corresponding table entries are created in switch.
11. Unbind single egress service on tunnel to see the corresponding table entries are deleted in switch.
12. Bind multiple egress services on tunnel in priority order to see if metadata changes are proper on the flow table.
13. Unbind multiple egress services on tunnel in priority order to see if metadata changes are proper on the flow table on each unbind.
14. Bind multiple egress services out of priority order to see if metadata changes are proper on the flow table.
15. Unbind multiple egress services out of priority order.
16. Delete tunnel port to check if egress dispatcher flows for bound services get deleted.
17. Add tunnel port back to check if egress dispatcher flows for bound services get added back.

Documentation Impact

This will require changes to User Guide and Developer Guide.

There is a pipeline change for tunnel datapath introduced due to this change. This should go in User Guide.

Developer Guide should capture how to configure egress service binding on tunnels.

References

Note: This template was derived from [2], and has been modified to support our project.

Table of Contents

- *References*

Service Recovery Framework

https://git.opendaylight.org/gerrit/#/q/topic:service-recovery

Service Recovery Framework is a feature that enables recovery of services. This recovery can be trigerred by user, or eventually, be used as a self-healing mechanism.

Problem description

Status and Diagnostic adds support for reporting current status of different services. However, there is no means to recover individual service or service instances that have failed. Only recovery that can be done today is to restart the controller node(s) or manually restart the bundle or reinstall the karaf feature itself.

Restarting the controller can be overkill and needlessly disruptive. Manually restarting bundle or feature requires user to be aware of and have access to these CLIs. There may not be one-to-one mapping from a service to corresponding bundle or feature. Also, a truly secure system would provide role based access to users. Only someone with administrative rights will have access to Karaf CLI to restart/reinstall while a less privileged user should be able to trigger recovery without requiring higher level access.

Note that role based access is out of scope of this document

Use Cases

This feature will support following use cases:

- Use Case 1: Provide RPC and CLI to trigger reinstall of a service.
- Use Case 2: Provide RPC and CLI to trigger recover a service.
- Use Case 3: Provide RPC and CLI to trigger recovery of specific instance object managed by a service, referred to as service instance.

Proposed change

A new module Service Recovery Manager (SRM) will be added to Genius. SRM will provide single and common point of interaction with all individual services. Recovery options will vary from highest level service restart to restarting individual service instances.

SRM Terminology

SRM will introduce concept of service entities and operations.

SRM Entities

- EntityName - Every object in SRM is referred to as an entity and EntityName is the unique identifier for a given entity. e.g. `L3VPN`, `ITM`, `VPNInstance` etc.
- EntityType - Every entity has a corresponding type. Currently supported types are `service` and `instance`. e.g. `L3VPN` is a entity of type `service` and `VPNInstance` is an entity of type `instance`

- EntityId - Every entity of type `instance` will have a unique `entity-id` as an identifier. e.g. The `uuid` of `VPNInstance` is the `entity-id` identifying an individual VPN Instance from amongst many present in `L3VPN` service.

SRM Operations

- reinstall - This command will be used to reinstall a service. This will be similar to `karaf` bundle restart, but may result in restart of more than one bundle as per the service. This operation will only be applicable to entity-type `service`.
- recover - This command will be used to recover an individual entity, which can be `service` or `instance`. For `entity-type: service` the `entity-name` will be service name. For `entity-type: instance` the `entity-name` will be instance name and *entity-id'* will be a required field.

Example

This table gives some examples of different entities and operations for them:

OPERATION	EntityType	EntityName	EntityId	Remarks
reinstall	service	ITM	N.A.	Restart ITM
recover	service	ITM	ITM	Recover ITM Service
recover	instance	TEP	dpn-1	Recover TEP
recover	isntance	TransportZone	TZA	Recover Transport Zone

Out of Scope

- SRM will not be implementing actual recovery mechanisms, it will only act as intermediary between user and individual services.
- SRM will not provide status of services. Status and Diagnostic (SnD) framework is expected to provide service status.

Pipeline changes

N.A.

Yang changes

We'll be adding three new yang files

ServiceRecovery Types

This file will contain different types used by service recovery framework. Any service that wants to use ServiceRecovery will have to define its supported names and types in this file.

Listing 3.8: srm-types.yang

```
module srm-types {
    namespace "urn:opendaylight:genius:srm:types";
    prefix "srmtypes";

    revision "2017-05-31" {
        description "ODL Services Recovery Manager Types Module";
    }

    /* Entity TYPEs */

    identity entity-type-base {
        description "Base identity for all srm entity types";
    }
    identity entity-type-service {
        description "SRM Entity type service";
        base entity-type-base;
    }
    identity entity-type-instance {
        description "SRM Entity type instance";
        base entity-type-base;
    }


    /* Entity NAMEs */

    /* Entity Type SERVICE names */
    identity entity-name-base {
        description "Base identity for all srm entity names";
    }
    identity genius-ifm {
        description "SRM Entity name for IFM service";
        base entity-type-base;
    }
    identity genius-itm {
        description "SRM Entity name for ITM service";
        base entity-type-base;
    }
    identity netvirt-vpn {
        description "SRM Entity name for VPN service";
        base entity-type-base;
    }
    identity netvirt-elan {
        description "SRM Entity name for elan service";
        base entity-type-base;
    }
    identity ofplugin {
        description "SRM Entity name for openflowplugin service";
        base entity-type-base;
    }


    /* Entity Type INSTANCE Names */

    /* Entity types supported by GENIUS */
    identity genius-itm-tep {
        description "SRM Entity name for ITM's tep instance";
```

```
        base entity-type-base;
    }
    identity genius-itm-tz {
        description "SRM Entity name for ITM's transportzone instance";
        base entity-type-base;
    }

    identity genius-ifm-interface {
        description "SRM Entity name for IFM's interface instance";
        base entity-type-base;
    }

    /* Entity types supported by NETVIRT */
    identity netvirt-vpninstance {
        description "SRM Entity name for VPN instance";
        base entity-type-base;
    }

    identity netvirt-elaninstance {
        description "SRM Entity name for ELAN instance";
        base entity-type-base;
    }


    /* Service operations */
    identity service-op-base {
        description "Base identity for all srm operations";
    }
    identity service-op-reinstall {
        description "Reinstall or restart a service";
        base service-op-base;
    }
    identity service-op-recover {
        description "Recover a service or instance";
        base service-op-recover;
    }

}
```

ServiceRecovery Operations

This file will contain different operations that individual services must support on entities exposed by them in *servicesrecovery-types.yang*. These are not user facing operations but used by SRM to translate user RPC calls to

Listing 3.9: srm-ops.yang

```
module srm-ops {
    namespace "urn:opendaylight:genius:srm:ops";
    prefix "srmops";

    import srm-types {
        prefix srmtype;
    }

    revision "2017-05-31" {
        description "ODL Services Recovery Manager Operations Model";
```

```
    }

    /* Operations  */

    container service-ops {
        config false;
        list services {
            key service-name
            leaf service-name {
                type identityref {
                    base srmtype:entity-name-base
                }
            }
            list operations {
                key entity-name;
                leaf entity-name {
                    type identityref {
                        base srmtype:entity-name-base;
                    }
                }
                leaf entity-type {
                    type identityref {
                        base srmtype:entity-type-base;
                        mandatory true;
                    }
                }
                leaf entity-id {
                    description "Optional when entity-type is service. Actual
                                 id depends on entity-type and entity-name"
                    type string;
                }
                leaf trigger-operation {
                    type identityref {
                        base srmtypes:service-op;
                        mandatory true;
                    }
                }
            }
        }
    }

}
```

ServiceRecovery RPCs

This file will contain different RPCs supported by SRM. These RPCs are user facing and SRM will translate these into ServiceRecovery Operations as defined in *srm-ops.yang*.

Listing 3.10: srm-rpcs.yang

```
module srm-rpcs {
    namespace "urn:opendaylight:genius:srm:rpcs";
    prefix "srmrpcs";

    import srm-types {
        prefix srmtype;
```

```
    }

    revision "2017-05-31" {
        description "ODL Services Recovery Manager Rpcs Module";
    }

    /* RPCs */

    rpc reinstall {
        description "Reinstall a given service";
        input {
            leaf entity-name {
                type identityref {
                    base srmtype:entity-name-base;
                    mandatory true;
                }
            }
            leaf entity-type {
                description "Currently supported entity-types:
                                service";
                type identityref {
                    base srmtype:entity-type-base;
                    mandatory false;
                }
            }
        }
        output {
            leaf successful {
                type boolean;
            }
            leaf message {
                type string;
            }
        }
    }


    rpc recover {
        description "Recover a given service or instance";
        input {
            leaf entity-name {
                type identityref {
                    base srmtype:entity-name-base;
                    mandatory true;
                }
            }
            leaf entity-type {
                description "Currently supported entity-types:
                                service, instance";
                type identityref {
                    base srmtype:entity-type-base;
                    mandatory true;
                }
            }
            leaf entity-id {
                description "Optional when entity-type is service. Actual
                             id depends on entity-type and entity-name"
                type string;
```

```
                mandatory false;
            }
        }
        output {
            leaf response {
                type identityref {
                    base rpc-result-base;
                    mandatory true;
                }
            }
            leaf message {
                type string;
                mandatory false;
            }
        }
    }

    /* RPC RESULTs */

    identity rpc-result-base {
        description "Base identity for all SRM RPC Results";
    }
    identity rpc-success {
        description "RPC result successful";
        base rpc-result-base;
    }
    identity rpc-fail-op-not-supported {
        description "RPC failed:
                     operation not supported for given parameters";
        base rpc-result-base;
    }
    identity rpc-fail-entity-type {
        description "RPC failed:
                     invalid entity type";
        base rpc-result-base;
    }
    identity rpc-fail-entity-name {
        description "RPC failed:
                     invalid entity name";
        base rpc-result-base;
    }
    identity rpc-fail-entity-id {
        description "RPC failed:
                     invalid entity id";
        base rpc-result-base;
    }
    identity rpc-fail-unknown {
        description "RPC failed:
                     reason not known, check message string for details";
        base rpc-result-base;
    }
}
```

Configuration impact

N.A.

Clustering considerations

SRM will provide RPCs, which will only be handled on one of the nodes. In turn, it will write to `srm-ops.yang` and each individual service will have Clustered Listeners to track operations being triggered. Individual services will decide, based on service and instance on which recovery is triggered, if it needs to run on all nodes on cluster or individual nodes.

Other Infra considerations

Status and Diagnostics (SnD) may need to be updated to user service names similar to ones used in SRM.

Security considerations

Providing RPCs to trigger service restarts will eliminate the need to give administrative access to non-admin users just so they can trigger recovery though bundle restarts from karaf CLI. Expectation is access to these RPCs will be role based, but role based access and its implementation is out of scope of this feature.

Scale and Performance Impact

This feature allows recovery at a much fine grained level than full controller or node restart. Such restarts impact and trigger recovery of services that didn't need to be recover. Every restart of controller cluster or individual nodes has a significant overhead that impacts scale and performance. This feature aims to eliminate these overheads by allowing targeted recovery.

Targeted Release

Nitrogen.

Alternatives

Using existing karaf CLI for feature and bundle restart was considered but rejected due to reasons already captured in earlier sections.

Usage

TBD.

Features to Install

odl-genius

REST API

TBD.

CLI

srm:reinstall

All arguments are case insensitive unless specified otherwise.

```
DESCRIPTION
  srm:reinstall
  reinstall a given service

SYNTAX
  srm:reinstall <service-name>

ARGUMENTS
  service-name
          Name of service. to re-install e.g. itm/ITM, ifm/IFM etc.

EXAMPLE
  srm:reinstall ifm
```

srm:recover

```
DESCRIPTION
  srm:recover
  recover a service or service instance

SYNTAX
  srm:recover <entity-type> <entity-name> [<entity-id>]

ARGUMENTS
  entity-type
          Type of entity as defined in srm-types.
          e.g. service, instance etc.
  entity-name
          Entity name as defined in srm-types.
          e.g. itm, itm-tep etc.
  entity-id
          Entity Id for instances, requierd for entity-type instance.
          e.g. 'TZA', 'tunxyz' etc.

EXAMPLES
  srm:recover service itm
  srm:recover instance itm-tep TZA
  srm:recover instance vpn-instance e5e2e1ee-31a3-4d0c-a8d8-b86d08cd14b1
```

Implementation

Assignee(s)

Primary assignee: Vishal Thapar

Other contributors: Faseela K Hema Gopalakrishnan

Work Items

1. Add srm modules and features
2. Add srm yang models
3. Add code for CLI
4. Add backend implementation for RPCs to tigger SRM Operations
5. Optionally, for each service and supported instances, add implementation for SRM Operations
6. Add UTs
7. Add CSITs

Dependencies

- Infrautils

Testing

TBD.

Unit Tests

Integration Tests

CSIT

Documentation Impact

This will require changes to User Guide based on information provided in Usage section.

References

[1] Genius Nitrogen Release Plan https://wiki.opendaylight.org/view/Genius:Nitrogen_Release_Plan

[2] https://specs.openstack.org/openstack/nova-specs/specs/kilo/template.html

---

Note: This template was derived from [2], and has been modified to support our project.

3.7 Infrautils Documentation

This documentation provides critical information needed to help you write ODL Applications/Projects using Infrautils, which offers various generic utilities and infrastructure for ease of application development.

Contents:

3.7.1 InfraUtils Design Specifications

Starting from Carbon, InfraUtils project uses RST format Design Specification document for all new features. These specifications are perfect way to understand various InfraUtils features.

Contents:

Table of Contents

Title of the feature

[link to gerrit patch]

Brief introduction of the feature.

Problem description

Detailed description of the problem being solved by this feature

Use Cases

Use cases addressed by this feature.

Proposed change

Details of the proposed change.

Yang changes

This should detail any changes to yang models.

Configuration impact

Any configuration parameters being added/deprecated for this feature? What will be defaults for these? How will it impact existing deployments?

Note that outright deletion/modification of existing configuration is not allowed due to backward compatibility. They can only be deprecated and deleted in later release(s).

Clustering considerations

This should capture how clustering will be supported. This can include but not limited to use of CDTCL, EOS, Cluster Singleton etc.

Other Infra considerations

This should capture impact from/to different infra components like MDSAL Datastore, karaf, AAA etc.

Security considerations

Document any security related issues impacted by this feature.

Scale and Performance Impact

What are the potential scale and performance impacts of this change? Does it help improve scale and performance or make it worse?

Targeted Release

What release is this feature targeted for?

Alternatives

Alternatives considered and why they were not selected.

Usage

How will end user use this feature? Primary focus here is how this feature will be used in an actual deployment.

For most InfraUtils features users will be other projects but this should still capture any user visible CLI/API etc. e.g. Counters

This section will be primary input for Test and Documentation teams. Along with above this should also capture REST API and CLI.

Features to Install

odl-infrautils-all

Identify existing karaf feature to which this change applies and/or new karaf features being introduced. These can be user facing features which are added to integration/distribution or internal features to be used by other projects.

REST API

Sample JSONS/URIs. These will be an offshoot of yang changes. Capture these for User Guide, unit tests etc.

CLI

Any CLI if being added.

Implementation

Assignee(s)

Who is implementing this feature? In case of multiple authors, designate a primary assigne and other contributors.

Primary assignee: <developer-a>

Other contributors: <developer-b> <developer-c>

Work Items

Break up work into individual items. This should be a checklist on Trello card for this feature. Give link to trello card or duplicate it.

Dependencies

Any dependencies being added/removed? Dependencies here refers to internal [other ODL projects] as well as external [OVS, karaf, JDK etc.] This should also capture specific versions if any of these dependencies. e.g. OVS version, Linux kernel version, JDK etc.

This should also capture impacts on existing project that depend on InfraUtils. Following projects currently depend on Infrautils: * Netvirt * GENIUS

Testing

Capture details of testing that will need to be added.

Unit Tests

Integration Tests

Documentation Impact

What is impact on documentation for this change? If documentation change is needed call out one of the <contributors> who will work with Project Documentation Lead to get the changes done.

Don't repeat details already discussed but do reference and call them out.

References

Add any useful references. Some examples:

- Links to Summit presentation, discussion etc.
- Links to mail list discussions
- Links to patches in other projects
- Links to external documentation

[1] OpenDaylight Documentation Guide

[2] https://specs.openstack.org/openstack/nova-specs/specs/kilo/template.html

---

Note: This template was derived from [2], and has been modified to support our project.

Table of Contents

Job Coordinator

https://git.opendaylight.org/gerrit/#/q/topic:JC

Job Coordinator is a framework for executing jobs in sequential/parallel based on their job-keys. One such job,to give an example, can be for MD-SAL config/operational datastore updates.

Problem description

The concept of datastore jobcordinator was derived from the following pattern seen in many ODL project implementations :

- The Business Logic for the configuration/state handling is performed in the Actor Thread itself. This will cause the Actor's mailbox to get filled up and may start causing unnecessary back-pressure.
- Actions that can be executed independently will get unnecessarily serialized. Can cause other unrelated applications starve for chance to execute.
- Available CPU power may not be utilized fully. (for instance, if 1000 interfaces are created on different ports, all 1000 interfaces creation will happen one after the other.)
- May depend on external applications to distribute the load across the actors.

Use Cases

This feature will support following use cases:

- Use case 1: JC framework should enable applications to enqueue their jobs based on a job key.
- Use case 2: JC framework should run jobs queued on same key sequentially, and different keys parallelly.
- Use case 3: JC framework should provide a framework for retry mechanism in case the jobs fail.
- Use case 4: JC framework should provide a framework for rollback in case the jobs fail permanently.
- Use case 3: JC should provide applications the flexibility to input the number of retries on a need basis.

Proposed change

The proposed feature adds a new module in infrautils called "jobcoordinator", which will have the following functionalities:

- "Job" is a set of operations, (eg : updates to the Config/Operational MD-SAL Datastore)
- Dependent Jobs [eg. Operations on interfaces on same port] that need to be run one after the other will continue to be run in sequence.
- Independent Jobs [eg. Operations on interfaces across different Ports] will be allowed to run parallel.
- Makes use of ForkJoin Pools that allows for work-stealing across threads. ThreadPool executor flavor is also available. But would be deprecating that soon.
- Jobs are enqueued and dequeued to/from a Hash structure that ensures point 2 & 3 above are satisfied and are executed using the ForkJoinPool mentioned in point 4.
- The jobs are enqueued by the application along with an application job-key (type: string). The Coordinator dequeues and schedules the job for execution as appropriate. All jobs enqueued with the same job-key will be executed sequentially.
- Job Coordination function gets the list of listenable futures returned from each job.
- The Job is deemed complete only when the onSuccess callback is invoked and the next enqueued job for that job-key will be dequeued and executed.
- On Failure, based on application input, retries and/or rollback will be performed. Rollback failures are considered as double-fault and system bails out with error message and moves on to the next job with that Job-Key.

YANG changes

N/A

Workflow

Define Job Workers

Applications can define their own worker threads for their job. A job is defined as a piece of code that can be independently executed.

Define Rollback Workers

Applications should define a rollback worker, which will have the code to be executed in case the main job fails permanently. In usual scenarios, this will be the code to clean up all partially completed transactions by the main worker.

Decide Job Key

Applications should carefully choose the job-key for their job worker. All jobs based on the same job-key will be executed sequentially, and all jobs on different keys will be executed parallelly depending on the available threadpool size.

Enqueue Job

Applications can enqueue their job worker to JC framework for execution.JC has a hash structure to handle the execution of the tasks sequentially/parallelly. Whenever a job is enqueued, JC creates a Job Entry for the particular job. A Job Entry is characterized by - job-key, the main worker, the rollback worker and the number of retries. This JobEntry will be added to a JobQueue, which inturn is part of a JobQueueMap.

Job Queue Handling

There is a JobQueueHandler task which runs periodically, which will poll each of the JobQueues to execute the main task of the corresponding JobEntry. Within a JobQueue, execution will be synchronized.

Retries in case of failure

The list of listenable futures for the transactions from the application main worker will be available to JC, and if at all the transaction fails, the main worker will be retried the 'max-retries' number of times which is application specified. If all the retries fail, JC will bail out and the rollback worker will be executed.

Configuration impact

N/A

Clustering considerations

- Job Coordinator synchronization is not cluster-wide
- This will still work in a clustered mode by handling optimistic lock exceptions and retrying of the job.
- Future scope can be : Cluster-Wide Datastore & Switch Job Coordination in:
- Fully replicated Followers also listening Mode.
- Distributed system where no. of replicas is less than the no. of nodes in the cluster.

Other Infra considerations

N.A.

Security considerations

N.A.

Scale and Performance Impact

This feature is aiming at improving the scale and performance of applications by providing the cabability to execute their functions parallelly wherever it can be done.

Targeted Release(s)

Carbon.

Known Limitations

JC synchronization is not currently clusterwide.

Alternatives

N/A

Usage

Features to Install

This feature doesn't add any new karaf feature.

REST API

N/A

CLI

N/A

JAVA API

JobCoordinator provides the below APIs which can be used by other applications:

```
void enqueueJob(String key, Callable<List<ListenableFuture<Void>>> mainWorker).

void enqueueJob(String key, Callable<List<ListenableFuture<Void>>> mainWorker,␣
↪RollbackCallable rollbackWorker).

void enqueueJob(String key, Callable<List<ListenableFuture<Void>>> mainWorker, int␣
↪maxRetries).

void enqueueJob(String key, Callable<List<ListenableFuture<Void>>> mainWorker,␣
↪RollbackCallable rollbackWorker,
        int maxRetries).
```

key is the JobKey for synchronization, mainWorker will be the actual Job Task, maxRetries is the number of times a Job will be retried if the mainWorker results in ERROR, rollbackWorker is the Task to be executed if the Job fails with any ERROR maxRetries times.

Implementation

Assignee(s)

Primary assignee: <Periyasamy Palanisamy>

Other contributors: <Yakir Dorani> <Faseela K>

Work Items

1. spec review.
2. jobcoordinator module bring-up.
3. API definitions.
4. Enqueue Job Implementation.
5. Job Queue Handler Implementation.
6. Job Callback Implementation including retry and rollback
7. Add CLI.
8. Add UTs.
9. Add Documentation.

Dependencies

Following projects currently depend on InfraUtils:

- Netvirt
- Genius

Testing

Unit Tests

Appropriate UTs will be added for the new code coming in once framework is in place.

Integration Tests

N/A

CSIT

N/A

Documentation Impact

This will require changes to Developer Guide.

Developer Guide can capture the new set of APIs added by JobCoordinator as mentioned in API section.

References

- https://wiki.opendaylight.org/view/Infrastructure_Utilities:Carbon_Release_Plan

3.8 NetVirt Contributor Guide

3.9 Openflowplugin Documentation

This documentation provides information needed to help you write ODL Applications/Projects that can co-exist with other ODL Projects.

Contents:

3.9.1 Openflowplugin Design Specifications

Starting from Nitrogen, Openflowplugin uses RST format Design Specification document for all new features. These specifications are perfect way to understand various Openflowplugin features.

Contents:

Table of Contents

Reconciliation Framework

Reconciliation Framework Reviews

This feature aims to overcome the drawbacks of the current reconciliation implementation. As part of this enhancement, reconciliation framework will be introduced which will coordinate the reconciliation across various applications.

Applications should register themself with reconciliation framework with a specific priority. Application should decide the priority and the reconciliation framework will use it for executing in an priority.

Problem description

When a switch connected to controller, the current ODL reconciliation implementation pushes all the table/meters/groups/flows from the inventory configuration datastore to the switch.

When the switch is connected, all the applications including FRM(Forwarding Rules Manager) will receive the node added DTCN(Data Tree Change Listener) and starts pushing the flows for the openflow switch. FRM reconciliation will read the data from the config and starts pushing the flows one by one. In the meantime, applications can react to the node added DTCN change and will start pushing the flows through the config DS. With this, there is a high chance the application flow can be overwritten by the old flows by FRM via reconciliation.

With framework, the problem will be avoided by doing the reconciliation for all the registered services including FRM and then the openflow switch will be submitted to the DS. With this, applications won't receive the node added DTCN until registered applications are done with reconciliation for the switch.

The current reconciliation mechanism lacks an ordered execution of tasks across multiple applications resulting in the forwarding plane not correctly reflecting the changes in the control plane. The issue becomes more prominent in case of multi-application scenarios, resulting in errors.

Use Cases

Priority based/Ordered Coordination of Reconciliation across multiple applications.

Proposed change

Reconciliation Framework will be introduced, framework will coordinate the reconciliation across applications. The Openflow switch won't be advertised to application until Openflow switch is in KNOWN state.

`KNOWN state` controller and switch state should be in sync(reconciliation), once the switch connects.

Application participating in reconciliation needs to register with framework.

- Application can either be FRM, FRS or any other application(s).
- Application(s) registering with Reconciliation module is encouraged since: Applications would know the right Flows/Groups/Meters which needs to be replayed (Add/Delete/Update). FRM/FRS(Forwarding Rules Sync) would not have application view of flows/group, it would blindly replay the flows/groups. Also flows having idle/hard timeout can be gracefully handled by application rather than FRM/FRS.

As applications register with reconciliation module

- Reconciliation module maintains the numbers of application registered in an order based on the priority.
- Applications will be executed in the priority order of higher to lower, 1 - Highest n - lowest
- Reconciliation will be triggered as per the priority, applications with same priority will be processed in parallel, once the higher priority application completed, next priority of applications will be processed.

Openflow switch establishes connections with openflowplugin.

- Openflow switch sends connection request.
- Openflowplugin accepts connection and than establishes the connection.

Openflowplugin after establishing the connection with openflow switch, elects the mastership and invokes reconciliation framework through ReconciliationFrameworkEvent onDevicePrepared.

- Before invoking the reconciliation API, all the RPCs are registered with MD-SAL by openflowplugin.
- Reconciliation framework will register itself with the MastershipChangeServiceManager.

All registered applications would be indicated to start the reconciliation. * DeviceInfo would be passed for the API/Event and it contains all the information needed by application.

Application(s) would than fetch the flows / groups for that particular Node, which needs to be replayed.

Application(s) would than replay the selected flows / group on to the switch.

Application(s) would also wait for error from switch, for pre-defined time.

Application(s) would inform the reconciliation status to reconciliation module.

Reconciliation framework would co-relate result status from all the applications and decides the final status. If success, framework will report back DO_NOTHING and in case of failure it will be DISCONNECT.

Based on result state, openflowplugin should do the following

- On success case, openflowplugin should continue with the openflow switch –> write the switch to the operational datastore.
- On failure case, openflowplugin should disconnect the openflow switch.
- When the switch reconnects, the same steps will be followed again.

When there is a disconnect/mastership change while the reconciliation is going on, openflowplugin should notify the framework and the framework should halt the current reconciliation.

Implementation Details

Following new interface will be introduced from Reconciliation framework (RF).

- ReconciliationManager
- ReconciliationNotificationListener

ReconciliationManager

```
/* Application who are interested in reconciliation should use this API to register␣
↪themself to the RF */
/* NotificationRegistration will be return to the registered application, who needs␣
↪to take of closing the registration */
NotificationRegistration registerService(ReconciliationNotificationListener object);
```

```
/* API exposed by RF for get list of registered services */
Map<Integer, List<ReconciliationNotificationListener>> getRegisteredServices();
```

ReconciliationNotificationListener

```
/* This method will be a callback from RF to start the application reconciliation */
ListenableFuture<Boolean> startReconciliation(DeviceInfo deviceInfo);

/* This method will be a callback from RF when openflow switch disconnects during␣
↪reconciliation */
ListenableFuture<Boolean> endReconciliation(DeviceInfo deviceInfo);

/* Priority of the application */
int getPriority();

/* Name of the application */
String getName();

/* Application's intent when the application's reconciliation fails */
ResultState getResultState();
```

Priority

Framework will maintain the list of registered applications in an order based on the priority. Applications having the same priority will be executed in parallel and once those are done. Next priority applications will be called. Consider 2 applications, A and B. A is handling of programming groups and flows and B is handling of programming flows which is dependent of the groups programmed by A. So, B has to register with lower priority than A.

Application don't do any conflict resolution or guarantee any specific order among the application registered at the same priority level.

Result State - Intent Action

When the application fails to reconcile, what is the action that framework should take.

- DO_NOTHING - continue with the next reconciliation
- DISCONNECT - disconnect the switch (reconciliation will start again once the switch connects back)

Name

Name of the application who wants to register for reconciliation

ReconciliationNotificationListener

Applications who wants to register should implement ReconciliationNotificationListener interface.

- ReconciliationNotificationListener having api's like startReconciliation and endReconciliation
- startReconciliation –> applications can take action to trigger reconciliation
- endReconciliation –> application can take action to cancel their current reconcile tasks

Command Line Interface (CLI)

CLI interface will be provided to get all the registered services and their status

- List of registered services
- Status of each application for respective openflow switch

Other Changes

Pipeline changes

None.

Yang changes

None

Configuration impact

None

Clustering considerations

None

Other Infra considerations

N.A.

Security considerations

None.

Scale and Performance Impact

None.

Targeted Release

Nitrogen.

Alternatives

N.A.

Usage

Features to Install

Will be updated

REST API

None

CLI

None

Implementation

Assignee(s)

Primary assignee:

- Prasanna Huddar(prasanna.k.huddar@ericsson.com)
- Arunprakash D (d.arunprakash@ericsson.com)
- Gobinath Suganthan (gobinath@ericsson.com)

Other contributors:

Work Items

N.A.

Dependencies

This doesn't add any new dependencies.

Testing

Capture details of testing that will need to be added.

Unit Tests

None

Integration Tests

None

CSIT

None

Documentation Impact

This feature will not require any change in User Guide.

References

[1] Openflowplugin reconciliation enhancements

3.10 SFC Documentation

This documentation provides critical information needed to help you write ODL Applications/Projects that can co-exist with other ODL Projects.

Contents:

3.10.1 SFC Design Specifications

Starting from Nitrogen, SFC uses RST format Design Specification document for all new features. These specifications are perfect way to understand various SFC features.

Contents:

Table of Contents

Title of the feature

[gerrit filter: https://git.opendaylight.org/gerrit/#/q/topic:cool-topic]

Brief introduction of the feature.

Problem description

Detailed description of the problem being solved by this feature

Use Cases

Use cases addressed by this feature.

Proposed change

Details of the proposed change.

Pipeline changes

Any changes to pipeline must be captured explicitly in this section.

Yang changes

This should detail any changes to yang models.

Listing 3.11: example.yang

```
module example {
    namespace "urn:opendaylight:sfc:example";
    prefix "example";

    import ietf-yang-types {prefix yang; revision-date "2013-07-15";}

    description "An example YANG model.";

    revision 2017-02-14 { description "Initial revision"; }
}
```

Configuration impact

Any configuration parameters being added/deprecated for this feature? What will be defaults for these? How will it impact existing deployments?

Note that outright deletion/modification of existing configuration is not allowed due to backward compatibility. They can only be deprecated and deleted in later release(s).

Clustering considerations

This should capture how clustering will be supported. This can include but not limited to use of CDTCL, EOS, Cluster Singleton etc.

Other Infra considerations

This should capture impact from/to different infra components like MDSAL Datastore, karaf, AAA etc.

Security considerations

Document any security related issues impacted by this feature.

Scale and Performance Impact

What are the potential scale and performance impacts of this change? Does it help improve scale and performance or make it worse?

Targeted Release

What release is this feature targeted for?

Alternatives

Alternatives considered and why they were not selected.

Usage

How will end user use this feature? Primary focus here is how this feature will be used in an actual deployment.

This section will be primary input for Test and Documentation teams. Along with above this should also capture REST API and CLI.

Features to Install

odl-sfc-openflow-renderer

Identify existing karaf feature to which this change applies and/or new karaf features being introduced. These can be user facing features which are added to integration/distribution or internal features to be used by other projects.

REST API

Sample JSONS/URIs. These will be an offshoot of yang changes. Capture these for User Guide, CSIT, etc.

CLI

Any CLI if being added.

Implementation

Assignee(s)

Who is implementing this feature? In case of multiple authors, designate a primary assignee and other contributors.

Primary assignee: <developer-a>, <irc nick>, <email>

Other contributors: <developer-b>, <irc nick>, <email> <developer-c>, <irc nick>, <email>

Work Items

Break up work into individual items. This should be a checklist on a Trello card for this feature. Provide the link to the trello card or duplicate it.

Dependencies

Any dependencies being added/removed? Dependencies here refers to internal [other ODL projects] as well as external [OVS, karaf, JDK etc]. This should also capture specific versions if any of these dependencies. e.g. OVS version, Linux kernel version, JDK etc.

This should also capture impacts on existing projects that depend on SFC.

Following projects currently depend on SFC: GBP Netvirt

Testing

Capture details of testing that will need to be added.

Unit Tests

Integration Tests

CSIT

Documentation Impact

What is the impact on documentation for this change? If documentation changes are needed call out one of the <contributors> who will work with the Project Documentation Lead to get the changes done.

Don't repeat details already discussed but do reference and call them out.

References

Add any useful references. Some examples:

- Links to Summit presentation, discussion etc.
- Links to mail list discussions
- Links to patches in other projects
- Links to external documentation

[1] OpenDaylight Documentation Guide

[2] https://specs.openstack.org/openstack/nova-specs/specs/kilo/template.html

---

Note: This template was derived from [2], and has been modified to support our project.

Table of Contents

Karaf Command Line Interface (CLI) for SFC

[S: https://git.opendaylight.org/gerrit/#/q/topic:sfc-shell]

The Karaf Container offers a very complete Unix-like console that allows managing the container. This console can be extended with custom commands to manage the features deployed on it. This feature will add some basic commands to show the provisioned SFC's entities.

Problem description

This feature will implement commands to show some of the provisioned SFC's entities:

- Service Functions
- Service Function Forwarders
- Service Function Chains
- Service Function Paths
- Service Function Classifiers
- Service Nodes
- Service Function Types

Use Cases

- Use Case 1: list one/all provisioned Service Functions.

- Use Case 2: list one/all provisioned Service Function Forwarders.
- Use Case 3: list one/all provisioned Service Function Chains.
- Use Case 4: list one/all provisioned Service Function Paths.
- Use Case 5: list one/all provisioned Service Function Classifiers.
- Use Case 6: list one/all provisioned Service Nodes.
- Use Case 7: list one/all provisioned Service Function Types.

Proposed change

Details of the proposed change.

Pipeline changes

None

Yang changes

None

Configuration impact

None

Clustering considerations

None

Other Infra considerations

Creation of new commands for the Karaf's console.

Security considerations

None

Scale and Performance Impact

None

Targeted Release

Nitrogen

Alternatives

None

Usage

The feature will add CLI commands to the Karaf's console to list some of the provisioned SFC's entities. See the CLI section for details about the syntax of those commands.

Features to Install

odl-sfc-shell

REST API

None

CLI

- UC 1: list one/all provisioned Service Functions.

  sfc:sf-list [–name <name>]
- UC 2: list one/all provisioned Service Function Forwarders.

  sfc:sff-list [–name <name>]
- UC 3: list one/all provisioned Service Function Chains.

  sfc:sfc-list [–name <name>]
- UC 4: list one/all provisioned Service Function Paths.

  sfc:sfp-list [–name <name>]
- UC 5: list one/all provisioned Service Function Classifiers.

  sfc:sc-list [–name <name>]
- UC 6: list one/all provisioned Service Nodes.

  sfc:sn-list [–name <name>]
- UC 7: list one/all provisioned Service Function Types.

  sfc:sft-list [–name <name>]

Implementation

Assignee(s)

Primary assignee: David Suárez, #edavsua, david.suarez.fuentes@gmail.com Brady Johson, #ebrjohn, bradyallen-johnson@gmail.com

Work Items

- Implement UC 1: list one/all provisioned Service Functions.
- Implement UC 2: list one/all provisioned Service Function Forwarders.
- Implement UC 3: list one/all provisioned Service Function Chains.
- Implement UC 4: list one/all provisioned Service Function Paths.
- Implement UC 5: list one/all provisioned Service Function Classifiers.
- Implement UC 6: list one/all provisioned Service Nodes.
- Implement UC 7: list one/all provisioned Service Types.

Dependencies

This feature uses the new Karaf 4.x API to create CLI commands.

No changes needed on projects depending on SFC.

Testing

Capture details of testing that will need to be added.

Unit Tests

Integration Tests

CSIT

None

Documentation Impact

The new CLI for SFC will be documented in both the User and Developer guides.

References

Add any useful references. Some examples:

- https://docs.google.com/presentation/d/1RKkJsTUF65t40ASXVztNMcKAxMzI_owyZ-c6Mpm4Ss8/edit?usp=sharing

[1] OpenDaylight Documentation Guide

Bibliography

[QBGP] Quagga Routing Suite

[RFC2385] IETF RFC 2385: Protection of BGP Sessions via the TCP MD5 Signature Option

[TBaseProcessor] thrift java library's TBaseProcessor.process

[ZRPC] Zebra Remote Procedure Call

Python Module Index

S

Index