ODL OpenFlowPlugin

Release master

Sep 18, 2019

Contents

Contents:

CHAPTER 1

Openflowplugin User Guides

Contents:

1.1 OpenFlow Plugin Architeture

1.1.1 Overview

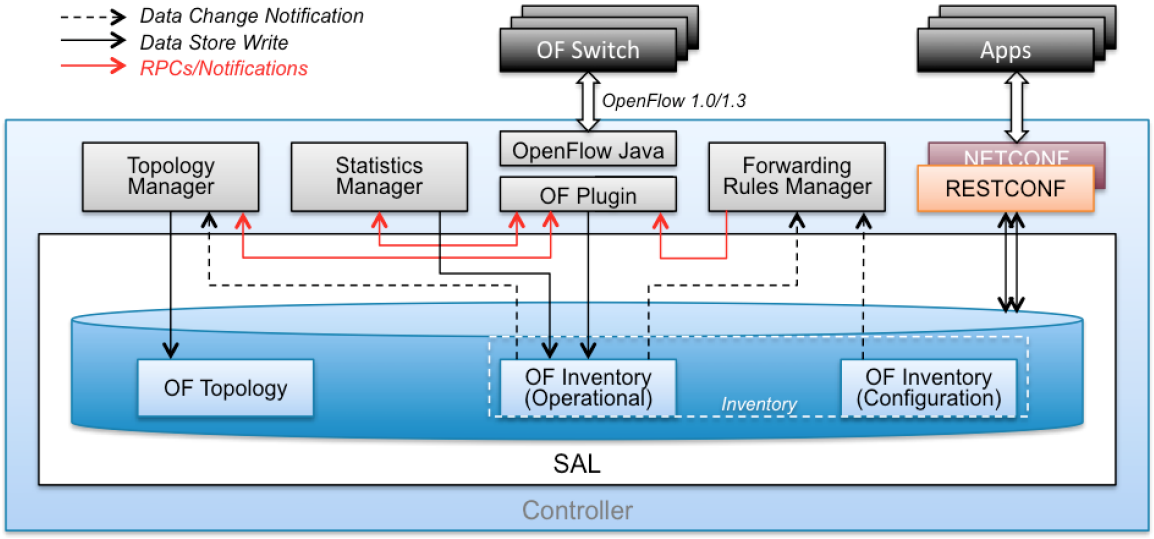
OpenFlow is a vendor-neutral standard communications interface defined to enable interaction between the control and forwarding layers of an SDN architecture. The OpenFlow plugin project intends to develop a plugin to support implementations of the OpenFlow specification as it develops and evolves. Specifically 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.

1.1.2 Goals

- Southbound plugin and integration of OpenFlow 1.0/1.3.x library.
- 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.

1.1.3 High Level Architecture

- **OpenFlowJava**: is a library that implements the OpenFlow codec – it translates OpenFlow messages into their respective internal representations and vice versa.
- **OpenFlow Plugin**: terminates sessions to OpenFlow switches, provides a per-switch low-level OpenFlow service API (add-modify-flow, delete-flow, etc.)



- **Statistics Manager**: is responsible for collecting statistics and status from attached OpenFlow switches and storing them into the operational data store for applications' use.
- **Topology Manager**: is responsible for discovering the OpenFlow topology using LLDP and putting them into the operational data store for applications' use.
- **Forwarding Rules Manager**: the "top level" OpenFlow module that exposes the OF functionality to controller apps, provides the app-level API. Main entity that manages the OpenFlow switch inventory and the configuration (programming) of flows in switches. It also reconciles user configuration with network state discovered by the OpenFlow plugin.

1.1.4 Security

It is strongly recommended that any production deployments utilising the OpenFlow Plugin do so with **TLS** encryption to protect against various man-in-the-middle attacks. For more details please refer to the *TLS section of the Operations guide*.

1.1.5 Protocol 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.

1.2 OpenFlow Plugin Installation

OpenFlow Plugin installation follows standard OpenDaylight installation procedure described in install-odl.

Next sections describe typical OpenFlow user and test features. For a complete list of available features check the OpenFlow Plugin Release Notes.

1.2.1 Typical User features

- **odl-openflowplugin-flow-services-rest**: OF plugin with REST API.
- **odl-openflowplugin-app-table-miss-enforcer**: Adds default flow to controller.
- **odl-openflowplugin-nxm-extensions**: Nicira extensions for OVS.

1.2.2 Typical Test features

- **odl-openflowplugin-app-table-miss-enforcer**: Adds default flow to controller.
- **odl-openflowplugin-drop-test**: Test application for pushing flows on packet-in.
- **odl-openflowplugin-app-bulk-o-matic**: Test application for pushing bulk flows.

1.3 OpenFlow Plugin Operation

1.3.1 Overview

The OpenFlow standard describes a communications protocol that allows an external application, such as an SDN Controller, to access and configure the forwarding plane of a network device normally called the OpenFlow-enabled switch.

The switch and the controller communicate by exchanging OpenFlow protocol messages, which the controller uses to add, modify, and delete flows in the switch. By using OpenFlow, it is possible to control various aspects of the network, such as traffic forwarding, topology discovery, Quality of Service, and so on.

For more information about OpenFlow, refer to the Open Networking Foundation website openflow-specs.

The OpenFlow Plugin provides the following RESTCONF APIs:

- OpenFlow Topology
- OpenFlow Statistics
- OpenFlow Programming

1.3.2 OpenFlow Topology

The controller provides a centralized logical view of the OpenFlow network.

The controller uses Link Layer Discovery Protocol (LLDP) messages to discover the links between the connected OpenFlow devices. The topology manager stores and manages the information (nodes and links) in the controller data stores.

This works as follows:

- LLDP speaker application sends LLDP packets to all the node connectors of all the switches that are connected.

- LLDP speaker application also monitors status events for a node connector. If the status of a node connector for the connected switch changes from up to down, the LLDP speaker does not send packets out to that node connector. If the status changes from down to up, the LLDP speaker sends packets to that node connector.
- The LLDP discovery application monitors the LLDP packets that are sent by a switch to the controller and notifies the topology manager of a new link-discovery event. The information includes: source node, source node connector, destination node, and destination node connector, from the received LLDP packets.
- The LLDP discovery application also checks for an expired link and notifies the topology manager. A link expires when it does not receive an update from the switch for the three LLDP speaker cycles.

Retrieving topology details by using RESTCONF

You can retrieve OpenFlow topology information (nodes and links) from the controller by sending a RESTCONF request. The controller fetches the topology data from the operational datastore and returns it in response to the RESTCONF request.

To view the topology data for all the connected nodes, send the following request to the controller:

Headers:

- **Content-type:** `application/xml`
- **Accept:** `application/xml`
- **Authentication:** `admin:admin`

URL: `/restconf/operational/network-topology:network-topology/topology/flow:1`

Method: GET

Sample output:

```
<topology xmlns="urn:TBD:params:xml:ns:yang:network-topology">
    <topology-id>flow:1</topology-id>
    <node>
        <node-id>openflow:1</node-id>
        <inventory-node-ref xmlns="urn:opendaylight:model:topology:inventory" xmlns:a=
↪"urn:opendaylight:inventory">/a:nodes/a:node[a:id='openflow:1']</inventory-node-ref>
        <termination-point>
            <tp-id>openflow:1:1</tp-id>
            <inventory-node-connector-ref xmlns=
↪"urn:opendaylight:model:topology:inventory" xmlns:a="urn:opendaylight:inventory">/
↪a:nodes/a:node[a:id='openflow:1']/a:node-connector[a:id='openflow:1:1']</inventory-
↪node-connector-ref>
        </termination-point>
        <termination-point>
            <tp-id>openflow:1:LOCAL</tp-id>
            <inventory-node-connector-ref xmlns=
↪"urn:opendaylight:model:topology:inventory" xmlns:a="urn:opendaylight:inventory">/
↪a:nodes/a:node[a:id='openflow:1']/a:node-connector[a:id='openflow:1:LOCAL']</
↪inventory-node-connector-ref>
        </termination-point>
        <termination-point>
            <tp-id>openflow:1:2</tp-id>
            <inventory-node-connector-ref xmlns=
↪"urn:opendaylight:model:topology:inventory" xmlns:a="urn:opendaylight:inventory">/
↪a:nodes/a:node[a:id='openflow:1']/a:node-connector[a:id='openflow:1:2']</inventory-
↪node-connector-ref>
        </termination-point>
```

(continues on next page)

(continued from previous page)

```
    </node>
    <node>
        <node-id>openflow:2</node-id>
        <inventory-node-ref xmlns="urn:opendaylight:model:topology:inventory" xmlns:a=
↪"urn:opendaylight:inventory">/a:nodes/a:node[a:id='openflow:2']</inventory-node-ref>
        <termination-point>
            <tp-id>openflow:2:2</tp-id>
            <inventory-node-connector-ref xmlns=
↪"urn:opendaylight:model:topology:inventory" xmlns:a="urn:opendaylight:inventory">/
↪a:nodes/a:node[a:id='openflow:2']/a:node-connector[a:id='openflow:2:2']</inventory-
↪node-connector-ref>
        </termination-point>
        <termination-point>
            <tp-id>openflow:2:LOCAL</tp-id>
            <inventory-node-connector-ref xmlns=
↪"urn:opendaylight:model:topology:inventory" xmlns:a="urn:opendaylight:inventory">/
↪a:nodes/a:node[a:id='openflow:2']/a:node-connector[a:id='openflow:2:LOCAL']</
↪inventory-node-connector-ref>
        </termination-point>
        <termination-point>
            <tp-id>openflow:2:1</tp-id>
            <inventory-node-connector-ref xmlns=
↪"urn:opendaylight:model:topology:inventory" xmlns:a="urn:opendaylight:inventory">/
↪a:nodes/a:node[a:id='openflow:2']/a:node-connector[a:id='openflow:2:1']</inventory-
↪node-connector-ref>
        </termination-point>
    </node>
    <link>
        <link-id>openflow:2:2</link-id>
        <source>
            <source-node>openflow:2</source-node>
            <source-tp>openflow:2:2</source-tp>
        </source>
        <destination>
            <dest-node>openflow:1</dest-node>
            <dest-tp>openflow:1:2</dest-tp>
        </destination>
    </link>
    <link>
        <link-id>openflow:1:2</link-id>
        <source>
            <source-node>openflow:1</source-node>
            <source-tp>openflow:1:2</source-tp>
        </source>
        <destination>
            <dest-node>openflow:2</dest-node>
            <dest-tp>openflow:2:2</dest-tp>
        </destination>
    </link>
</topology>
```

Note: In the example above the OpenFlow node is represented as openflow:1 where 1 is the datapath ID of the OpenFlow-enabled device.

Note: In the example above the OpenFlow node connector is represented as openflow:1:2 where 1 is the datapath ID and 2 is the port ID of the OpenFlow-enabled device.

1.3.3 OpenFlow Statistics

The controller provides the following information for the connected OpenFlow devices:

Inventory information:

- **Node description:** Description of the OpenFlow-enabled device, such as the switch manufacturer, hardware revision, software revision, serial number, and so on.
- **Flow table features:** Features supported by flow tables of the switch.
- **Port description:** Properties supported by each node connector of the node.
- **Group features:** Features supported by the group table of the switch.
- **Meter features:** Features supported by the meter table of the switch.

Statistics:

- **Individual flow statistics:** Statistics related to individual flow installed in the flow table.
- **Aggregate flow statistics:** Statistics related to aggregate flow for each table level.
- **Flow table statistics:** Statistics related to the individual flow table of the switch.
- **Port statistics:** Statistics related to all node connectors of the node.
- **Group description:** Description of the groups installed in the switch group table.
- **Group statistics:** Statistics related to an individual group installed in the group table.
- **Meter configuration:** Statistics related to the configuration of the meters installed in the switch meter table.
- **Meter statistics:** Statistics related to an individual meter installed in the switch meter table.
- **Queue statistics:** Statistics related to the queues created on each node connector of the switch.

The controller fetches both inventory and statistics information whenever a node connects to the controller. After that the controller fetches statistics periodically within a time interval of three seconds. The controller augments inventory information and statistics fetched from each connected node to its respective location in the operational data store. The controller also cleans the stale statistics at periodic intervals.

You can retrieve the inventory information (nodes, ports, and tables) and statistics (ports, flows, groups and meters) by sending a RESTCONF request. The controller fetches the inventory data from the operational data store and returns it in response to the RESTCONF request.

The following sections provide a few examples for retrieving inventory and statistics information.

Example of node inventory data

To view the inventory data of a connected node, send the following request to the controller:

Headers:

- **Content-type:** `application/xml`
- **Accept:** `application/xml`
- **Authentication:** `admin:admin`

URL: `/restconf/operational/opendaylight-inventory:nodes/node/openflow:1`

Method: `GET`

Sample output:

```
<node>
   <hardware xmlns="urn:opendaylight:flow:inventory">Open vSwitch</hardware>
   <description xmlns="urn:opendaylight:flow:inventory">None</description>
   <switch-features xmlns="urn:opendaylight:flow:inventory">
       <max_tables>254</max_tables>
       <max_buffers>0</max_buffers>
       <capabilities>flow-feature-capability-queue-stats</capabilities>
       <capabilities>flow-feature-capability-table-stats</capabilities>
       <capabilities>flow-feature-capability-flow-stats</capabilities>
       <capabilities>flow-feature-capability-port-stats</capabilities>
       <capabilities>flow-feature-capability-group-stats</capabilities>
   </switch-features>
   <manufacturer xmlns="urn:opendaylight:flow:inventory">Nicira, Inc.</manufacturer>
   <serial-number xmlns="urn:opendaylight:flow:inventory">None</serial-number>
   <software xmlns="urn:opendaylight:flow:inventory">2.8.1</software>
   <ip-address xmlns="urn:opendaylight:flow:inventory">192.168.0.24</ip-address>

   --- Omitted output ----
```

Note: In the example above the OpenFlow node is represented as openflow:1 where 1 is the datapath ID of the OpenFlow-enabled device.

Example of port description and port statistics

To view the port description and port statistics of a connected node, send the following request to the controller:

Headers:

- **Content-type:** `application/xml`
- **Accept:** `application/xml`
- **Authentication:** `admin:admin`

URL: `/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/node-connector/openflow:1:2`

Method: `GET`

Sample output:

```
<node-connector xmlns="urn:opendaylight:inventory">
    <id>openflow:1:2</id>
    <supported xmlns="urn:opendaylight:flow:inventory"></supported>
    <peer-features xmlns="urn:opendaylight:flow:inventory"></peer-features>
    <port-number xmlns="urn:opendaylight:flow:inventory">2</port-number>
    <hardware-address xmlns="urn:opendaylight:flow:inventory">4e:92:4a:c8:4c:fa</
↪hardware-address>
    <current-feature xmlns="urn:opendaylight:flow:inventory">ten-gb-fd copper</
↪current-feature>
    <maximum-speed xmlns="urn:opendaylight:flow:inventory">0</maximum-speed>
    <reason xmlns="urn:opendaylight:flow:inventory">update</reason>
```

(continues on next page)

(continued from previous page)

```
    <configuration xmlns="urn:opendaylight:flow:inventory"></configuration>
    <advertised-features xmlns="urn:opendaylight:flow:inventory"></advertised-
↪features>
    <current-speed xmlns="urn:opendaylight:flow:inventory">10000000</current-speed>
    <name xmlns="urn:opendaylight:flow:inventory">s1-eth2</name>
    <state xmlns="urn:opendaylight:flow:inventory">
        <link-down>false</link-down>
        <blocked>false</blocked>
        <live>true</live>
    </state>
    <flow-capable-node-connector-statistics xmlns="urn:opendaylight:port:statistics">
        <receive-errors>0</receive-errors>
        <packets>
            <transmitted>444</transmitted>
            <received>444</received>
        </packets>
        <receive-over-run-error>0</receive-over-run-error>
        <transmit-drops>0</transmit-drops>
        <collision-count>0</collision-count>
        <receive-frame-error>0</receive-frame-error>
        <bytes>
            <transmitted>37708</transmitted>
            <received>37708</received>
        </bytes>
        <receive-drops>0</receive-drops>
        <transmit-errors>0</transmit-errors>
        <duration>
            <second>2181</second>
            <nanosecond>550000000</nanosecond>
        </duration>
        <receive-crc-error>0</receive-crc-error>
    </flow-capable-node-connector-statistics>
</node-connector>
```

Note: In the example above the OpenFlow node connector is represented as openflow:1:2 where 1 is the datapath ID and 2 is the port ID of the OpenFlow-enabled device.

Example of flow table and aggregated statistics

To view the flow table and flow aggregated statistics for a connected node, send the following request to the controller:

Headers:

- **Content-type:** `application/xml`
- **Accept:** `application/xml`
- **Authentication:** `admin:admin`

URL: `/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/table/0/`

Method: `GET`

Sample output:

```
<table xmlns="urn:opendaylight:flow:inventory">
  <id>0</id>
  <flow-table-statistics xmlns="urn:opendaylight:flow:table:statistics">
     <active-flows>3</active-flows>
     <packets-looked-up>548</packets-looked-up>
     <packets-matched>535</packets-matched>
  </flow-table-statistics>

--- Omitted output ----
```

Note: In the example above the OpenFlow node table is 0.

Example of flow description and flow statistics

To view the individual flow statistics, send the following request to the controller:

Headers:

- **Content-type:** `application/xml`
- **Accept:** `application/xml`
- **Authentication:** `admin:admin`

URL: `/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/fm-sr-link-discovery`

Method: `GET`

Sample output:

```
<flow>
    <id>fm-sr-link-discovery</id>
    <flow-statistics xmlns="urn:opendaylight:flow:statistics">
        <packet-count>536</packet-count>
        <duration>
            <nanosecond>174000000</nanosecond>
            <second>2681</second>
        </duration>
        <byte-count>45560</byte-count>
    </flow-statistics>
    <priority>99</priority>
    <table_id>0</table_id>
    <cookie_mask>0</cookie_mask>
    <hard-timeout>0</hard-timeout>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>35020</type>
            </ethernet-type>
        </ethernet-match>
    </match>
    <cookie>1000000000000001</cookie>
    <flags></flags>
    <instructions>
        <instruction>
```

(continues on next page)

(continued from previous page)

```
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                        <max-length>65535</max-length>
                        <output-node-connector>CONTROLLER</output-node-connector>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <idle-timeout>0</idle-timeout>
</flow>
```

Note: In the example above the flow ID fm-sr-link-discovery is internal to the controller and has to match the datastore configured flow ID. For more information see flow ID match section *Flow ID match function*.

Example of group description and group statistics

To view the group description and group statistics, send the following request to the controller:

Headers:

- **Content-type:** `application/xml`
- **Accept:** `application/xml`
- **Authentication:** `admin:admin`

URL: `/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/group/2`

Method: `GET`

Sample output:

```
<group xmlns="urn:opendaylight:flow:inventory">
   <group-id>2</group-id>
   <buckets>
        <bucket>
            <bucket-id>0</bucket-id>
            <action>
                <order>1</order>
                <output-action>
                    <max-length>0</max-length>
                    <output-node-connector>2</output-node-connector>
                </output-action>
            </action>
            <action>
                <order>0</order>
                <pop-mpls-action>
                    <ethernet-type>34887</ethernet-type>
                </pop-mpls-action>
            </action>
            <watch_group>4294967295</watch_group>
```

(continues on next page)

(continued from previous page)

```
            <weight>0</weight>
            <watch_port>2</watch_port>
        </bucket>
   </buckets>
   <group-type>group-ff</group-type>
   <group-statistics xmlns="urn:opendaylight:group:statistics">
        <buckets>
            <bucket-counter>
                <bucket-id>0</bucket-id>
                <packet-count>0</packet-count>
                <byte-count>0</byte-count>
            </bucket-counter>
        </buckets>
        <group-id>2</group-id>
        <packet-count>0</packet-count>
        <byte-count>0</byte-count>
        <duration>
            <second>4116</second>
            <nanosecond>746000000</nanosecond>
        </duration>
        <ref-count>1</ref-count>
   </group-statistics>
</group>
```

Note: In the example above the group ID 2 matches the switch stored group ID.

Example of meter description and meter statistics

To view the meter description and meter statistics, send the following request to the controller:

Headers:

- **Content-type:** `application/xml`
- **Accept:** `application/xml`
- **Authentication:** `admin:admin`

URL: `/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/meter/2`

Method: `GET`

Sample output:

```
<?xml version="1.0"?>
<meter xmlns="urn:opendaylight:flow:inventory">
  <meter-id>2</meter-id>
  <flags>meter-kbps</flags>
  <meter-statistics xmlns="urn:opendaylight:meter:statistics">
    <packet-in-count>0</packet-in-count>
    <byte-in-count>0</byte-in-count>
    <meter-band-stats>
      <band-stat>
        <band-id>0</band-id>
        <byte-band-count>0</byte-band-count>
```

(continues on next page)

(continued from previous page)

```
        <packet-band-count>0</packet-band-count>
      </band-stat>
    </meter-band-stats>
    <duration>
      <nanosecond>364000000</nanosecond>
      <second>114</second>
    </duration>
    <meter-id>2</meter-id>
    <flow-count>0</flow-count>
  </meter-statistics>
  <meter-band-headers>
    <meter-band-header>
      <band-id>0</band-id>
      <band-rate>100</band-rate>
      <band-burst-size>0</band-burst-size>
      <meter-band-types>
        <flags>ofpmbt-drop</flags>
      </meter-band-types>
      <drop-burst-size>0</drop-burst-size>
      <drop-rate>100</drop-rate>
    </meter-band-header>
  </meter-band-headers>
</meter>
```

Note: In the example above the meter ID 2 matches the switch stored meter ID.

1.3.4 OpenFlow Programming

The controller provides interfaces that can be used to program the connected OpenFlow devices. These interfaces interact with the OpenFlow southbound plugin that uses OpenFlow modification messages to program flows, groups and meters in the switch.

The controller provides the following RESTCONF interfaces:

- **Configuration Datastore:** allows user to configure flows, groups and meters. The configuration is stored in the controller datastore, persisted in disk and replicated in the controller cluster. The OpenFlow southbound plugin reads the configuration and sends the appropriate OpenFlow modification messages to the connected devices.
- **RPC Operations:** allows user to configure flows, groups and meters overriding the datastore. In this case the OpenFlow southbound plugin translates the use configuration straight into an OpenFlow modification message that is sent to the connected device.

Example of flow programming by using config datastore

This example programs a flow that matches IPv4 packets (ethertype 0x800) with destination address in the 10.0.10.0/24 subnet and sends them to port 1. The flow is installed in table 0 of the switch with datapath ID 1.

Headers:

- **Content-type:** `application/xml`
- **Accept:** `application/xml`
- **Authentication:** `admin:admin`

URL: `/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1`

Method: `PUT`

Request body:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <hard-timeout>0</hard-timeout>
    <idle-timeout>0</idle-timeout>
    <cookie>1</cookie>
    <priority>2</priority>
    <flow-name>flow1</flow-name>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
        <ipv4-destination>10.0.10.0/24</ipv4-destination>
    </match>
    <id>1</id>
    <table_id>0</table_id>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <output-action>
                        <output-node-connector>1</output-node-connector>
                    </output-action>
                    <order>0</order>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
</flow>
```

Note: In the example above the flow ID 1 is internal to the controller and the same ID can be found when retrieving the flow statistics if controller finds a match between the configured flow and the flow received from switch. For more information see flow ID match section *Flow ID match function*.

Note: To use a different flow ID or table ID, ensure that the URL and the request body are synchronized.

Note: For more examples of flow programming using datastore, refer to the OpenDaylight OpenFlow plugin *Flow Examples*.

For more information about flow configuration options check the opendaylight_models.

To verify that the flow has been correctly programmed in the switch, issue the RESTCONF request as provided in *Example of flow description and flow statistics*.

Deleting flows from config datastore:

This example deletes the flow with ID 1 in table 0 of the switch with datapath ID 1.

Headers:

- **Content-type:** `application/xml`
- **Accept:** `application/xml`
- **Authentication:** `admin:admin`

URL: `/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1`

Method: `DELETE`

You can also use the below URL to delete all flows in table 0 of the switch with datapath ID 1:

URL: `/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0`

To verify that the flow has been correctly removed in the switch, issue the RESTCONF request as provided in *Example of flow table and aggregated statistics*.

Example of flow programming by using RPC operation

This example programs a flow that matches IPv4 packets (ethertype 0x800) with destination address in the 10.0.10.0/24 subnet and sends them to port 1. The flow is installed in table 0 of the switch with datapath ID 1.

Headers:

- **Content-type:** `application/xml`
- **Accept:** `application/xml`
- **Authentication:** `admin:admin`

URL: `/restconf/operations/sal-flow:add-flow`

Method: `POST`

Request body:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<input xmlns="urn:opendaylight:flow:service">
    <node xmlns:inv="urn:opendaylight:inventory">/inv:nodes/inv:node[inv:id=
↪"openflow:1"]</node>
    <table_id>0</table_id>
    <priority>2</priority>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
        <ipv4-destination>10.0.1.0/24</ipv4-destination>
    </match>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
```

(continues on next page)

(continued from previous page)

```
                    <output-action>
                        <output-node-connector>1</output-node-connector>
                    </output-action>
                    <order>0</order>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
</input>
```

Note: This payload does not require flow ID as this value is internal to controller and only used to store flows in the datastore. When retrieving flow statistics users will see an alien flow ID for flows created this way. For more information see flow ID match section *Flow ID match function*.

To verify that the flow has been correctly programmed in the switch, issue the RESTCONF request as provided in *Example of flow table and aggregated statistics*.

Deleting flows from switch using RPC operation:

This example removes a flow that matches IPv4 packets (ethertype 0x800) with destination address in the 10.0.10.0/24 subnet from table 0 of the switch with datapath ID 1.

Headers:

- **Content-type:** `application/xml`
- **Accept:** `application/xml`
- **Authentication:** `admin:admin`

URL: `/restconf/operations/sal-flow:remove-flow`

Method: `POST`

Request body:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<input xmlns="urn:opendaylight:flow:service">
    <node xmlns:inv="urn:opendaylight:inventory">/inv:nodes/inv:node[inv:id=
↪"openflow:1"]</node>
    <table_id>0</table_id>
    <priority>2</priority>
    <strict>true</strict>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
        <ipv4-destination>10.0.10.0/24</ipv4-destination>
    </match>
</input>
```

To verify that the flow has been correctly programmed in the switch, issue the RESTCONF request as provided in *Example of flow table and aggregated statistics*.

Example of a group programming by using config datastore

This example programs a select group to equally load balance traffic across port 1 and port 2 in switch with datapath ID 1.

Headers:

- **Content-type:** `application/json`
- **Accept:** `application/json`
- **Authentication:** `admin:admin`

URL: `/restconf/config/opendaylight-inventory:nodes/node/openflow:1/group/1`

Method: `PUT`

Request body:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<group xmlns="urn:opendaylight:flow:inventory">
  <group-type>group-select</group-type>
  <buckets>
      <bucket>
       <weight>1</weight>
          <action>
              <output-action>
                  <output-node-connector>1</output-node-connector>
              </output-action>
              <order>1</order>
          </action>
          <bucket-id>1</bucket-id>
      </bucket>
      <bucket>
        <weight>1</weight>
          <action>
              <output-action>
                  <output-node-connector>2</output-node-connector>
              </output-action>
              <order>1</order>
          </action>
          <bucket-id>2</bucket-id>
      </bucket>
  </buckets>
  <barrier>false</barrier>
  <group-name>SelectGroup</group-name>
  <group-id>1</group-id>
</group>
```

Note: In the example above the group ID 1 will be stored in the switch and will be used by the switch to report group statistics.

Note: To use a different group ID, ensure that the URL and the request body are synchronized.

For more information about group configuration options check the opendaylight_models.

To verify that the group has been correctly programmed in the switch, issue the RESTCONF request as provided in *Example of group description and group statistics*.

To add a group action in a flow just add this statement in the flow body:

```
<apply-actions>
    <action>
        <group-action>
            <group-id>1</group-id>
        </group-action>
        <order>1</order>
    </action>
</apply-actions>
```

Deleting groups from config datastore

This example deletes the group ID 1 in the switch with datapath ID 1.

Headers:

- **Content-type:** `application/xml`
- **Accept:** `application/xml`
- **Authentication:** `admin:admin`

URL: `/restconf/config/opendaylight-inventory:nodes/node/openflow:1/group/1`

Method: `DELETE`

Example of a meter programming by using config datastore

This example programs a meter to drop traffic exceeding 256 kbps with a burst size of 512 in switch with datapath ID 1.

Headers:

- **Content-type:** `application/json`
- **Accept:** `application/json`
- **Authentication:** `admin:admin`

URL: `/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">
    <flags>meter-kbps</flags>
    <meter-band-headers>
        <meter-band-header>
            <band-id>0</band-id>
            <drop-rate>256</drop-rate>
            <drop-burst-size>512</drop-burst-size>
            <meter-band-types>
                <flags>ofpmbt-drop</flags>
            </meter-band-types>
        </meter-band-header>
    </meter-band-headers>
    <meter-id>1</meter-id>
```

(continues on next page)

(continued from previous page)

```
    <meter-name>Foo</meter-name>
</meter>
```

Note: In the example above the meter ID 1 will be stored in the switch and will be used by the switch to report group statistics.

Note: To use a different meter ID, ensure that the URL and the request body are synchronized.

For more information about meter configuration options check the opendaylight_models.

To verify that the meter has been correctly programmed in the switch, issue the RESTCONF request as provided in *Example of meter description and meter statistics*.

To add a meter instruction in a flow just add this statement in the flow body:

```
<instructions>
   <instruction>
       <order>1</order>
       <meter>
         <meter-id>1</meter-id>
       </meter>
   </instruction>
</instructions>
```

Deleting meters from config datastore

This example deletes the meter ID 1 in the switch with datapath ID 1.

Headers:

- **Content-type:** `application/xml`
- **Accept:** `application/xml`
- **Authentication:** `admin:admin`

URL: `/restconf/config/opendaylight-inventory:nodes/node/openflow:1/meter/1`

Method: `DELETE`

1.3.5 Flow ID match function

When the controller receives flow information from a switch, this information is compared with all flows stored in the configuration datastore, in case of a match the flow ID in the flow configuration is automatically added to the flow operational information. This way we can easily relate flows stored in controller with flows received from the switch.

However in case of flows added via RPC or in general when the controller cannot match received flow information with any flow in datastore, it adds an alien ID in the flow operational information like in the example below.

```
<flow>
    <id>#UF$TABLE*0-555</id>
    <flow-statistics xmlns="urn:opendaylight:flow:statistics">
        <packet-count>5227</packet-count>
```

(continues on next page)

(continued from previous page)

```
        <duration>
            <nanosecond>642000000</nanosecond>
            <second>26132</second>
        </duration>
        <byte-count>444295</byte-count>
    </flow-statistics>
    <priority>99</priority>
    <table_id>0</table_id>
    <cookie_mask>0</cookie_mask>
    <hard-timeout>0</hard-timeout>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>35020</type>
            </ethernet-type>
        </ethernet-match>
    </match>
    <cookie>1000000000000001</cookie>
    <flags></flags>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                        <max-length>65535</max-length>
                        <output-node-connector>CONTROLLER</output-node-connector>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <idle-timeout>0</idle-timeout>
</flow>
```

1.3.6 OpenFlow clustering

For high availability, it is recommended a three-node cluster setup in which each switch is connected to all nodes in the controller cluster.

Note: Normal OpenFlow operations, such as adding a flow, can be done on any cluster member. For more information about OpenFlow operations, refer to *OpenFlow Programming*.

In OpenFlow 1.3, one of the following roles is assigned to each switch-controller connection:

- Master: All synchronous and asynchronous messages are sent to the master controller. This controller has write privileges on the switch.
- Slave: Only synchronous messages are sent to this controller. Slave controllers have only read privileges on the switch.
- Equal: When the equal role is assigned to a controller, it has the same privileges as the master controller. By default, a controller is assigned the equal role when it first connects to the switch.

A switch can be connected to one or more controllers. Each controller communicates the OpenFlow channel role through an OFTP_ROLE_REQUEST message. The switch must retain the role of each switch connection; a controller may change this role at any time.

If a switch connects to multiple controllers in the cluster, the cluster selects one controller as the master controller; the remaining controllers assume the slave role. The election of a master controller proceeds as follows.

1. Each controller in the cluster that is handling switch connections registers to the Entity Ownership Service (EOS) as a candidate for switch ownership.

   Note: The EOS is a clustering service that plays the role of the arbiter to elect an owner (master) of an entity from a registered set of candidates.

2. The EOS then selects one controller as the owner.

   Note: Master ownership is for each device; each individual controller can be a master for a set of connected devices and a slave for the remaining set of connected devices.

3. The selected owner then sends an OFTP_ROLE_REQUEST message to the switch to set the connection to the master role, and the other controllers send the role message to set the slave role.

When the switch master connection goes down, the election of a new master controller proceeds as follows.

1. The related controller deregisters itself as a candidate for Entity Ownership from the EOS.
2. The EOS then selects a new owner from the remaining candidates.
3. The new owner accordingly sends an OFTP_ROLE_REQUEST message to the switch to set the connection to the master role.

If a controller that currently has the master role is shut down, a new master from the remaining candidate controllers is selected.

1.3.7 Verifying the EOS owner and candidates by using RESTCONF

To verify the EOS owner and candidates in an OpenFlow cluster, send the following request to the controller:

Headers:

- **Content-type:** `application/json`
- **Accept:** `application/json`
- **Authentication:** `admin:admin`

URL: `/restconf/operational/entity-owners:entity-owners`

Method: `GET`

Sample output:

```
{
   "entity-owners":{
      "entity-type":[
         {
            "type":"org.opendaylight.mdsal.ServiceEntityType",
            "entity":[
               {
```

(continues on next page)

(continued from previous page)

```
                    "id":"/odl-general-entity:entity[odl-general-entity:name='openflow:1
↪']",
                    "candidate":[
                        {
                            "name":"member-3"
                        },
                        {
                            "name":"member-2"
                        },
                        {
                            "name":"member-1"
                        }
                    ],
                    "owner":"member-3"
                },
                {
                    "id":"/odl-general-entity:entity[odl-general-entity:name='openflow:2
↪']",
                    "candidate":[
                        {
                            "name":"member-1"
                        },
                        {
                            "name":"member-3"
                        },
                        {
                            "name":"member-2"
                        }
                    ],
                    "owner":"member-1"
                },
                {
                    "id":"/odl-general-entity:entity[odl-general-entity:name='openflow:3
↪']",
                    "candidate":[
                        {
                            "name":"member-1"
                        },
                        {
                            "name":"member-2"
                        },
                        {
                            "name":"member-3"
                        }
                    ],
                    "owner":"member-1"
                }
            ]
        },
        {
            "type":"org.opendaylight.mdsal.AsyncServiceCloseEntityType",
            "entity":[
                {
                    "id":"/odl-general-entity:entity[odl-general-entity:name='openflow:1
↪']",
                    "candidate":[
                        {
```

(continues on next page)

(continued from previous page)


```
                        "name":"member-3"
                     }
                  ],
                  "owner":"member-3"
               },
               {
                  "id":"/odl-general-entity:entity[odl-general-entity:name='openflow:2
↪']",
                  "candidate":[
                     {
                        "name":"member-1"
                     }
                  ],
                  "owner":"member-1"
               },
               {
                  "id":"/odl-general-entity:entity[odl-general-entity:name='openflow:3
↪']",
                  "candidate":[
                     {
                        "name":"member-1"
                     }
                  ],
                  "owner":"member-1"
               }
            ]
         }
      ]
   }
}
```


In the above sample output, `member 3` is the master controller (EOS owner) for the OpenFlow device with datapath ID `1`, and `member-1` is the master controller (EOS owner) for the OpenFlow devices with the datapath IDs of `2` and `3`.

1.3.8 Configuring the OpenFlow Plugin

OpenFlow plugin configuration file is in the opendaylight /etc folder: `opendaylight-0.9.0/etc/org.opendaylight.openflowplugin.cfg`

The `org.opendaylight.openflowplugin.cfg` file can be modified at any time, however a controller restart is required for the changes to take effect.

This configuration is local to a given node. You must repeat these steps on each node to enable the same functionality across the cluster.

1.3.9 Configuring OpenFlow TLS

This section describes how to secure OpenFlow connections between controller and OpenFlow devices using Transport Layer Security (TLS).

TLS Concepts

TLS uses digital certificates to perform remote peer authentication, message integrity and data encryption. Public Key Infrastructure (PKI) is required to create, manage and verify digital certificates.

For OpenFlow symmetric authentication (controller authenticates device and device authenticates controller) both controller and device require:

1. A private key: used to generate own public certificate and therefore required for own authentication at the other end.
2. A public certificate or a chain of certificates if public certificate is signed by an intermediate (not root) CA: the chain contains the public certificate as well as all the intermediate CA certificates used to validate the public certificate, this public information is sent to the other peer during the TLS negotiation and it is used for own authentication at the other end.
3. A list of root CA certificates: this contains the root CA certificate that signed the remote peer certificate or the remote peer intermediate CA certificate (in case of certificate chain). This public information is used to authenticate the other end.

Note: Some devices like Open vSwitch (OVS) do not support certificate chains, this means controller can only send its own certificate and receive the switch certificate without any intermediate CA certificates. For TLS negotiation to be successful in this scenario both ends need to store all intermediate CA certificates used by the other end (in addition to the remote peer root CA certificate).

Generate Controller Private Key and Certificate

You may skip this step if you already have the required key and certificate from an external Public Key Infrastructure (PKI). In the examples below we use openSSL tool to generate private key and certificates for controller.

1. Generate controller private key

   The command below generates 2048 bytes RSA key:

   ```
   openssl genrsa -out controller.key 2048
   ```

   This will generate the private key file controller.key

2. Generate controller certificate

   The command below creates a certificate sign request:

   ```
   openssl req -new -sha256 -key controller.key -out controller.csr
   ```

   This will generate the certificate signing request file controller.csr

   Submit the file to the desired Certificate Authority (CA) and get the CA signed certificate along with any intermediate CA certificate in the file controller.crt (X.509 format).

   The following is not recommended for production but if you want to just check the TLS communication you can create a "self-signed" certificate for the controller using below command:

   ```
   openssl req -new -x509 -nodes -sha1 -days 1825 -key controller.key -out␣
   ↪controller.crt
   ```

Create Controller Key Stores

Controller requires 2 Key Stores for OpenFlow TLS:

- Keystore: Used for controller authentication in the remote device. This contains the controller private key (controller.key) and the controller certificate or the controller certificate chain (controller.crt) in case of an intermediate CA signs the controller certificate.
- Truststore: Used to authenticate remote devices. This contains the root CA certificates signing the OpenFlow devices certificates or the intermediate CA certificates (in case of certificate chain).

You may skip this step if you already generated the Key Stores from a previous TLS installation. In the examples below we will use openSSL and Java keytool tooling to create the Key Stores.

1. Create the controller Keystore

   The command below generates the controller Keystore in PKCS12 format:

   ```
   openssl pkcs12 -export -in controller.crt -inkey controller.key -out keystore.p12␣
   ↪-name controller
   ```

   When asked for a password select 'opendaylight' (or anything else).

   This will generate the keystore.p12 file.

   Note: If device (e.g. Open vSwitch) does not support certificate chains, make sure controller.crt only contains the controller certificate with no extra intermediate CA certificates.

2. Create the controller Truststore

   The command below generates the controller Truststore in PKCS12 format and adds the device root CA certificates rootca1.crt and rootca2.crt:

   ```
   keytool -importcert -storetype pkcs12 -file rootca1.crt -keystore truststore.p12 -
   ↪storepass opendaylight -alias root-ca-1
   keytool -importcert -storetype pkcs12 -file rootca2.crt -keystore truststore.p12 -
   ↪storepass opendaylight -alias root-ca-2
   ```

   Note in the examples we use 'opendaylight' as the store password.

   This will generate the truststore.p12 file.

   Note: If device (e.g. Open vSwitch) does not support certificate chains, make sure you add all device intermediate CA certificates in the controller Truststore.

Enable Controller TLS

Controller listens for OpenFlow connections on ports 6633 and 6653 (TCP). You can enable TLS in both or just one of the ports.

1. Copy the Key Stores to a controller folder (e.g. opendaylight /etc folder)
2. Enable TLS on port 6633:

   Create file legacy-openflow-connection-config.xml with following content:

```
<switch-connection-config xmlns=
↪"urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:config">
  <instance-name>openflow-switch-connection-provider-legacy-impl</instance-name>
  <port>6633</port>
  <transport-protocol>TLS</transport-protocol>
  <tls>
    <keystore>etc/keystore.p12</keystore>
    <keystore-type>PKCS12</keystore-type>
    <keystore-path-type>PATH</keystore-path-type>
    <keystore-password>opendaylight</keystore-password>
    <truststore>etc/truststore.p12</truststore>
    <truststore-type>PKCS12</truststore-type>
    <truststore-path-type>PATH</truststore-path-type>
    <truststore-password>opendaylight</truststore-password>
    <certificate-password>opendaylight</certificate-password>
  </tls>
</switch-connection-config>
```

Note: Change password 'opendaylight' above if you used different password.

Note: Change the path above of you used different folder than opendaylight /etc.

Copy the file to opendaylight folder: /etc/opendaylight/datastore/initial/config

3. Enable TLS on port 6653:

   Create file default-openflow-connection-config.xml with following content:

```
<switch-connection-config xmlns=
↪"urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:config">
  <instance-name>openflow-switch-connection-provider-default-impl</instance-name>
  <port>6653</port>
  <transport-protocol>TLS</transport-protocol>
  <tls>
    <keystore>etc/keystore.p12</keystore>
    <keystore-type>PKCS12</keystore-type>
    <keystore-path-type>PATH</keystore-path-type>
    <keystore-password>opendaylight</keystore-password>
    <truststore>etc/truststore.p12</truststore>
    <truststore-type>PKCS12</truststore-type>
    <truststore-path-type>PATH</truststore-path-type>
    <truststore-password>opendaylight</truststore-password>
    <certificate-password>opendaylight</certificate-password>
  </tls>
</switch-connection-config>
```

Note: Change password 'opendaylight' above if you used different password.

Note: Change the path above of you used different folder than opendaylight /etc.

Copy the file to opendaylght folder /etc/opendaylight/datastore/initial/config

4. Restart Controller

For changes to take effect, controller has to be restarted.

1.3.10 Troubleshooting

Controller log is in opendaylight /data/log folder: `opendaylight-0.9.0/data/log/karaf.log`

Logs can be also displayed on karaf console:

```
log:display
```

To troubleshoot OpenFlow plugin enable this TRACE in karaf console:

```
log:set TRACE org.opendaylight.openflowplugin.openflow.md.core
log:set TRACE org.opendaylight.openflowplugin.impl
```

To restore log settings:

```
log:set INFO org.opendaylight.openflowplugin.openflow.md.core
log:set INFO org.opendaylight.openflowplugin.impl
```

1.4 Flow Examples

1.4.1 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>:8181/restconf/config/opendaylight-inventory:nodes/node/<Node-
↪id>/table/<Table-#>/flow/<Flow-#>

- Accept: application/xml
- Content-Type: application/xml
```

For example:

```
PUT http://localhost:8181/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.

1.4.2 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/>
```

(continues on next page)

(continued from previous page)

```
                </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>
```

(continues on next page)

(continued from previous page)

```
                <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>
```

(continues on next page)

(continued from previous page)

```
</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>
```

(continues on next page)

(continued from previous page)

```
        <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>
```

(continues on next page)

(continued from previous page)

```
        <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>
```

(continues on next page)

(continued from previous page)

```
        </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>
```

(continues on next page)

(continued from previous page)

```
    <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>
```

(continues on next page)

(continued from previous page)

```
                <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>
```

(continues on next page)

(continued from previous page)

```
        <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>
```

(continues on next page)

(continued from previous page)

```
        </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>
```

(continues on next page)

(continued from previous page)

```
    <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>
```

(continues on next page)

(continued from previous page)

```
    <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>
```

1.4.3 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>
```

(continues on next page)

(continued from previous page)

```
    <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>
```

(continues on next page)

(continued from previous page)

```
    <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>
```

(continues on next page)

(continued from previous page)

```
    <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>
```

(continues on next page)

(continued from previous page)

```
        <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>
```

(continues on next page)

(continued from previous page)

```
                    <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>
```

(continues on next page)

(continued from previous page)

```
            </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>
```

(continues on next page)

(continued from previous page)

```
        </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>
```

(continues on next page)

(continued from previous page)

```
    </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>
```

(continues on next page)

(continued from previous page)

```
                    <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>
```

(continues on next page)

(continued from previous page)

```
                    </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>
```

(continues on next page)

(continued from previous page)

```
    <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>
```

(continues on next page)

(continued from previous page)

```
        <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>
```

(continues on next page)

(continued from previous page)

```
        <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>
```

(continues on next page)

(continued from previous page)

```
      </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>
```

CHAPTER 2

Openflowplugin Developer Guides

Contents:

2.1 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.

2.1.1 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**
    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

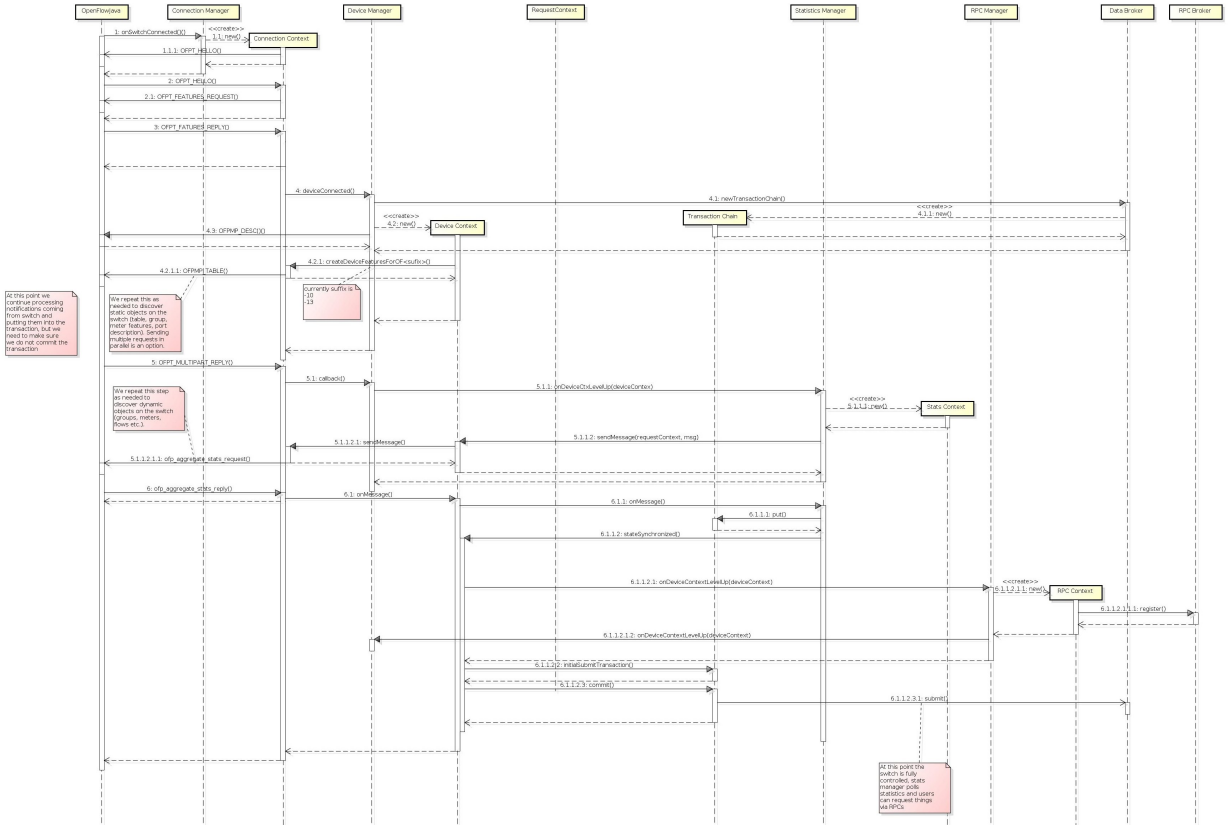


Fig. 1: Session establishment

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**.

2.1.2 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.

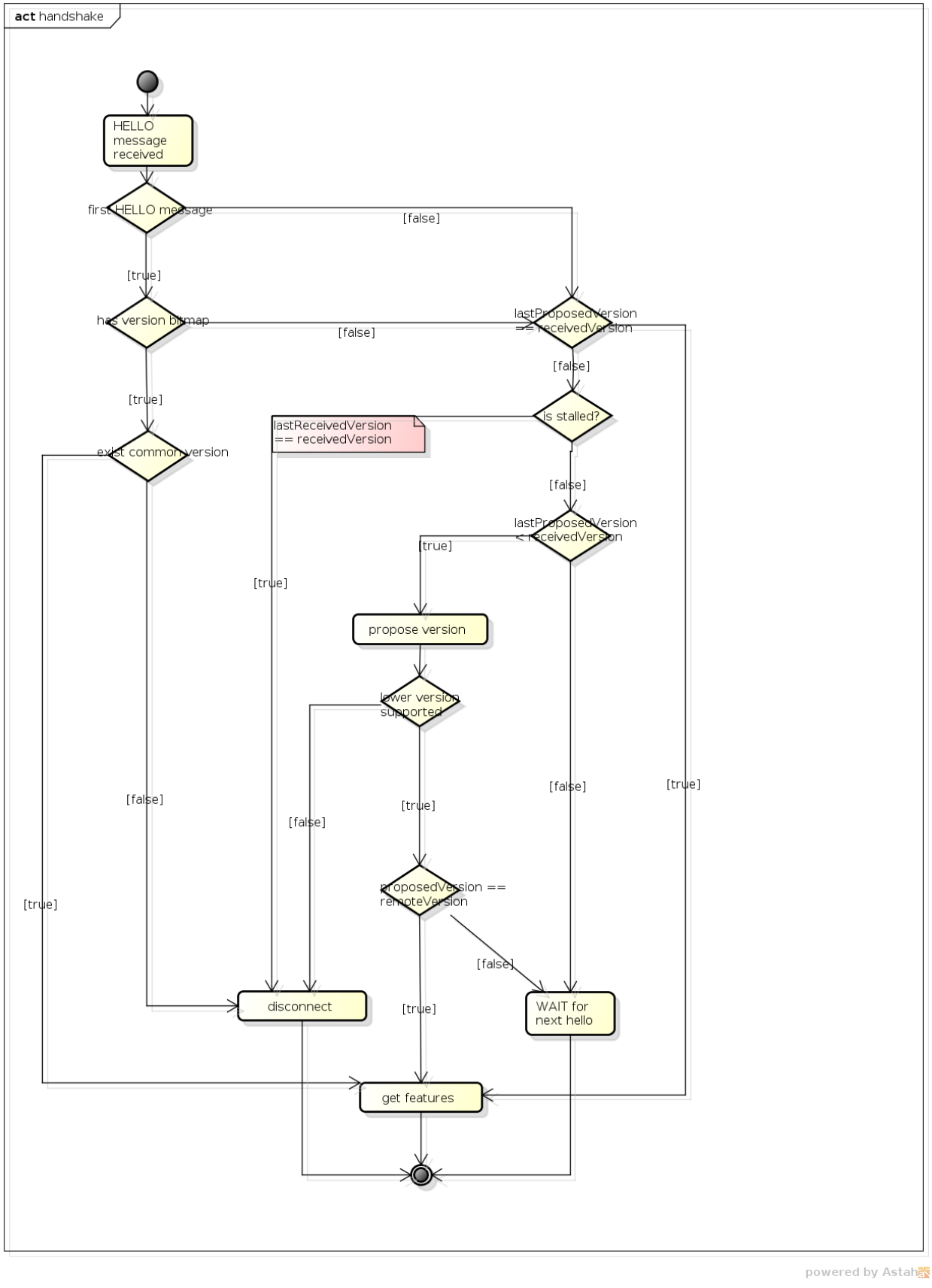


Fig. 2: Handshake process

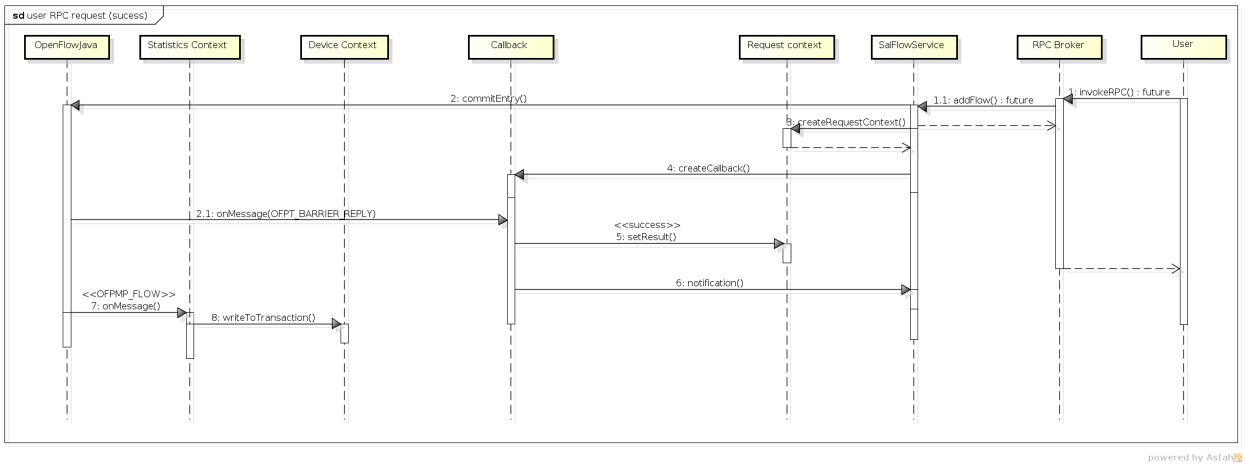


Fig. 3: Add flow

- the provided module is defined (`identity openflow-provider`)
- and target implementation is assigned (`...OpenflowPluginProvider`)

```
module openflow-provider {
   yang-version 1;
   namespace
↪"urn:opendaylight:params:xml:ns:yang:openflow:common:config[urn:opendaylight:params:xml:ns:yang:ope
↪";
   prefix "ofplugin-cfg";

   import config {prefix config; revision-date 2013-04-05; }
   description
       "openflow-plugin-custom-config";
   revision "2014-03-26" {
       description
           "Initial revision";
   }
   identity openflow-provider{
       base config:service-type;
       config:java-class "org.opendaylight.openflowplugin.openflow.md.core.sal.
↪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)

```
module openflow-provider-impl {
   yang-version 1;
```

(continues on next page)

(continued from previous page)

```
   namespace
↪"urn:opendaylight:params:xml:ns:yang:openflow:common:config:impl[urn:opendaylight:params:xml:ns:yan
↪";
   prefix "ofplugin-cfg-impl";

   import config {prefix config; revision-date 2013-04-05;}
   import openflow-provider {prefix openflow-provider;}
   import openflow-switch-connection-provider {prefix openflow-switch-connection-
↪provider;revision-date 2014-03-28;}
   import opendaylight-md-sal-binding { prefix md-sal-binding; revision-date 2013-10-
↪28;}


   description
       "openflow-plugin-custom-config-impl";

   revision "2014-03-26" {
       description
           "Initial revision";
   }

   identity openflow-provider-impl {
       base config:module-type;
       config:provided-service openflow-provider:openflow-provider;
       config:java-name-prefix ConfigurableOpenFlowProvider;
   }

   augment "/config:modules/config:module/config:configuration" {
       case openflow-provider-impl {
           when "/config:modules/config:module/config:type = 'openflow-provider-impl'
↪";

           container binding-aware-broker {
               uses config:service-ref {
                   refine type {
                       mandatory true;
                       config:required-identity md-sal-binding:binding-broker-osgi-
↪registry;
                   }
               }
           }
           list openflow-switch-connection-provider {
               uses config:service-ref {
                   refine type {
                       mandatory true;
                       config:required-identity openflow-switch-connection-
↪provider:openflow-switch-connection-provider;
                   }
               }
           }
       }
   }
}
```

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>
```

(continues on next page)

(continued from previous page)

```
          <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>
```

(continues on next page)

(continued from previous page)

```
   <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>
```

(continues on next page)

(continued from previous page)

```
      </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.

2.1.3 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

```
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
```

(continues on next page)

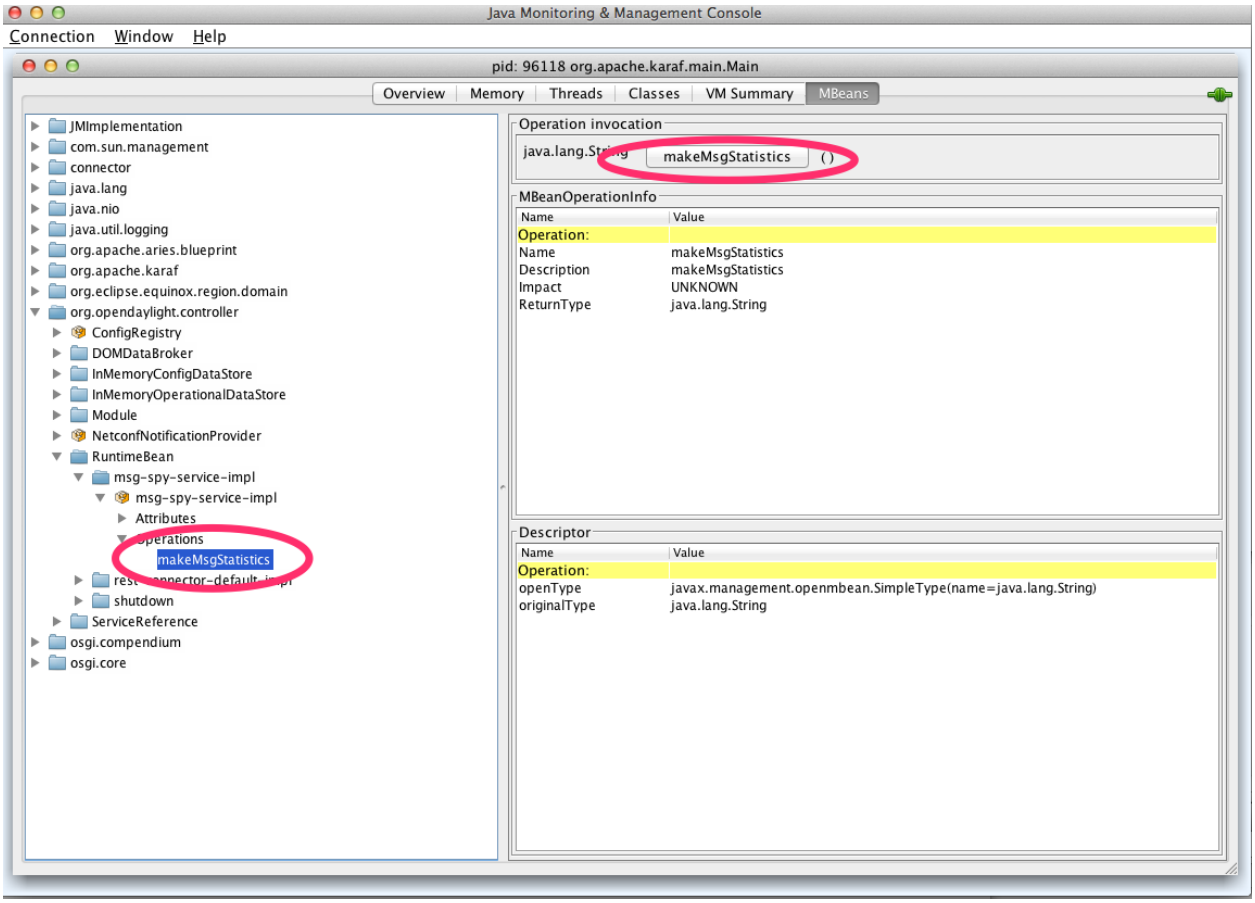


Fig. 4: OFplugin Debug stats.png

(continued from previous page)

```
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
```

2.1.4 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

2.1.5 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).

2.1.6 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.

2.1.7 Yang models and API

Model
Openflow basic types
opendaylight-table-types.yang
opendaylight-action-types.yang

Continued on next page

Table 1 – continued from previous page

Model
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

2.1.8 Karaf feature tree

Fig. 5: Openflow plugin karaf feature tree

Short HOWTO create such a tree.

2.1.9 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<>();
```

(continues on next page)

(continued from previous page)

```
        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.

2.1.10 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.

2.2 OpenFlow Protocol Library Developer Guide

2.2.1 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.

2.2.2 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.

2.2.3 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

2.2.4 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

2.2.5 Key APIs and Interfaces

Basic API / SPI classes are ConnectionAdapter (Rpc/notifications) and SwitchConnectionProcider (configure, start, shutdown)

2.2.6 Installation

Pull the code and import project into your IDE.

```
git clone ssh://<username>@git.opendaylight.org:29418/openflowjava.git
```

2.2.7 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
```

(continues on next page)

(continued from previous page)

```
            - 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.

2.2.8 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).

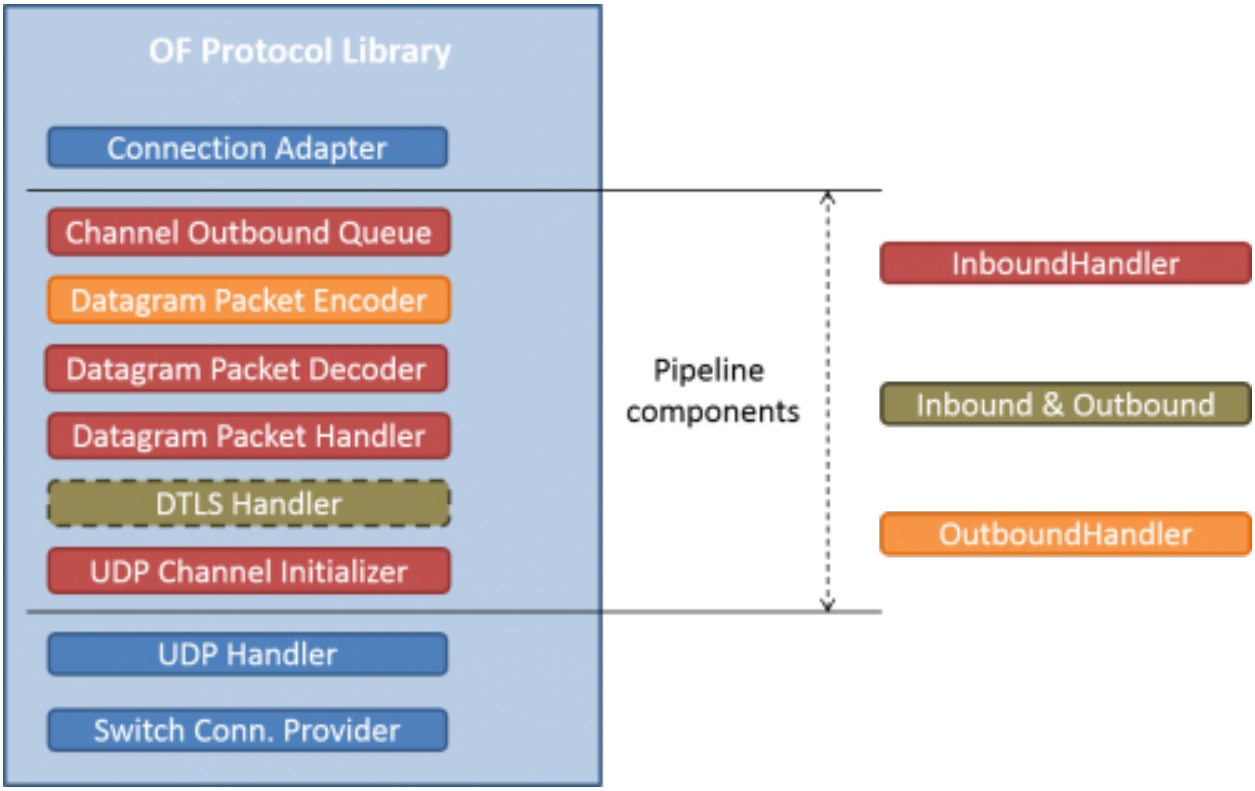


Fig. 6: UDP Channel pipeline

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.

2.2.9 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

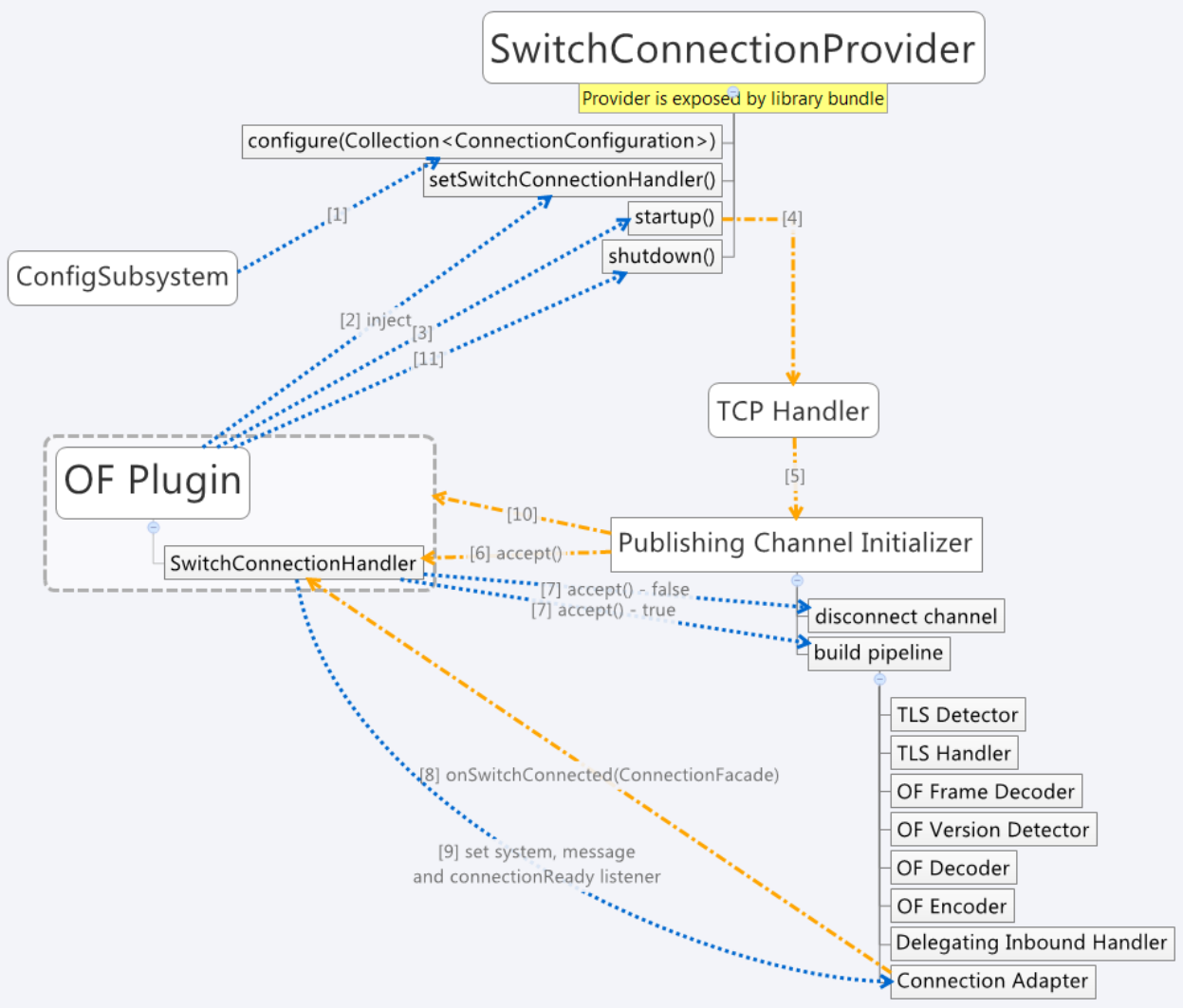


Fig. 7: Library lifecycle

2.2.10 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.

2.2.11 TLS Support

Note: see OpenFlow Plugin Developper Guide

2.2.12 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
```

(continues on next page)

(continued from previous page)

```
        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.

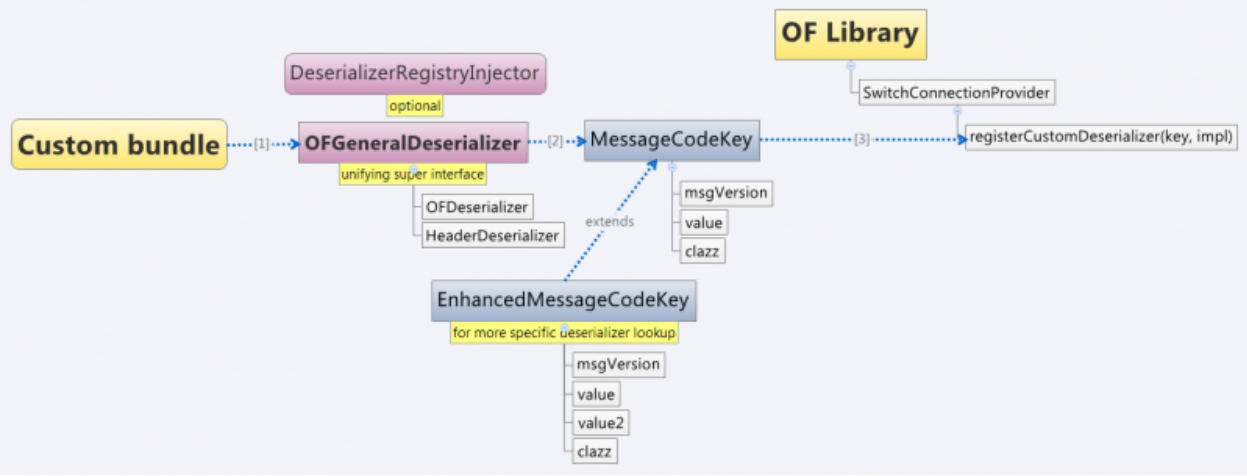


Fig. 8: Deserialization scenario walkthrough

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)

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 `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

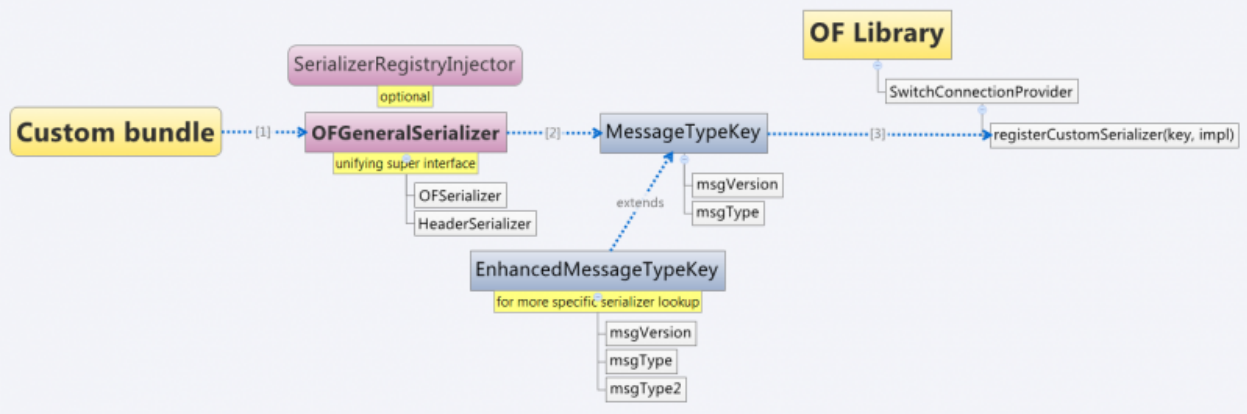


Fig. 9: Serialization scenario walkthrough

`(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(experi menter 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	OpenFlow	Registration key	Utility class
Vendor message	1.0	ExperimenterIdSerializerKey< >(1, experimenterId, ExperimenterInput.class)	ExperimenterSerialize rKeyFactory
Action	1.0	ExperimenterActionSerializer Key(1, experimenterId, subtype)	.
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, subtype)	.
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**

CHAPTER 3

Openflowplugin Design Specifications

Contents:

Table of Contents

3.1 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.

3.1.1 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.

3.1.2 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.

3.1.3 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

3.1.4 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

3.1.5 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.

3.1.6 Usage

Features to Install

Will be updated

REST API

None

CLI

None

3.1.7 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.

3.1.8 Dependencies

This doesn't add any new dependencies.

3.1.9 Testing

Capture details of testing that will need to be added.

Unit Tests

None

Integration Tests

None

CSIT

None

3.1.10 Documentation Impact

This feature will not require any change in User Guide.

3.1.11 References

[1] Openflowplugin reconciliation enhancements

Table of Contents

3.2 Group Command OFPGC_ADD_OR_MOD support

Group ADD-MOD Reviews

This spec addresses following enhancement in Openflowplugin module:

Addition of new command OFPGC_ADD_OR_MOD for OFPT_GROUP_MOD message that adds a new group that does not exist (like ADD) or modifies an existing groups (like MODIFY).

OFPGC_ADD_OR_MOD group command will be supported only for OVS2.6 and above.

3.2.1 Problem description

In OpenFlow 1.x the Group Mod commands OFPGC_ADD and OFPGC_MODIFY have strict semantics: ADD fails if the group exists, while MODIFY fails if the group does not exist. This requires a controller to exactly know the state of the switch when programming a group in order not run the risk of getting an OFP Error message in response. This is hard to achieve and maintain at all times in view of possible switch and controller restarts or other connection losses between switch and controller.

Due to the un-acknowledged nature of the Group Mod message programming groups safely and efficiently at the same time is virtually impossible as the controller has to either query the existence of the group prior to each Group Mod message or to insert a Barrier Request/Reply after every group to be sure that no Error can be received at a later stage and require a complicated roll-back of any dependent actions taken between the failed Group Mod and the Error.

Reconciliation

The current implementation of reconciliation is to read the complete set of groups from config inventory and start pushing the groups one by one. This will always end up in GROUP_ALREADY_EXITS error as the reconciliation will always send GROUP ADD.

This can be avoided by reading the groups from switch and compare with the list from inventory config and push only the delta. This is an overhead comparision and can be simply avoided by updating the group command as OFPGC_ADD_OR_MOD.

Use Cases

a. Normal group provisioning via FRM: ADD/UPDATE group should send new command OFPGC_ADD_OR_MOD.

b. Reconciliation of groups should send OFPGC_ADD_OR_MOD. Current implementation of openflowplugin will always send group add OFPGC_ADD irrespective of the state of the switch. This results in failure with GROUP_ALREADY_EXISTS error.

Proposed change

The implementation of OFPGC_ADD_OR_MOD command is specific to OVS2.6 and above and the same can be extended to other openflow switch based on the group command support by them.

New configuration parameter will be introduced in default-openflow-connection-config.xml and legacy-openflow-connection-config.xml, which can be modified by users to enable the GROUP ADD MOD support.

Listing 1: default(legacy)-openflow-connection-config.xml

```
<group-add-mod-enabled>false</group-add-mod-enabled>
```

By default the group-add-mod-enabled flag will be kept as false, which means existing group mod commands OFPGC_ADD/OFPGC_MODIFY will be used.

GroupMessageSerializer will use the above flag to determine which group command should be set for group add/update. The above class is applicable for single layer serialization and the for multi-layer serialization changes will be done in openflowjava GroupModInputMessageFactory java classs.

When flag is enabled, openflowplugin will always send OFPGC_ADD_OR_MOD (32768) for both group add and modify.

Pipeline changes

None

Yang changes

Below yang changes will be done in order to provide configuration support for group-add-mod-enabled field.

Listing 2: openflow-switch-connection-config.yang

```
leaf group-add-mod-enabled {
     description "Group Add Mod Enabled";
     type boolean;
     default false;
}
```

Configuration impact

None

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

Unknown

Targeted Release

Oxygen

Alternatives

None

3.2.2 Usage

No external rpc/api will be provided. The implementation is internal to openflowplugin.

User can enable OFPGC_ADD_OR_MOD by changing the value to true in below files,

Listing 3: default(legacy)-openflow-connection-config.xml

```
default-openflow-connection-config.xml  <group-add-mod-enabled>false</group-add-mod-
↪enabled>
legacy-openflow-connection-config.xml   <group-add-mod-enabled>false</group-add-mod-
↪enabled>
```

REST API

No new REST API is being added.

CLI

No new CLI being added.

3.2.3 Implementation

Assignee(s)

Primary assignee: Arunprakash D <d.arunprakash@ericsson.com>

Other contributors: Gobinath Suganthan <gobinath@ericsson.com>

Work Items

- Implementation of GROUP ADD MOD support
- Addition of configuration flag to enable/disable group add mod command

3.2.4 Dependencies

No new dependencies.

3.2.5 Testing

Unit Tests

1. Verify group provisioning via FRM with group-add-mod-supported disabled
2. Verify group provisioning via FRM with group-add-mod-supported enabled
3. Verify reconciliation via FRM with with group-add-mod-supported disabled
4. Verify reconciliation via FRM with with group-add-mod-supported enabled

CSIT

CSIT test cases will be added in future

3.2.6 Documentation Impact

None

3.2.7 References

Openvswitch ADD_OR_MOD

Table of Contents

3.3 Openflow Bundle Reconciliation

Bundle Reconciliation Review

This spec addresses following enhancement in Openflowplugin module:

Addition of new reconciliation mechanism in openflowplugin using openflow bundles.

Bundle reconciliation will be supported from OVS2.6 and above.

3.3.1 Problem description

Current reconciliation mechanism exists in FRM will read the config inventory data and push all the groups and flows via group and flow add messages and this mechanism is having the following limitations,

1. Group add during reconcilation will fail with GROUP_ALREADY_EXISTS error
2. Stale flows won't be removed from openflow switch after reconciliation. This leads to stale flow aggregation after every controller version upgarde.
3. Datapath traffic will get impacted as the flows will get replaced during reconciliation window.

Bundle Reconciliation

Reconciliation using openflow bundles will overcome all the above mentioned limitations. Mainly there will be minimal or no datapath traffic hit.

Bundle Concepts

A bundle is a sequence of OpenFlow requests from the controller that is applied as a single OpenFlow operation. The first goal of bundles is to group related state changes on a switch so that all changes are applied together or that none of them is applied. The second goal is to better synchronise changes across a set of OpenFlow switches, bundles can be prepared and pre-validated on each switch and applied at the same time.

A bundle is specified as all controllers messages encoded with the same bundle_id on a specific controller connection. Messages part of the bundle are encapsulated in a Bundle Add message, the payload of the Bundle Add message is formatted like a regular OpenFlow messages and has the same semantic. The messages part of a bundle are pre-validated as they are stored in the bundle, minimising the risk of errors when the bundle is applied. The applications of the message included in the Bundle Add message is postponed to when the bundle is committed.

A switch is not required to accept arbitrary messages in a bundle, a switch may not accept some message types in bundles, and a switch may not allow all combinations of message types to be bundled together. For example, a switch should not allow to embed a bundle message within a Bundle Add message. At a minimum, a switch must be able to support a bundle of multiple flow-mods and port-mods in any order.

When a bundle is opened, modifications are saved into a temporary staging area without taking effect. When the bundle is committed, the changes in the staging area are applied to the state (e.g. tables) used by the switch. If an error occurs in one modification, no change is applied to the state.

Use Cases

a. Reconciliation using openflow bundles when controller restarts

b. Reconciliation using openflow bundles when openflow switch restarts

Proposed change

Bundle reconciliation will be supported by ovs2.6 and above version or any openflow switch with bundles support.

Bundle reconciliation will be disabled by default and user has to manually enable it when needed by making a configuration change. New configuration parameter will be introduced in openflowplugin.cfg to support the same.

Listing 4: openflowplugin.cfg

```
#
# Bundle reconciliation can be enabled by making this flag to true.
# By default bundle reconciliation is disabled and reconciliation happens
# via normal flow/group mods.
# NOTE: This option will be effective with disable_reconciliation=false.
#
# bundle-based-reconciliation-enabled=false
```

By default bundle-based-reconciliation-enabled flag will be kept as false, which means reconciliation will happen via flow/group mod commands.

Following steps will be executed in order to achieve bundle reconciliation,

1. Send open bundle message to the openflow switch
2. Send delete all flows bundle message
3. Send delete all groups bundle message
4. Read flows and groups from config inventory
5. Push groups via bundle message
6. Push flows via bundle message
7. Send commit bundle message to the openflow switch

Pipeline changes

None

Yang changes

Below yang changes will be done in order to provide configuration support for bundle-based-reconciliation-enabled field.

Listing 5: forwardingrules-manager-config.yang

```
leaf bundle-based-reconciliation-enabled {
     type boolean;
     default false;
}
```

Configuration impact

None

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

Unknown

Targeted Release

Oxygen

Alternatives

None

3.3.2 Usage

No external rpc/api will be provided. The implementation is internal to openflowplugin.

User can enable bundles reconciliation by changing the value to true in openflowplugin.cfg

Listing 6: openflowplugin.cfg

```
#
# Bundle reconciliation can be enabled by making this flag to true.
# By default bundle reconciliation is disabled and reconciliation happens
# via normal flow/group mods.
# NOTE: This option will be effective with disable_reconciliation=false.
#
bundle-based-reconciliation-enabled=true
```

REST API

No new REST API is being added.

CLI

No new CLI being added.

3.3.3 Implementation

Assignee(s)

Primary assignee: Arunprakash D <d.arunprakash@ericsson.com>

Other contributors: Sunil Kumar G <sunil.g.kumar@ericsson.com>

Suja T <suja.t@ericsson.com>

Work Items

- Implementation of bundle reconciliation
- Addition of configuration flag to enable/disable bundle reconciliation

3.3.4 Dependencies

No new dependencies.

3.3.5 Testing

Unit Tests

1. Verify bundle reconciliation for controller restart
2. Verify bundle reconciliation for openflow switch restart

CSIT

CSIT test cases will be added in future

3.3.6 Documentation Impact

None

3.3.7 References

[1] https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:Bundles_extension_support

[2] https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:Bundles_usage

Table of Contents

3.4 Southbound CLI

Southbound CLI Reviews

This spec addresses following enhancement in Openflowplugin module:

Addition of new Karaf feature *odl-openflowplugin-app-southbound-cli* under openflowplugin module that provides useful CLIs for users. This feature won't be part of any existing openflowplugin feature and user needs to explicitly install it in addition to the existing features.

3.4.1 Problem description

Currently there is no way of getting the formatted list of openflow nodes connected to the OpenDaylight controller. User has to fetch operational inventory using Restconf and search for all the connected nodes. Even to get the list of ports available under a OpenFlow node, user need to search the entire inventory dump. From user experience perspective it's not really very helpful, and at scale fetching the entire inventor from data store can cause CPU spike for the controller because of the huge data present under inventory tree.

3.4.2 Southbound CLI

New Karaf feature is developed that will provide command line interface to the user using which user can retrieving the list of connected OpenFlow nodes and the ports available under each OpenFlow node.

Use Cases

- List of all OpenFlow node(s) connected to the OpenDaylight controller in either standalone or cluster environment.
- List ports information available under a connected OpenFlow node

Proposed change

New karaf feature *odl-openflowplugin-app-southbound-cli* will be added and it will not be part of any existing openflowplugin feature. User will have to explicitly install the feature to get the available CLIs.

Following 2 CLIs will be added:

- *openflow:getallnodes*
- *openflow:shownode*

openflow:getallnodes will display information like NodeId and NodeName(datapath description) for all the connected nodes.

openflow:shownode will display information like NodeId, NodeName(datapath description) and Ports for a given openflow node.

Yang changes

None

Targeted Release

Oxygen

Alternatives

Use RestConf to fetch entire operational inventory and parse through it.

3.4.3 Usage

Install `odl-openflowplugin-app-southbound-cli` feature as it is not part of any existing openflowplugin features.

List the connected openflow nodes under odl controller either in standalone or cluster environment. In clustered environment user need to install this feature on all the three nodes if it wants to use any node to run these CLI commands, but user also can choose to install it on a dedicated node only if that's the master node to run CLI commands. This feature can be install at any point of time during or after controller start.

Listing 7: openflow:getallnodes

```
opendaylight-user@root>openflow:getallnodes
Number of nodes: 1
NodeId              NodeName

------------------------------------------------------------------------
137313212546623     None
```

List the available ports under openflow node.

Listing 8: openflow:shownode

```
opendaylight-user@root>openflow:shownode -d 137313212546623
OFNode                    Name                 Ports
------------------------------------------------------------------------------
137313212546623           None                 br-int
```

3.4.4 Implementation

Assignee(s)

Primary assignee: * Arunprakash D <d.arunprakash@ericsson.com>

Contributors: * Gobinath Suganthan <gobinath@ericsson.com>

Work Items

- Implementation of cli to list the connected openflow nodes across standalone or clustered environment.
- Implementation of cli to list the ports available under openflow node.

3.4.5 Dependencies

No new dependencies.

3.4.6 Testing

Unit Tests

1. Verify CLI to list all the connected openflow nodes
2. Verify CLI to list all the ports under openflow node

CSIT

None

3.4.7 Documentation Impact

None

3.4.8 References

None

Table of Contents

3.5 Reconciliation CLI and Alarm

This spec addresses following enhancement in Openflowplugin module:

Addition of user triggered reconciliation via karaf cli command or rpc in Openflowplugin.

3.5.1 Problem description

Whenever there is a state (flow/group) mismatch between config inventory and Openflow switch, user has to either restart the Openflow switch or odl controller. This will sync the state again between odl controller and Openflow switch.

3.5.2 Reconciliation

User can trigger reconciliation to sync the state between controller and Openflow switch. It can be done either via karaf cli command or rest rpc.

3.5.3 Reconciliation Alarm

Reconciliation alarm will be generated whenever user trigger the reconciliation via cli command or rest rpc and the same will be cleared once reconciliation is completed.

Use Cases

a. Trigger reconciliation for a single Openflow switch

b. Trigger reconciliation for a list of Openflow switch

c. Trigger reconciliation for all the connected Openflow switches

d. Raise alarm whenever user triggers reconciliation for a Openflow switch

e. Clear the alarm when the reconciliation completed for a Openflow swtich

Proposed change

Karaf CLI command will be added to trigger reconciliation for the given Openflow nodes. Rest rpc will be exposed to trigger reconciliation for the given Openflow nodes.

Feature *odl-openflowplugin-app-southbound-cli* should be installed in order to get these karaf cli and rest rpc. This feature is not part of any existing openflowplugin features and has to be installed explicitly by user.

Ref: Southbound CLI

Below two CLIs will be added,

- openflow:reconcile
- openflow:getreconciliationcount

Pipeline changes

None

Yang changes

Listing 9: reconciliation.yang

```
container reconciliation-counter {
    description "Number of reconciliation triggered for openflow nodes";
    config false;
    list reconcile-counter {
        key node-id;
        uses counter;
    }
}

grouping counter {
    leaf node-id {
        type uint64;
    }
    leaf success-count {
        type uint32;
```

(continues on next page)

(continued from previous page)

```
        default 0;
    }
    leaf failure-count {
        type uint32;
        default 0;
    }
    leaf last-request-time {
        description "Timestamp when reconciliation was last requested";
        type string;
    }
}

container reconciliation-state {
    description "Reconciliation state for the given openflow nodes";
    config false;
    list reconciliation-state-list {
        key node-id;
        uses node-reconcile-state;
    }
}

grouping node-reconcile-state {
    leaf node-id {
        type uint64;
    }
    leaf state {
        description "Expresses the current state of the reconcile on a specific NODE";
        type enumeration {
            enum IN_PROGRESS;
            enum COMPLETED;
            enum FAILED;
        }
    }
}

rpc reconcile {
    description "Request the reconciliation for given device or set of devices to the␣
↪controller."
    input {
        leaf-list nodes {
            description "List of nodes to be reconciled";
            type uint64;
        }
        leaf reconcile-all-nodes {
            description "Flag to indicate that all nodes to be reconciled";
            type boolean;
            mandatory false;
            default false;
        }
    }
    output {
        leaf result {
            type boolean;
        }
        leaf-list inprogress-nodes {
            description "List of nodes that are already in reconciling mode";
            type uint64;
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

Targeted Release

Flourine

Alternatives

Disconnect the device from controller and reconnect or restart the controller.

REST API

- POST: http://localhost:8181/restconf/operations/reconciliation:reconcile
- GET: http://localhost:8181/restconf/operational/reconciliation:reconciliation-counter

3.5.4 Usage

Install `odl-openflowplugin-app-southbound-cli` feature.

CLI:

Trigger reconciliation for a connected openflow node via cli command `openflow:reconcile`.

Listing 10: openflow:reconcile

```
opendaylight-user@root>openflow:reconcile 244711506862915
reconcile successfully completed for the nodes
```

Trigger reconciliation for all the connected openflow nodes via cli command `openflow:reconcile -all`.

Listing 11: openflow:reconcile -all

```
    opendaylight-user@root>openflow:reconcile -all
    reconcile successfully completed for the nodes
```

Get details about number of times user triggered reconciliation for openflow nodes via `openflow:getreconciliationcount`.

Listing 12: openflow:getreconciliationcount

```
opendaylight-user@root>openflow:getreconcilecount
NodeId              ReconcileSuccessCount     ReconcileFailureCount     ␣
↪LastReconcileTime
------------------------------------------------------------------------------------
↪----------
244711506862915     2                         0                           2018-06-
↪06T11:51:51.989
```

REST:

Trigger reconciliation for a single datapath node.

Listing 13: http://localhost:8181/restconf/operations/reconciliation:reconcile

```
POST /restconf/operations/reconciliation:reconcile
{
  "input" :  {
    "nodes":["244711506862915"]
  }
}
```

Get reconciliation counter details

Listing 14: http://localhost:8181/restconf/operational/reconciliation:reconciliation-counter

```
GET /restconf/operational/reconciliation:reconciliation-counter

OUTPUT:
=======
Request URL
http://localhost:8181/restconf/operational/reconciliation:reconciliation-counter

Response Body
{
  "reconciliation-counter": {
    "reconcile-counter": [
      {
        "node-id": 244711506862915,
        "success-count": 4,
        "last-request-time": "2018-06-06T12:09:53.325"
      }
    ]
  }
}
```

Trigger reconciliation for a openflow switch using routed rpc. This rpc will be exposed without installing southbound-cli feature and user can trigger reconciliation for the given Openflow node. This will not affect the counter and alarm.

Listing 15: http://localhost:8181/restconf/operations/reconciliation:reconcile-node

```
POST /restconf/operations/reconciliation:reconcile-node
{
  "input": {
    "nodeId": "244711506862915",
    "node": "/opendaylight-inventory:nodes/opendaylight-inventory:node[opendaylight-
↪inventory:id='openflow:244711506862915']"
  }
}

Request URL
http://localhost:8181/restconf/operations/reconciliation:reconcile-node
```

(continues on next page)

(continued from previous page)

```
Response Body
{
  "output": {
    "result": true
  }
}
```

3.5.5 Implementation

Assignee(s)

Primary assignee:

- Arunprakash D <d.arunprakash@ericsson.com>

Contributors:

- Suja T <suja.t@ericsson.com>
- Somashekhar Javalagi <somashekhar.manohara.javalagi@ericsson.com>

Work Items

- Implementation of cli to trigger reconciliation for openflow node(s).
- Implementation of reconciliation alarm for user triggered reconciliation.

3.5.6 Dependencies

No new dependencies.

3.5.7 Testing

Unit Tests

1. Verify reconciliation for single openflow node
2. Verify reconciliation for list of openflow nodes
3. Verify reconciliation for all the openflow nodes
4. Verify reconciliation alarm generated for user triggered reconciliation node
5. Verify reconciliation alarm cleared once the reconciliation completed

CSIT

None

3.5.8 Documentation Impact

None

3.5.9 References

None

Table of Contents

3.6 Arbitrator Reconciliation using OpenFlow bundle

Arbitrator Reconciliation Reviews

This spec addresses following enhancement in openflowplugin module:

Addition of new reconciliation mode in openflowplugin which will allow applications to program flow/group within reconciliation window instead of frm reads and pushes the configuration down to the openflow switch.

AUTONOMOUS mode implies that Openflowplugin shall perform reconciliation autonomously as it does now without any change in the workflow - ie. Switch-connection-handling followed by flow-based/bundle-based reconciliation execution followed by publishing of switch to the Inventory Operational datastore. This will be the default mode until arbitrated mode is enabled.

ARBITRATED mode implies that the default openflowplugin reconciliation will be disabled and consumer application will have to initiate and complete the reconciliation including any error-handling. In the current implementation ARBITRATED mode will only supported bundle based reconciliation.

Openflowplugin will switch to arbitrator reconciliation mode based on the upgradeState provided by ServiceUtils.

3.6.1 Problem description

During replay based upgrade, the inventory configuration DS will be empty and applications has to program flows/groups based on the configuration pushed by user or external orchestrator. These new configurations has to applied on the switch without datapath disruption.

This can be achieved using OpenFlow bundles. Bundle is a sequence of OpenFlow requests from odl controller that switch will apply in atomic transaction.

Use Cases

Application controlled reconciliation of OpenFlow devices after controller re/start.

Proposed change

Arbitrator Reconciliation using bundles support will be provided. Openflowplugin will switch to arbitrator reconciliation based on the upgradeState provided by ServiceUtils. Orchestrator can enable or disable this mode as per their deployment requirements.

upgradeInProgress presents in ServiceUtils project and can be changed to true to enable arbitrator reconciliation.

Listing 16: serviceutils-upgrade-config.xml

```
<upgrade-config xmlns="urn:opendaylight:serviceutils:upgrade">
    <upgradeInProgress>false</upgradeInProgress>
</upgrade-config>
```

ArbitratorReconciliation module registers itself with reconciliation framework with priority 1.

When OpenFlow switch connect event received by Openflowplugin, it notifies Reconciliation Framework(RF).

FlowNode Reconciliation will be notified first by RF as it registered with higher priority. FlowNode reconciliation module is the one responsible for reconciliation of OpenFlow node. It can be done either via flow/group based or OpenFlow bundle based.

When upgradeInProgress is set to true, FlowNode reconciliation will be skipped as the config datastore will be empty and return success to the RF.

RF callbacks Arbitrator Reconcilition to executes its task.

Arbitrator Reconcilition will do the following steps in arbitrator-reconcilition(upgradeInProgress) mode

- Open OpenFlow bundle on the connected OpenFlow switch and stores the bundle id in the local cache
- Send delete-all groups and delete-all flows message to the opened bundle in the OpenFlow switch

NOTE: Above clean up step is needed during upgrade to clean the previous version controller states, but the real switch clean-up will only happen when controller will commit the bundle.

Arbitrator Reconciliation module sends success to RF if the previous steps are successful or it sends failure.

RF notifies Openflowplugin with the completion state.

- Success: Openflowplugin writes the OpenFlow node information into operational inventory datastore.
- Failure: OpenFlow node will be disconnected and all the above steps will be repeated on the next reconnect till the mode is in arbitrator reconciliation

Consumer application listening to inventory data store will receive Node added, Port status Data Tree Change Notification(DTCN) from data store.

Applications programs flows and groups into config inventory datastore and Forwarding Rules Manager(FRM) application in in Openflowplugin receives DTCN from config inventory for the flows and groups.

Arbitrator Reconciliation exposes rpc to get Active bundle id for the OpenFlow node.

FRM Flow/Group Forwarder invokes get-active-bundle rpc and gets the bundle id.

GetActiveBundle will executes the following steps.

- Check if bundle commit is in progress for the requested node, if yes wait on commit bundle future
- Returns Active bundle id and the same will be used by FRM forwarder to push the configuration via bundle add messages.
- This call will return null in case of arbitrator-reconciliation disabled and FRM will push the configuration via normal Flow/Group messages.

Listing 17: arbitrator-reconcile.yang

```
rpc get-active-bundle {
    description "Fetches the active available bundle in openflowplugin";
    input {
        uses "inv:node-context-ref";
        leaf node-id {
            description "Node for which the bundle active has to be fetched";
            type uint64;
        }
    }
    output {
        leaf result {
            description "The retrieved active bundle for the node";
            type "onf-ext:bundle-id";
        }
    }
}
```

Routed RPC will be exposed for committing the bundle on a specified Openflow node. It's orchestrator responsibility to commit the bundle across connected OpenFlow node. Configurations will be pushed only via OpenFlow bundles till the commit bundle rpc is invoked.

Listing 18: arbitrator-reconcile.yang

```
rpc commit-active-bundle {
    description "Commits the active available bundle for the given node in␣
↪openflowplugin";
    input {
        uses "inv:node-context-ref";
```

(continues on next page)

(continued from previous page)

```
        leaf node-id {
            description "Node for which the commit bundle to be executed";
            type uint64;
        }
    }
    output {
        leaf result {
            description "Success/Failure of the commit bundle for the node";
            type boolean;
        }
    }
}
```

Consumer application calls commit-active-bundle rpc with OpenFlow node id

- It commits the current active bundle on the OpenFlow node and stores the future till it gets completed.
- When bundle commit is in progress, configuration pushed via config datastore will be blocked on the commit future. This will make sure the new configuration is not lost during the transient state. The logic during arbitrator reconciliation will clear all the existing flows and groups and programs the new configuration and if we allow the flow programming during commit bundle phase, we might loose the new configuration.
- When commit bundle is done, it will return the rpc result to the orchestrator and removes the future from the cache.
- Subsequent flow/group provisioning will be done via flow-mod/group-mod messages.
- Orchestrator can decide further actions based on the rpc result.

Once commit bundle executes on all the connected OpenFlow switch, orchestrator can disable the arbitrator reconciliation by invoking rest rpc call on ServiceUtils *http://localhost:8383/restconf/config/odl-serviceutils-upgrade:upgrade-config/*.

Subsequent OpenFlow switch connect/re-connect will go through FlowNode reconciliation.

Note: There is no bundle timeout logic available as of now and the same will be added in future and will be kept as configurable parameter by user.

Pipeline changes

None

Yang changes

Below yang changes will done to enable arbitrator reconciliation.

RPC will be exposed to get current active bundle id for the given openflow node.

Listing 19: arbitrator-reconcile.yang

```
rpc get-active-bundle {
    description "Fetches the active available bundle in openflowplugin";
    input {
        uses "inv:node-context-ref";
        leaf node-id {
            description "Node for which the bundle active has to be fetched";
            type uint64;
```

(continues on next page)

(continued from previous page)

```
        }
    }
    output {
        leaf result {
            description "The retrieved active bundle for the node";
            type "onf-ext:bundle-id";
        }
    }
}
```

RPC will be exposed for external application/user/consumer applications to commit the active bundle for OpenFlow switch.

Listing 20: arbitrator-reconcile.yang

```
rpc commit-active-bundle {
    description "Commits the active available bundle for the given node in␣
↪openflowplugin";
    input {
        uses "inv:node-context-ref";
        leaf node-id {
            description "Node for which the commit bundle to be executed";
            type uint64;
        }
    }
    output {
        leaf result {
            description "Success/Failure of the commit bundle for the node";
            type boolean;
        }
    }
}
```

Configuration impact

None

Clustering considerations

User can fire the commit-bundle rpc call to any controller node in the cluster. This rpc will only be executed by the node that currently be owning the device.

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

Unknown

Targeted Release

Flourine

Alternatives

Default reconciliation will be used or application can just reconfigure all the configuration using the normal flow/group add/remove process.

3.6.2 Usage

None

REST API

Listing 21: http://localhost:8181/restconf/operations/arbitrator-reconcile:get-active-bundle

```
Output:
======
{
 "output": {1}
}
```

Listing 22: http://localhost:8181/restconf/operations/arbitrator-reconcile:commit-bundle-node

```
{
   "input": {
      "node": "/opendaylight-inventory:nodes/opendaylight-inventory:node[opendaylight-
↪inventory:id='openflow:<OpenFlow datapath id>']",
      "node-id": "<OpenFlow datapath id>"
   }
}
```

CLI

None.

3.6.3 Implementation

Assignee(s)

Primary assignee:

Arunprakash D <d.arunprakash@ericsson.com>

Gobinath Suganthan <gobinath@ericsson.com>

Muthukumaran K <muthukumaran.k@ericsson.com>

Work Items

- Implementation of arbritrator reconcile module
- Changes in FRM for flow/group programming via openflow bundle
- Read reconciliation mode(upgradeInProgress) from service utils
- Expose RPC to commit bundle for a given OpenFlow node

3.6.4 Dependencies

No new dependencies.

3.6.5 Testing

Unit Tests

CSIT

3.6.6 Documentation Impact

None

3.6.7 References

Bundle Extension Support

Table of Contents

3.7 Device Connection Rate Limiter

Device Connection Rate Limiter Reviews

When many openflow devices try to connect at the same time, this feature helps to reduce load on ODL controller, by limiting the number of devices connecting per minute.

Before starting the contoller, user should uncomment and configure `device-connection-rate-limit-per-min` property value in `openflowplugin.cfg` file for limiting the number of device connections.

3.7.1 Problem Description

When many openflow devices try to connect to the ODL controller via openflowplugin at the same time, controller gets overloaded by processing too many device connection requests, port information, switch feature information and supported statistics. Due to which controller gets overwhelmed, that can result in device disconnection and message drops. Hence this can largely impact the performance of the controller.

Device connection rate limiter is intended to overcome this problem by limiting the number of openflow devices connecting to the ODL controller, there by reducing the load on the controller. Due to which only configured number of devices will be able to connect to the ODL controller per minute. The remaining devices which are not able to get the permit, will be disconnected. The disconnected devices will keep on trying to connect and will be succeeded in subsequent retries, when they acquire the permit as per rate limiter logic.

Use Cases

1. By default device connection rate limiter feature will be disabled. So there will be no effect on the rate at which openflow devices connect to the ODL controller.
2. The property can be uncommented and set to any non-zero positive value in openflowplugin.cfg file, then those many number of openflow devices are allowed to connect to the ODL controller in a minute.

3.7.2 Proposed Change

1. Device connection rate limiter service is created as part of blueprint container initialization for openflowplugin-impl module.
2. Rate limiter service is created using Ratelimiter entity/class of Google's concurrency framework. ConnectionManager will be creating rate limiter service and HandshakeManager will be holding the reference to the rate limiter service.
3. Based on the value of device-connection-rate-limit-per-min property present in openflowplugin.cfg file, the rate limiter value is decided. If the value is zero, then the rate limiting functionality will be disabled or else the functionality will be enabled by allowing specified number of permits per minute.
4. At the openflow handshake phase after fetching the device features, if the rate limiter is enabled then an attempt will be made to acquire a connection permit for the openflow device. If device is able to get the permit, then the handshake process will be continued or else the device will be rejected to connect to the ODL controller. Then a disconnection event will be sent to the openflow device. The device will be succeeded to connect in subsequent retries.
5. As device-connection-rate-limit-per-min is a static property, any change in the property value will be effective only when the ODL controller is started with changed value.

3.7.3 Command Line Interface (CLI)

None.

3.7.4 Other Changes

Pipeline changes

None.

Yang changes

openflow-provider-config.yang file is modified to define the rate limiter property.

Listing 23: openflow-provider-config.yang

```
leaf device-connection-rate-limit-per-min {
     type uint16;
     default 0;
}
```

Configuration impact

New property `device-connection-rate-limit-per-min` added to openflowplugin.cfg file.

Listing 24: openflowplugin.cfg

```
# To limit the number of datapath nodes to be connected to the controller instance
# per minute. When the default value of zero is set, then the device connection rate
# limiter will be disabled. If it is set to any value, then only those many
# number of datapath nodes are allowed to connect to the controller in a minute
#
# device-connection-rate-limit-per-min=0
```

Clustering considerations

The device connection rate limiter service will be per controller basis even if controllers are connected in a clustered environment.

Other Infra considerations

N.A.

Security considerations

None.

Scale and Performance Impact

As this feature will control the rate at which the openflow devices connect to the ODL controller, it will improve the performance of controller by reducing the load in connection request processing during controller/cluster reboot.

Targeted Release

Fluorine.

Alternatives

N.A.

3.7.5 Usage

Features to Install

included with common openflowplugin features.

REST API

None

CLI

None

3.7.6 Implementation

Assignee(s)

Primary assignee:

- Somashekhar Javalagi(somashekhar.manohara.javalagi@ericsson.com)

Other contributors:

- Gobinath Suganthan (gobinath@ericsson.com)

Work Items

N.A.

3.7.7 Dependencies

This doesn't add any new dependencies.

3.7.8 Testing

1. Verifying the number of openflow device connections to the ODL controller without doing any modification to the openflowplugin.cfg file.
2. Verifying the rate at which the openflow devices connecting to the ODL controller in case if the property is having any non-zero positive value, with many devices trying to connect at the same time.

Unit Tests

None added newly.

Integration Tests

None

CSIT

None

3.7.9 Documentation Impact

3.7.10 References

CHAPTER 4

Openflowplugin Test Plans

Contents:

Table of Contents

        * *Assignee(s)*
        * *Work Items*
        * *Links*
    - *References*

4.1 Bundles-Resync

Test Suite for testing Bundles-Reconciliation functionality.

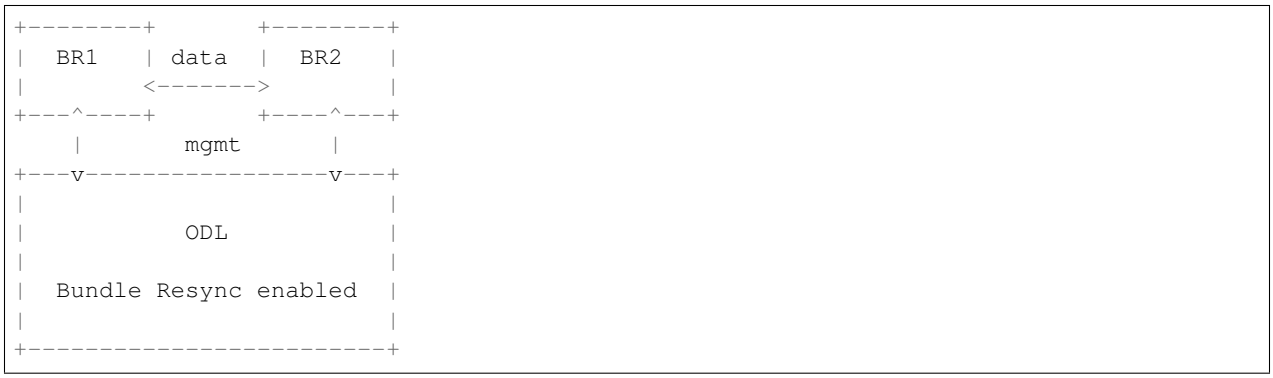
4.1.1 Test Setup

Test setup consists of ODL and two switches(Openflow nodes) connected to ODL via OpenflowPlugin Channel (6653).

Testbed Topologies

This suit uses the default topology.

Default Topology



Hardware Requirements

N.A.

Software Requirements

OVS 2.6+

4.1.2 Test Suite Requirements

Test Suite Bringup

Following steps are followed at beginning of test suite:

- Bring up ODL with *odl-openflowplugin-flow-services-rest* installed.
- Add bridge br-int to openflow node
- Connect bridge to OpenFlow using *ovs-vsctl set-controller*
- Repeat above steps for other openflow nodes

Test Suite Cleanup

Following steps are followed at the end of test suite:

- Delete bridge br-int on openflow node
- Repeat the same for other openflow nodes

Debugging

Following DataStore models are captured at end of each test case:

- config/opendaylight-inventory:nodes
- operational/opendaylight-inventory:nodes

4.1.3 Test Cases

Testcases covered in automation:

- Verify the Bundle based reconciliation with switch(OVS) restart scenario
- Verify the Bundle based reconciliation by pushing group dependent flow with switch(OVS) restart scenario
- Verify the Bundle Based reconciliation by connecting a new switch(OVS)

Verify the default reconciliation

This Verifies the default reconciliation (bundle-based-reconciliation-enabled=false)

Test Steps and Pass Criteria

1. Bring up the Controller.
2. Set the switch fail mode to Secure.
3. Push flow via Rest call and add a flow in the ovs-switch via ovs-ofctl add-flow
4. Restart the switch
5. Check that the flows pushed via Rest call should be present and the static flow added
6. Capture via Wireshark and check that the OFPT_EXP messages are not captured.
7. Check in the karaf.log and confirm if EXP messages are not logged
8. Flap the management interface of the switch.
9. Also ensure that the static flow added would be present.
10. Capture via Wireshark and check that the OFPT_EXP messages are not captured.

11. Check in the karaf.log and confirm if EXP messages are not logged
    1. Verify the Test Procedure.

Troubleshooting

N.A.

Verify the Bundle based reconciliation by enabling the flag to True

The Objective of this Testcase is to check the bundle based resync mechanism by enabling the flag

Test Steps and Pass Criteria

1. Bring up the Controller.
2. Set the bundle-based-reconciliation-enabled=true.
3. Check if the flag set event is logged in karaf.log.
4. Set the bundle-based-reconciliation-enabled=false.
5. Check if the flag set event is logged in karaf.log.
    1. Verify the Test steps.

Troubleshooting

N.A.

Verify the Bundle based reconciliation with switch(OVS) restart scenario

The Objective of this Testcase to verify bundle based reconciliation with ovs restart scenario.

Test Steps and Pass Criteria

1. Bring up the Controller.
2. Set the bundle-based-reconciliation-enabled=true.
3. Push flow via Rest call and add a flow in the ovs-switch via ovs-ofctl add-flow
4. Set the Switch fail-mode set to secure.
5. Check if the flag set event is logged in karaf.log.
6. Restart the switch
7. Check if the Wireshark has the OFPT_EXP messages captured.
8. Check for the same messages to be logged in the karaf.log.
9. Repeat the same with fail-mode set to stand-alone
10. Static flow should not be present in both stand-alone and secure mode as the switch is restarted.
    1. Verify the Test steps.

Troubleshooting

N.A.

Verify the Bundle based reconciliation by pushing group dependent flow with switch(OVS) restart scenario

The Objective of this Testcase to verify bundle based reconciliation with ovs restart scenario.

Test Steps and Pass Criteria

1. Bring up the Controller.
2. Set the bundle-based-reconciliation-enabled=true.
3. Push flow via Rest call and add a flow in the ovs-switch via ovs-ofctl add-flow
4. Set the Switch fail-mode set to secure.
5. Check if the flag set event is logged in karaf.log.
6. Restart the switch
7. Check if the Wireshark has the OFPT_EXP messages captured.
8. Check for the same messages to be logged in the karaf.log.
9. Repeat the same with fail-mode set to stand-alone
10. Static flow should not be present in both stand-alone and secure mode as the switch is restarted.
    1. Verify the Test steps.

Troubleshooting

N.A.

Verify the Bundle Based reconciliation by connecting a new switch(OVS)

The Objective of this Testcase to verify the bundle based reconciliation by connecting a new switch to the controller.

Test Steps and Pass Criteria

1. Bring up the Controller.
2. Set the bundle-based-reconciliation-enabled=true
3. Push group dependent flow via Rest call and add a flow in the ovs-switch via ovs-ofctl add-flow
4. Set the Switch fail-mode set to secure.
5. Check if the flag set event is logged in karaf.log.
6. Check if the pushed flows are there in the OVS.
7. Get a new switch connected to the Controller.
8. Push flow via Rest call and add a flow in the ovs-switch via ovs-ofctl add-flow to the newly added switch

9. Flap the management interface of the new switch.
10. Ensure the flows are pushed via bundles to the new switch.
11. Flows remain intact in the switch that was already connected.
    1. Verify the Test Steps

Troubleshooting

N.A.

Verify the Bundle based reconciliation by killing the OVS Switch Process

The Objective of this Testcase to verify the bundle based reconciliation by killing the ovs switch.

Test Steps and Pass Criteria

1. Bring up the Controller.
2. Set the bundle-based-reconciliation-enabled=true.
3. Push flow via Rest call and add a flow in the ovs-switch via ovs-ofctl add-flow
4. Set the Switch fail-mode set to secure.
5. Check if the flag set event is logged in karaf.log.
6. Kill the OVS Switch process
7. Check if the Wireshark has the OFPT_EXP messages captured.
8. Check for the same messages to be logged in the karaf.log.
9. Repeat the same with fail-mode set to stand-alone
10. Static flow should not be present in both stand-alone and secure mode.
    1. Verify the Test Steps

Troubleshooting

N.A.

4.1.4 Implementation

Assignee(s)

Primary assignee: Fathima Thasneem (a.fathima.thasneem@ericsson.com)

Other contributors: N.A.

Work Items

N.A.

Links

- https://git.opendaylight.org/gerrit/#/c/68364/
- Script path test/csit/suites/openflowplugin/Bundlebased_Reconciliation/010_bundle_resync.robot

4.1.5 References

Table of Contents

4.2 Title of Test Suite

Brief introduction of the Test Suite and feature it is testing. This should include links to relevant documents and specs.

Note: Name of suite and test cases should map exactly to as they appear in Robot reports.

4.2.1 Test Setup

Brief description of test setup.

Testbed Topologies

Detailed information on testbed topologies, including topology diagrams. Each should be numbered so it can be referenced by number in Test Cases.

Default Topology

```
+------------+
|   Dummy    |
|  Topology  |
+------------+
```

Hardware Requirements

Any specific hardware requirements e.g. SRIOV NICs.

Software Requirements

Any specific software and version requirements e.g. Mininet, OVS 2.8 etc. This should also capture specific versions of OpenDaylight this suit applies to. e.g. Nitrogen, Nitrogen-SR2 etc. This will be used to determine to which jobs this suite can/should be added.

4.2.2 Test Suite Requirements

Test Suite Bringup

Initial steps before any tests are run. This should include any cleanup, sanity checks, configuration etc. that needs to be done before test cases in suite are run. This should also capture if this suite depends on another suite to do bringup for it.

Test Suite Cleanup

Final steps after all tests in suite are done. This should include any cleanup, sanity checks, configuration etc. that needs to be done once all test cases in suite are done.

Debugging

Capture any debugging information that is captured at start of suite and end of suite.

4.2.3 Test Cases

This section should capture high level details about all the test cases part of the suite. Individual test cases should be subsections in the order they should be executed.

Test Case 1

Give a brief description of the test case including topology used if multiple specified in *Testbed Topologies*.

Test Steps and Pass Criteria

Step by step procedure of what is done as part of this test.

1. Step 1
    1. Pass Criteria 1
    2. Pass Criteria 2
2. Step 2
    1. Pass Criteria 1
    2. Pass Criteria 2

Troubleshooting

Any test specific information captured. Specifically mention if it is captured always, pass only or fail only.

4.2.4 Implementation

Assignee(s)

Who is contributing test cases? In case of multiple authors, designate a primary assignee and other contributors. Primary assignee is also expected to be maintainer once test code is in.

Primary assignee: <developer-a>

Other contributors: <developer-b> <developer-c>

Work Items

Break up work into individual items. For most cases it would be just writing tests but in some cases will include changes to images, infra etc.

Links

- Link to implementation patche(s) in CSIT

4.2.5 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.