ODL SFC

Release master

Aug 14, 2018

Contents

This documentation provides critical information needed to help you write ODL Applications/Projects that can co-exist with other ODL Projects.

Contents:

CHAPTER 1

SFC Design Specifications

Starting from Nitrogen, SFC uses RST format Design Specification document for all new features. These specifications are perfect way to understand various SFC features.

Contents:

Table of Contents

1.1 Title of the feature

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

Brief introduction of the feature.

1.1.1 Problem description

Detailed description of the problem being solved by this feature

Use Cases

Use cases addressed by this feature.

1.1.2 Proposed change

Details of the proposed change.

Pipeline changes

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

Yang changes

This should detail any changes to yang models.

Listing 1: example.yang

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

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

    description "An example YANG model.";

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

Configuration impact

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

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

Clustering considerations

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

Other Infra considerations

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

Security considerations

Document any security related issues impacted by this feature.

Scale and Performance Impact

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

Targeted Release

What release is this feature targeted for?

Alternatives

Alternatives considered and why they were not selected.

1.1.3 Usage

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

This section will be primary input for Test and Documentation teams. Along with above this should also capture REST API and CLI.

Features to Install

odl-sfc-openflow-renderer

Identify existing karaf feature to which this change applies and/or new karaf features being introduced. These can be user facing features which are added to integration/distribution or internal features to be used by other projects.

REST API

Sample JSONS/URIs. These will be an offshoot of yang changes. Capture these for User Guide, CSIT, etc.

CLI

Any CLI if being added.

1.1.4 Implementation

Assignee(s)

Who is implementing this feature? In case of multiple authors, designate a primary assignee and other contributors.

Primary assignee: <developer-a>, <irc nick>, <email>

Other contributors: <developer-b>, <irc nick>, <email> <developer-c>, <irc nick>, <email>

Work Items

Break up work into individual items. This should be a checklist on a Trello card for this feature. Provide the link to the trello card or duplicate it.

1.1.5 Dependencies

Any dependencies being added/removed? Dependencies here refers to internal [other ODL projects] as well as external [OVS, karaf, JDK etc]. This should also capture specific versions if any of these dependencies. e.g. OVS version, Linux kernel version, JDK etc.

This should also capture impacts on existing projects that depend on SFC.

Following projects currently depend on SFC: GBP Netvirt

1.1.6 Testing

Capture details of testing that will need to be added.

Unit Tests

Integration Tests

CSIT

1.1.7 Documentation Impact

What is the impact on documentation for this change? If documentation changes are needed call out one of the <contributors> who will work with the Project Documentation Lead to get the changes done.

Don't repeat details already discussed but do reference and call them out.

1.1.8 References

Add any useful references. Some examples:

- Links to Summit presentation, discussion etc.
- Links to mail list discussions
- Links to patches in other projects
- Links to external documentation

[1] OpenDaylight Documentation Guide

[2] https://specs.openstack.org/openstack/nova-specs/specs/kilo/template.html

---

Note: This template was derived from [2], and has been modified to support our project.

Table of Contents

1.2 Title of the feature

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

Currently Rendered Service Paths (RSPs) are created directly in the Operational Data Store via an RPC operation. This Spec details the refactoring involved to create RSPs first in the Config data store, and then to the Operational data store.

1.2.1 Problem description

Since the Rendered Service Paths (RSPs) are currently only written to the Operational Data Store, the RSPs will not be present in the data store when ODL restarts.

With the refactoring proposed in this spec, SFP creation will trigger RSP creation, which will cause a write to the RSP Configuration Data Store with information that needs to be persisted across ODL restarts. The RSP Configuration Data Store creation will trigger RSP Operational Data Store creation. To avoid confusion, updates to the RSP Configuration Data Store will be rejected by a newly created RSP MD-SAL Validator.

Upon ODL restart, the RSPs will be written to Operational Data Store based on what is already present in the RSP Configuration Data Store. This change will make RSP creation be inline with traditional methods of creating entries using the Config/Operational data store methodology.

Use Cases

- RSP data integrity upon ODL restart.

1.2.2 Proposed change

Pipeline changes

The existing OpenFlow pipeline will not be affected by this change.

Yang changes

All yang models related to the current RSP creation via RPC will be deprecated in Oxygen and removed in Fluorine.

It will now be possible to create the RSP in the configuration data store. The actual RSP data model contents will not change otherwise.

The following RSP data model is the updated data model, with comments highlighting the changed nodes. The changed nodes will only have the "config false" attribute modified to reflect which nodes will only be written in the Operational data store.

Listing 2: rendered-service-path.yang

```
module rendered-service-path {
   container rendered-service-paths {
      // UPDATED This container is no longer "config false"
      description
        "A container that holds the list of all Rendered Service Paths
         in a SFC domain";
      list rendered-service-path {
        key "name";
        description
          "A list that holds operational data for all RSPs in the
           domain";
        leaf name {
          type sfc-common:rsp-name;
          description
            "The name of this rendered function path. This is the same
             name as the associated SFP";
        }
        leaf parent-service-function-path {
          type sfc-common:sfp-name;
          description
            "Service Function Path from which this RSP was
             instantiated";
        }
        leaf transport-type {
          // UPDATED This node is now "config false"
          config false;

          type sfc-sl:sl-transport-type-def;
          default "sfc-sl:vxlan-gpe";
          description
            "Transport type as set in the Parent Service Function
             Path";
        }
        leaf context-metadata {
          // UPDATED This node is now "config false"
          config false;

          type sfc-md:context-metadata-ref;
```

(continues on next page)

(continued from previous page)

```
          description
            "The name of the associated context metadata";
        }
        leaf variable-metadata {
          // UPDATED This node is now "config false"
          config false;

          type sfc-md:variable-metadata-ref;
          description
            "The name of the associated variable metadata";
        }
        leaf tenant-id {
          type string;
          description
            "This RSP was created for a specific tenant-id";
        }
        uses sfc-ss:service-statistics-group {
          // UPDATED This node is now "config false"
          config false;

          description "Global Rendered Service Path statistics";
        }
        list rendered-service-path-hop {
          key "hop-number";
          leaf hop-number {
            type uint8;
            description
              "A Monotonically increasing number";
          }
          leaf service-function-name {
            type sfc-common:sf-name;
            description
              "Service Function name";
          }
          leaf service-function-group-name {
            type string;
            description
              "Service Function group name";
          }
          leaf service-function-forwarder {
            type sfc-common:sff-name;
            description
              "Service Function Forwarder name";
          }
          leaf service-function-forwarder-locator {
            type sfc-common:sff-data-plane-locator-name;
            description
              "The name of the SFF data plane locator";
          }
          leaf service-index {
            type uint8;
            description
              "Provides location within the service path.
               Service index MUST be decremented by service functions
               or proxy nodes after performing required services.  MAY
               be used in conjunction with service path for path
               selection.  Service Index is also valuable when
```

(continues on next page)

(continued from previous page)

```
            troubleshooting/reporting service paths.  In addition to
            location within a path, SI can be used for loop
            detection.";
        }
        ordered-by user;
        description
          "A list of service functions that compose the
           service path";
      }
      leaf service-chain-name {
        // UPDATED This node is now "config false"
        config false;

        type sfc-common:sfc-name;
        mandatory true;
        description
          "The Service Function Chain used as blueprint for this
           path";
      }
      leaf starting-index {
        // UPDATED This node is now "config false"
        config false;

        type uint8;
        description
          "Starting service index";
      }
      leaf path-id {
        type uint32 {
          range "0..16777216";
        }
        mandatory true;
        description
          "Identifies a service path.
           Participating nodes MUST use this identifier for path
           selection.  An administrator can use the service path
           value for reporting and troubleshooting packets along
           a specific path.";
      }
      leaf symmetric-path-id {
        type uint32 {
          range "0..16777216";
        }
        description
          "Identifies the associated symmetric path, if any.";
      }
      leaf sfc-encapsulation {
        // UPDATED This node is now "config false"
        config false;

        type sfc-sl:sfc-encapsulation-type;
        description
          "The type of encapsulation used in this path for passing
          SFC information along the chain";
      }
    }
  }
```

(continues on next page)

(continued from previous page)

```
}
```

Configuration impact

All yang models related to the current RSP creation via RPC will be deprecated in Oxygen and removed in Fluorine. It will now be possible to create the RSP in the configuration data store. The "config false" flag will be removed from the RSP data model, thus allowing it to be created in the Config data store.

Although the RSP creation via RPC will be deprecated in the Oxygen release, it will still be supported until Fluorine. Once this change is implemented, the preferred way of creating RSPs will be via a write to the Config Data Store.

Clustering considerations

Currently RSPs support clustering, which will not be affected by this change.

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

With this change, there will be an additional write to the data store for each RSP creation. Considering there shouldnt be many RSPs created (typically less than 100) the impacts should be negligible.

Targeted Release

This feature is targeted for the Oxygen release.

Alternatives

None

1.2.3 Usage

Features to Install

All changes will be in the following existing Karaf features:

- odl-sfc-model
- odl-sfc-provider

REST API

The following JSON shows how an SFP is created, which will trigger the creation of the RSPs in both the Configuration and Operational data store.

```
URL: http://localhost:8181/config/service-function-path:rendered-service-paths/

{
  "service-function-paths": {
    "service-function-path": [
      {
        "name": "sfp1",
        "service-chain-name": "sfc1",
        "transport-type": "service-locator:vxlan-gpe",
        "symmetric": true
      }
    ]
  }
}
```

CLI

A new Karaf Shell command will be added to list the RSPs.

1.2.4 Implementation

Assignee(s)

Primary assignee:

- Brady Johnson, #ebrjohn, bradyallenjohnson@gmail.com

Other contributors:

- David Suárez, #edavsua, david.suarez.fuentes@gmail.com

Work Items

- Deprecate existing RSP RPC creation yang models.
- Deprecate existing RSP RPC Java classes and/or methods.
- Modify existing RSP data model "config false" values:
  - The entire RSP data model should no longer be "config false".
  - Mark those RSP data model leaf nodes as "config false" that will only be in operational.
- Create SFP configuration data store listener.
- Create RSP configuration data store listener.
- Create code to reject updates to the RSP configuration data store.
- Copy and retrofit existing code that writes RSPs to operational via RPCs to do so via the RSP configuration listener instead of via RPC.
- Create Karaf Shell CLI command to list RSPs in the config and operational data stores.

1.2.5 Dependencies

The following projects currently depend on SFC, and will be affected by this change:

- GBP
- Netvirt

1.2.6 Testing

Unit Tests

- The RSP creation in the existing UT will need to be updated as a result of this change.
- UT will need to be added to test RSP creation.

Integration Tests

None

CSIT

The RSP creation in the existing CSIT tests will need to be updated as a result of this change.

1.2.7 Documentation Impact

Both the User Guide and Developer Guide will need to be updated by the current ODL SFC Documentation contact: David Suárez.

1.2.8 References

[1] OpenDaylight Documentation Guide

Table of Contents

1.3 RSP Statistics

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

This feature introduces functionality to provide statistics for Rendered Service Paths (RSPs) defined with the SFC OpenFlow renderer.

1.3.1 Problem description

Currently there is no way to easily determine how many packets and bytes have traversed the different RSPs created with SFC. Its possible to obtain this information by looking at the SFC OpenFlow flows, but this would require a detailed knowledge of the flows.

Use Cases

Provide input and output bytes and packets for each RSP.

1.3.2 Proposed change

The RSP statistics will be collected by retrieving the OpenFlow statistics counters from certain OpenFlow flow entries created by SFC. The RSP statistics will be the following:

- RSP bytes in

- RSP bytes out
- RSP packets in
- RSP packets out

The RSP bytes/packets out can only be less than or equal to the bytes/packets in. The RSP bytes/packets out will be less than the bytes/packets in if the packets are dropped by either the SF or SFF.

The RSP statistics will be collected by retrieving the SFC OpenFlow NextHop flow entry from the first-hop SFF in the RSP. The RSP out statistics will be collected by retrieving the SFC OpenFlow NextHop flow entry from the last-hop SFF in the RSP. The flows will be retrieved based on their flow name.

RSP statistics collection will be provided on-demand by issuing a Northbound RPC RESTconf command, specifying the RSP to query. It will also be possible to query the statistics for all RSPs using the same RPC call. If the RSP specified in the RPC does not exist, then an RPC error message will be returned. Otherwise the packets and bytes in and out will be returned indicating the traffic that has gone through the RSP since its been created.

Using this on-demand approach, if time-series RSP statistics is needed, it can be handled externally to ODL, with a proper database since the MD-SAL is not an appropriate place to store time-series data.

A new SFC Karaf shell command will be created allowing RSP statistics to be retrieved via the Karaf shell.

Pipeline changes

This change will not require the SFC OpenFlow pipeline to be changed.

Yang changes

Although this feature only covers RSP statistics, the following YANG model also includes operations for Service Function (SF) and Service Function Forwarder (SFF) statistics collection.

```
module sfc-statistics-operations {

  namespace "urn:inocybe:params:xml:ns:yang:sfc-stats-ops";
  prefix sfc-stats-ops;

  import service-statistics {
      prefix sfc-ss;
      revision-date 2014-07-01;
  }

  organization "Inocybe, Inc.";
  contact "Brady Johnson <bjohnson@inocybe.com>";

  description
    "This module contains RPC operations to collect SFC statistics";

  revision 2017-12-15 {
    description
      "Initial Revision";
  }

  rpc get-rsp-statistics {
    description
      "Requests statistics for the specified Rendered Service Path";
    input {
      leaf name {
```

(continues on next page)

(continued from previous page)

```
        type string;
        description
          "The name of the Rendered Service Path. Leaving the
           name empty will return statistics for all Rendered
           Service Paths.";
      }
    }
    output {
      list statistics {
        leaf name {
          type string;
          description
            "The name of the Rendered Service Path.";
        }
        uses sfc-ss:service-statistics-group {
          description "Rendered Service Path statistics";
        }
      }
    }
  }

  rpc get-sff-statistics {
    description
      "Requests statistics for the specified Service Function Forwarder";
    input {
      leaf name {
        type string;
        description
          "The name of the Service Function Forwarder. Leaving
           the name empty will return statistics for all Service
           Function Forwarders.";
      }
    }
    output {
      list statistics {
        leaf name {
          type string;
          description
            "The name of the Service Function Forwarder.";
        }
        uses sfc-ss:service-statistics-group {
          description "Service Function Forwarder statistics";
        }
      }
    }
  }

  rpc get-sf-statistics {
    description
      "Requests statistics for the specified Service Function";
    input {
      leaf name {
        type string;
        description
          "The name of the Service Function. Leaving the
           name empty will return statistics for all
           Service Functions.";
```

(continues on next page)

(continued from previous page)

```
      }
    }
    output {
      list statistics {
        leaf name {
          type string;
          description
            "The name of the Service Function.";
        }
        uses sfc-ss:service-statistics-group {
          description "Service Function statistics";
        }
      }
    }
  }

}
```

Configuration impact

There will be no configuration impacts as a result of this feature.

Clustering considerations

The RSP statistics feature will not affect clustering, and will work with no problems in an ODL cluster

Other Infra considerations

N/A

Security considerations

N/A

Scale and Performance Impact

Since this will be an on-demand statistics request, there will be no scale and performance impacts.

Targeted Release

This feature is targeted to be implemented in the Oxygen release.

Alternatives

N/A

1.3.3 Usage

Nothing special needs to be done to use this feature, as it will be an on-demand request via the Northbound RPC RESTConf.

Features to Install

A new Karaf feature will be created called odl-sfc-statistics. No other existing SFC Karaf features will depend on this new feature.

REST API

The following example shows the new SFC statistics RPC definitions:

```
URL: http://localhost:8181/operations/sfc-statistics-operations:get-rsp-statistics

{
  "input": {
    "name": "RSP-1sf1sff"
  }
}

{
  "output": {
    "statistics" : [
      {
        "name": "RSP-1sf1sff",
        "statistic-by-timestamp": [
          {
            "service-statistic": {
              "bytes-in": 0,
              "bytes-out": 0,
              "packets-in": 0,
              "packets-out": 0
            },
            "timestamp": 1512418230327
          }
        ]
      }
    ]
  }
}
```

CLI

A new Karaf CLI will be added to retrieve RSP statistics. The syntax will similar to the following. Leaving the RSP name empty will return the statistics for all RSPs.

- sfc:rsp-statistics [RSP-name]

1.3.4 Implementation

Assignee(s)

Primary assignee: <Brady Johnson>, <ebrjohn>, <bjohnson@inocybe.com>

Work Items

Break up work into individual items. This should be a checklist on a Trello card for this feature. Provide the link to the trello card or duplicate it.

- Create the SFC statistics collection RPC data model.
- Create an RSP statistics collection handler that will retrieve the relevant OpenFlow flows and return the results.
- Create the necessary utils to assist the RSP handler in getting the flows and storing the results.
- Create the new odl-sfc-statistics Karaf feature.
- Create the Karaf shell command to retrieve the statistics.

1.3.5 Dependencies

No external projects will depend on this new feature. Nor will any additional dependencies on other ODL project be introduced.

1.3.6 Testing

Capture details of testing that will need to be added.

Unit Tests

A new Unit Test will be added for each of the new Java classes added.

Integration Tests

N/A

CSIT

A new test case will be added to CSIT for this feature. The test should inject packets and will verify that the RSP statistics counters are incremented as expected.

1.3.7 Documentation Impact

The User Guide will be updated to show how to use this new feature.

1.3.8 References

N/A

Table of Contents

1.4 Karaf Command Line Interface (CLI) for SFC

[S: https://git.opendaylight.org/gerrit/#/q/topic:sfc-shell]

The Karaf Container offers a very complete Unix-like console that allows managing the container. This console can be extended with custom commands to manage the features deployed on it. This feature will add some basic commands to show the provisioned SFC's entities.

1.4.1 Problem description

This feature will implement commands to show some of the provisioned SFC's entities:

- Service Functions
- Service Function Forwarders
- Service Function Chains
- Service Function Paths
- Service Function Classifiers
- Service Nodes
- Service Function Types

Use Cases

- Use Case 1: list one/all provisioned Service Functions.
- Use Case 2: list one/all provisioned Service Function Forwarders.
- Use Case 3: list one/all provisioned Service Function Chains.
- Use Case 4: list one/all provisioned Service Function Paths.
- Use Case 5: list one/all provisioned Service Function Classifiers.
- Use Case 6: list one/all provisioned Service Nodes.
- Use Case 7: list one/all provisioned Service Function Types.

1.4.2 Proposed change

Details of the proposed change.

Pipeline changes

None

Yang changes

None

Configuration impact

None

Clustering considerations

None

Other Infra considerations

Creation of new commands for the Karaf's console.

Security considerations

None

Scale and Performance Impact

None

Targeted Release

Nitrogen

Alternatives

None

1.4.3 Usage

The feature will add CLI commands to the Karaf's console to list some of the provisioned SFC's entities. See the CLI section for details about the syntax of those commands.

Features to Install

odl-sfc-shell

REST API

None

CLI

- UC 1: list one/all provisioned Service Functions.

  sfc:sf-list [–name <name>]
- UC 2: list one/all provisioned Service Function Forwarders.

  sfc:sff-list [–name <name>]
- UC 3: list one/all provisioned Service Function Chains.

  sfc:sfc-list [–name <name>]
- UC 4: list one/all provisioned Service Function Paths.

  sfc:sfp-list [–name <name>]
- UC 5: list one/all provisioned Service Function Classifiers.

  sfc:sc-list [–name <name>]
- UC 6: list one/all provisioned Service Nodes.

  sfc:sn-list [–name <name>]
- UC 7: list one/all provisioned Service Function Types.

  sfc:sft-list [–name <name>]

1.4.4 Implementation

Assignee(s)

Primary assignee: David Suárez, #edavsua, david.suarez.fuentes@gmail.com Brady Johson, #ebrjohn, bradyallen-johnson@gmail.com

Work Items

- Implement UC 1: list one/all provisioned Service Functions.
- Implement UC 2: list one/all provisioned Service Function Forwarders.
- Implement UC 3: list one/all provisioned Service Function Chains.
- Implement UC 4: list one/all provisioned Service Function Paths.
- Implement UC 5: list one/all provisioned Service Function Classifiers.
- Implement UC 6: list one/all provisioned Service Nodes.
- Implement UC 7: list one/all provisioned Service Types.

1.4.5 Dependencies

This feature uses the new Karaf 4.x API to create CLI commands.

No changes needed on projects depending on SFC.

1.4.6 Testing

Capture details of testing that will need to be added.

Unit Tests

Integration Tests

CSIT

None

1.4.7 Documentation Impact

The new CLI for SFC will be documented in both the User and Developer guides.

1.4.8 References

Add any useful references. Some examples:

- https://docs.google.com/presentation/d/1RKkJsTUF65t40ASXVztNMcKAxMzI_owyZ-c6Mpm4Ss8/edit?usp=sharing

[1] OpenDaylight Documentation Guide

Table of Contents

1.5 Directional data plane locators

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

This specification proposes to have an optional direction associated with data plane locators.

1.5.1 Problem description

Service functions that are not SFC encapsulation aware need to use alternative mechanisms to recognize and apply SFC features. One common alternative is to use multiple interfaces. For example, each interface could be associated exclusively to the ingress traffic of a specific service function path.

Service functions for which this concept will most commonly apply are those network devices that operate as 'bump in the wire', where two interfaces acting as ingress and egress interfaces for traffic in one direction, would be the egress and ingress interfaces respectively in the opposite direction.

Currently, the connection between a service function and a service function forwarder is described by single pair of data plane locators, one for each side of the connection. For symmetric service chains, the same locator pair is used when rendering the forward and reverse paths. This prevents the possibility to apply the mechanism explained previously.

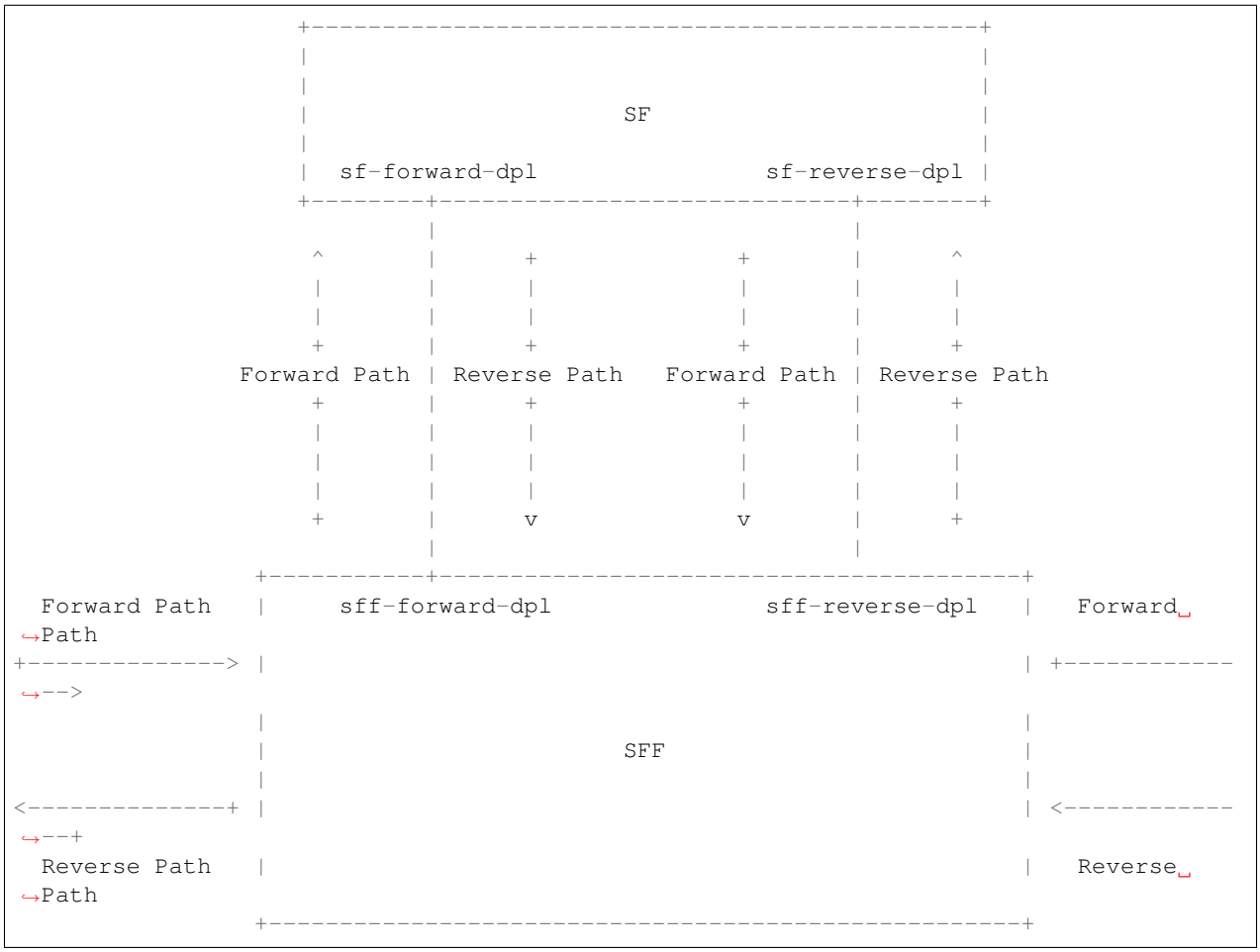
The purpose of this specification is to overcome this first limitation, enabling the support of bump in the wire service functions that are sfc aware, understood as service functions that allow to employ inline mechanisms to steer traffic (mac chaining, nsh, vlan, mpls...), versus bump in the wire service functions that are sfc unaware and thus might not allow any modification over the source traffic.

Use Cases

Support 'bump in the wire' network devices that can be made sfc aware to act as service functions.

1.5.2 Proposed change

The proposed solution is to add the flexibility to configure two pairs of data plane locators to describe the association between the service function and service function forwarder. Each pair will be assigned a direction, either forward or reverse, and will be used as ingress for paths on the same direction and as egress from paths on the opposite direction. The forward pair will be used to ingress traffic to the service function on forward paths and for the egress traffic from the service function on reverse paths. The reverse pair will be used for the egress traffic from the service function for the forward path and to ingress the traffic to the service function for the reverse path. This can seen in the following figure:



The SFF-SF dictionary will be expanded to accommodate two pairs of locators as depicted in *Yang changes*. Renderers that support this feature shall give precedence to the new directional locators over the old locators. Once all renderers support the feature, the single locator pair may be deprecated or may remain as a simple way to configure a bidirectional locator.

Logical SFF configuration model needs to change as it does not currently use the SFF-SF dictionary. Logical SFF configuration model uses a placeholder service function forwarder configuration. The proposal is to align the logical SFF configuration model with a standard SFC configuration, otherwise two different models will need to be continuously proposed for every other feature, causing more divergence over time. As depicted in *REST API*, logical interfaces shall be configured as locators both on the SF and the SFF side.

The implementation targets the openflow renderer. The openflow processor will provide to the transport specific processor the appropriate data plane locator pair based on the direction of the path being rendered.

Also in Logical SFF context, service binding will be performed on the service function forwarder logical interfaces as

soon as intervenes on a path. On egress towards the service function, egress actions will be requested from genius for the interface provided by the openflow processor.

It is worth mentioning that the proposed change may not be enough to fully support legacy 'bump in the wire' network devices that are sfc unaware acting as service functions. For this, it might be additionally needed to:

- Provide the service function with the original unmodified source traffic.
- And as a consequence, on service function egress, reclassify the traffic to a path based on the service function forwarder ingress port.
- And as a consequence, avoid using that port as ingress for more than one path.

Directional data plane locators is a step towards 'bump in the wire' full support and useful in itself for those service functions that while operating in this mode, are sfc aware in that they allow to use already supported mechanisms (mac chaining, nsh...) to steer SFC traffic.

Pipeline changes

The existing OpenFlow pipeline will not be affected by this change.

Yang changes

The following data model is the updated service-function-dictionary within the service function forwarder.

Listing 3: service-function-forwarder.yang

```
    list service-function-dictionary {
        key "name";
        leaf name {
          type sfc-common:sf-name;
          description
              "The name of the service function.";
        }
        container sff-sf-data-plane-locator {
          description
            "SFF and SF data plane locators to use when sending
             packets from this SFF to the associated SF";
          leaf sf-dpl-name {
            type sfc-common:sf-data-plane-locator-name;
            description
              "The SF data plane locator to use when sending
               packets to the associated service function.
               Used both as forward and reverse locators for
               paths of a symmetric chain.";
          }
          leaf sff-dpl-name {
            type sfc-common:sff-data-plane-locator-name;
            description
              "The SFF data plane locator to use when sending
               packets to the associated service function.
               Used both as forward and reverse locators for
               paths of a symmetric chain.";
          }
          leaf sf-forward-dpl-name {
            type sfc-common:sf-data-plane-locator-name;
            description
```

(continues on next page)

(continued from previous page)

```
                "The SF data plane locator to use when sending
                 packets to the associated service function
                 on the forward path of a symmetric chain";
            }
            leaf sf-reverse-dpl-name {
              type sfc-common:sf-data-plane-locator-name;
              description
                "The SF data plane locator to use when sending
                 packets to the associated service function
                 on the reverse path of a symmetric chain";
            }
            leaf sff-forward-dpl-name {
              type sfc-common:sff-data-plane-locator-name;
              description
                "The SFF data plane locator to use when sending
                 packets to the associated service function
                 on the forward path of a symmetric chain.";
            }
            leaf sff-reverse-dpl-name {
              type sfc-common:sff-data-plane-locator-name;
              description
                "The SFF data plane locator to use when sending
                 packets to the associated service function
                 on the reverse path of a symmetric chain.";
            }
          }
      }
```

Logical interface locator support is also added to the service function forwarder data plane locator.

Listing 4: service-function-forwarder-logical.yang

```
    augment "/sfc-sff:service-function-forwarders/"
          + "sfc-sff:service-function-forwarder/"
          + "sfc-sff:sff-data-plane-locator/"
          + "sfc-sff:data-plane-locator/"
          + "sfc-sff:locator-type/" {
      description "Augments the Service Function Forwarder to allow the use of␣
↪logical
                   interface locators";
      case logical-interface {
        uses logical-interface-locator;
      }
    }
```

A new leaf is added to the rendered service path model to flag reverse paths.

Listing 5: rendered-service-path.yang

```
    leaf reverse-path {
      type boolean;
      mandatory true;
      description
        "True if this path is the reverse path of a symmetric
         chain.";
    }
```

Configuration impact

New optional parameters are added to the SFF-SF dictionary. These new parameters may not be configured in which case behavior is not changed.

The new flag introduced in the rendered service path model does not have configuration impact as the entity is not meant to be configured.

Logical SFF configuration model will change. Both, previous and new configuration models will be supported.

Thus backward compatibility is preserved despite the introduced changes.

Clustering considerations

Clustering support will not be affected by this change.

Other Infra considerations

None.

Security considerations

None.

Scale and Performance Impact

None.

Targeted Release

This feature is targeted for the Oxygen release.

Alternatives

One first consideration is that if one SF interface required two data plane locators, two SF interfaces is going to require four data plane locators to be fully described, specially considering a scenario where we would like to explicitly configure different openflow ports on the SFF side for each direction. The proposed solution leverages the fact that the data plane locators are already contained in lists on both the SFF and the SF.

One alternative is to introduce a new locator type that serves as an indirection through which the names of forward and reverse locators can be specified. Thus three locators are required total for each side of the SFF-SF association:

one forward locator, one reverse locator and the new locator that tells which is which, whose name would be used in the SFF-SF dictionary. The advantage is that the SFF configuration model needs not to be changed. As disadvantages, it needs an extra data plane locator on each side, it might be confusing being able to specify different SFF names and transport types between the three locators on SF side, and finally, the indirection overall leads to a less explicit model/api and code wise it would probably require to hide all locator checks or manipulation behind helper code.

Another option is to expand the key of the SFF-SF dictionary to include direction so that two dictionary entries can be specified for each SFF/SF pair. This was discarded because is not backward compatible.

1.5.3 Usage

Features to Install

All changes will be in the following existing Karaf features:

- odl-sfc-genius
- odl-sfc-openflow-renderer

REST API

The following JSON shows how the service function and service function forwarder are configured in the context of Logical SFF with directional locators.

```
URL: http://localhost:8181/restconf/config/service-function:service-functions/

{
  "service-functions": {
    "service-function": [
      {
        "name": "firewall-1",
        "type": "firewall",
        "sf-data-plane-locator": [
          {
            "name": "firewall-ingress-dpl",
            "interface-name": "eccb57ae-5a2e-467f-823e-45d7bb2a6a9a",
            "transport": "service-locator:mac",
            "service-function-forwarder": "sfflogical1"
          },
          {
            "name": "firewall-egress-dpl",
            "interface-name": "df15ac52-e8ef-4e9a-8340-ae0738aba0c0",
            "transport": "service-locator:mac",
            "service-function-forwarder": "sfflogical1"
          }
        ]
      }
    ]
  }
}
```

```
URL: http://localhost:8181/restconf/config/service-function-forwarder:service-
↪function-forwarders/

{
```

(continues on next page)

(continued from previous page)

```
  "service-function-forwarders": {
    "service-function-forwarder": [
      {
        "name": "sfflogical1",
        "sff-data-plane-locator": [
          {
            "name": "firewall-ingress-dpl",
            "data-plane-locator": {
              "interface-name": "df15ac52-e8ef-4e9a-8340-ae0738aba0c0",
              "transport": "service-locator:mac"
            }
          },
          {
            "name": "firewall-egress-dpl",
            "data-plane-locator": {
              "interface-name": "eccb57ae-5a2e-467f-823e-45d7bb2a6a9a",
              "transport": "service-locator:mac"
            }
          }
        ],
        "service-function-dictionary": [
          {
            "name": "firewall-1",
            "sff-sf-data-plane-locator":
            {
              "sf-forward-dpl-name": "firewall-ingress-dpl",
              "sf-reverse-dpl-name": "firewall-egress-dpl",
              "sff-forward-dpl-name": "firewall-egress-dpl",
              "sff-reverse-dpl-name": "firewall-ingress-dpl",
            }
          }
        ]
      }
    ]
  }
}
```

CLI

No new CLI commands will be added but the existing ones will be enhanced to display more details about the associations between service function and service function forwarders.

1.5.4 Implementation

Assignee(s)

Primary assignee:

- Jaime Caamaño, #jaicaa, jcaamano@suse.com

Work Items

- Update the service function forwarder yang model.

- Update odl-sfc-genius to bind on service function forwarder interfaces.
- Update odl-openflow-renderer processor and surrounding utilities to use the proper data plane locator based on the direction of the path.
- Update provider to set the reverse flag on reverse rendered service paths.
- Update the shell command for service functions and service function forwarders to display the associations between them.
- Update CSIT Full Deploy to use new Logical SFF configuration model.
- Update the user & developer guide to document directional data plane locators.
- Update the user & developer guide to reflect the new Logical SFF configuration model.

1.5.5 Dependencies

The following projects currently depend on SFC:

- GBP
- Netvirt

No backward incompatible changes are introduced but these projects, as neutron implementations, are target users for the new feature in Logical SFF scenario.

1.5.6 Testing

Unit Tests

Unit tests will be added for new code introduced through this feature.

Integration Tests

None.

CSIT

Existing Full Deploy CSIT will be updated to use the new Logical SFF configuration model.

1.5.7 Documentation Impact

Both the User Guide and Developer Guide will need to be updated.

1.5.8 Open Issues

- Drop support for the old Logical SFF configuration model?
- New CSIT tests not proposed yet because it requires testing with traffic, which we don't currently have and is a major undertaking on itself.

1.5.9 References

NA