

---

# ODL NetVirt

*Release master*

**Apr 16, 2018**



---

## Contents

---

<b>1</b>	<b>NetVirt Contributor Guide</b>	<b>3</b>
<b>2</b>	<b>NetVirt Developer Guide</b>	<b>273</b>
<b>3</b>	<b>NetVirt Installation Guide</b>	<b>275</b>
<b>4</b>	<b>OpenStack with NetVirt</b>	<b>277</b>
<b>5</b>	<b>NetVirt User Guide</b>	<b>289</b>
	<b>Bibliography</b>	<b>299</b>



This documentation provides critical information needed to help you write code for the NetVirt project.

Contents:



### 1.1 NetVirt Design Specifications

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

Contents:

#### Table of Contents

- *Title of the feature*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*
  - *Usage*

- \* *Features to Install*
- \* *REST API*
- \* *CLI*
- *Implementation*
  - \* *Assignee(s)*
  - \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.1 Title of the feature

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

Brief introduction of the feature.

#### Problem description

Detailed description of the problem being solved by this feature

#### Use Cases

Use cases addressed by this feature.

#### Proposed change

Details of the proposed change.

#### Pipeline changes

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

#### Yang changes

This should detail any changes to yang models.



Listing 1.1: example.yang

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

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

  description "An example YANG model.";

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

### Configuration impact

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

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

### Clustering considerations

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

### Other Infra considerations

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

### Security considerations

Document any security related issues impacted by this feature.

### Scale and Performance Impact

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

### Targeted Release

What release is this feature targeted for?

### Alternatives

Alternatives considered and why they were not selected.

### Usage

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

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

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

### Features to Install

odl-netvirt-openstack

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

### REST API

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

### CLI

Any CLI if being added.

### Implementation

#### Assignee(s)

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

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

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

### Work Items

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

### Dependencies

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

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

**Following projects currently depend on Netvirt:** Unimgr

## Testing

Capture details of testing that will need to be added.

## Unit Tests

## Integration Tests

## CSIT

## Documentation Impact

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

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

## References

Add any useful references. Some examples:

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

[1] [OpenDaylight Documentation Guide](#)

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

---

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

This work is licensed under a Creative Commons Attribution 3.0 Unported License. <http://creativecommons.org/licenses/by/3.0/legalcode>

---

### Table of Contents

- *ACL Statistics*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *ACL Changes*
    - \* *Drop packets statistics support*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*

- \* *Clustering considerations*
- \* *Other Infra considerations*
- \* *Security considerations*
- \* *Scale and Performance Impact*
- \* *Targeted Release*
- \* *Alternatives*
- *Usage*
  - \* *Features to Install*
  - \* *REST API*
  - \* *CLI*
- *Implementation*
  - \* *Assignee(s)*
  - \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.2 ACL Statistics

<https://git.opendaylight.org/gerrit/#/q/topic:acl-stats>

This feature is to provide additional operational support for ACL through statistical counters. ACL rules provide security to VMs by filtering packets in either directions (ingress/egress). Using OpenFlow statistical counters, ODL will provide additional information on the number of packets dropped by the ACL rules. This information is made available to the operator “on demand”.

Drop statistics will be provided for below cases:

- Packets dropped due to ACL rules
- Packets dropped due to INVALID state. The INVALID state means that the packet can’t be identified or that it does not have any state. This may be due to several reasons, such as the system running out of memory or ICMP error messages that do not respond to any known connections.

The packet drop information provided through the statistical counters enable operators to trouble shoot any misbehavior and take appropriate actions through automated or manual intervention.

Collection and retrieval of information on the number of packets dropped by the SG rules

- Done for all (VM) ports in which SG is configured
- Flow statistical counters (in OpenFlow) are used for this purpose

- The information in these counters are made available to the operator, on demand, through an API

This feature will only be supported with Stateful ACL mode.

## Problem description

With only ACL support, operators would not be able to tell how many packets dropped by ACL rules. This enhancement planned is about ACL module supporting aforementioned limitation.

## Use Cases

Collection and retrieval of information on the number of packets dropped by the ACL rules

- Done for all (VM) ports in which ACL is configured
- The information in these counters are made available to the operator, on demand, through an API
- Service Orchestrator/operator can also specify ports selectively where ACL rules are configured

## Proposed change

### ACL Changes

Current Stateful ACL implementation has drop flows for all ports combined for a device. This needs to be modified to have drop flows for each of the OF ports connected to VMs (Neutron Ports).

With the current implementation, drop flows are as below:

```
cookie=0x6900000, duration=938.964s, table=252, n_packets=0, n_bytes=0,
→priority=62020,
    ct_state=+inv+trk actions=drop

cookie=0x6900000, duration=938.969s, table=252, n_packets=0, n_bytes=0, priority=50,
    ct_state=+new+trk actions=drop
```

Now, for supporting Drop packets statistics per port, ACL will be updated to replace above flows with new DROP flows with lport tag as metadata for each of the VM (Neutron port) being added to OVS as specified below:

```
cookie=0x6900001, duration=938.964s, table=252, n_packets=0, n_bytes=0,
→priority=62015,
    metadata=0x10000000000/0xffffffff0000000000, ct_state=+inv+trk actions=drop

cookie=0x6900001, duration=938.969s, table=252, n_packets=0, n_bytes=0, priority=50,
    metadata=0x10000000000/0xffffffff0000000000, ct_state=+new+trk actions=drop
```

Drop flows details explained above are for pipeline egress direction. For ingress side, similar drop flows would be added with table=41.

Also, new cookie value 0x6900001 would be added with drop flows to identify it uniquely and priority 62015 would be used with +inv+trk flows to give higher priority for +est and +rel flows.

## Drop packets statistics support

ODL Controller will be updated to provide a new RPC/NB REST API <get-acl-port-statistics> in ACL module with ACL Flow Stats Request and ACL Flow Stats Response messages. This RPC/API will

retrieve details of dropped packets by Security Group rules for all the neutron ports specified as part of ACL Flow Stats Request. The retrieved information (instantaneous) received in the OF reply message is formatted as ACL Flow Stats Response message before sending it as a response towards the NB.

<get-acl-port-statistics> RPC/API implementation would be triggering `opendaylight-direct-statistics:get-flow-statistics` request of OFPlugin towards OVS to get the flow statistics of ACL tables (ingress / egress) for the required ports.

ACL Flow Stats Request/Response messages are explained in subsequent sections.

## Pipeline changes

No changes needed in OF pipeline. But, new flows as specified in above section would be added for each of the Neutron ports being added.

## Yang changes

New yang file will be created with RPC as specified below:

Listing 1.2: acl-live-statistics.yang

```
module acl-live-statistics {
  namespace "urn:opendaylight:netvirt:acl:live:statistics";

  prefix "acl-stats";

  import ietf-interfaces {prefix if;}
  import aclservice {prefix aclservice; revision-date "2016-06-08";}

  description "YANG model describes RPC to retrieve ACL live statistics.";

  revision "2016-11-29" {
    description "Initial revision of ACL live statistics";
  }

  typedef direction {
    type enumeration {
      enum ingress;
      enum egress;
      enum both;
    }
  }

  grouping acl-drop-counts {
    leaf drop-count {
      description "Packets/Bytes dropped by ACL rules";
      type uint64;
    }
    leaf invalid-drop-count {
      description "Packets/Bytes identified as invalid";
      type uint64;
    }
  }

  grouping acl-stats-output {
    description "Output for ACL port statistics";
```

```

    list acl-interface-stats {
        key "interface-name";
        leaf interface-name {
            type leafref {
                path "/if:interfaces/if:interface/if:name";
            }
        }
        list acl-drop-stats {
            max-elements "2";
            min-elements "0";
            leaf direction {
                type identityref {
                    base "aclservice:direction-base";
                }
            }
            container packets {
                uses acl-drop-counts;
            }
            container bytes {
                uses acl-drop-counts;
            }
        }
        container error {
            leaf error-message {
                type string;
            }
        }
    }
}

grouping acl-stats-input {
    description "Input parameters for ACL port statistics";

    leaf direction {
        type identityref {
            base "aclservice:direction-base";
        }
        mandatory "true";
    }
    leaf-list interface-names {
        type leafref {
            path "/if:interfaces/if:interface/if:name";
        }
        max-elements "unbounded";
        min-elements "1";
    }
}

rpc get-acl-port-statistics {
    description "Get ACL statistics for given list of ports";

    input {
        uses acl-stats-input;
    }
    output {
        uses acl-stats-output;
    }
}

```

}

### Configuration impact

No configuration parameters being added/deprecated for this feature

### Clustering considerations

No additional changes required to be done as only one RPC is being supported as part of this feature.

### Other Infra considerations

N.A.

### Security considerations

N.A.

### Scale and Performance Impact

N.A.

### Targeted Release

Carbon

### Alternatives

Dispatcher table (table 17 and table 220) based approach of querying drop packets count was considered. ie., arriving drop packets count by below rule:

**<total packets entered ACL tables> - <total packets entered subsequent service>**

This approach was not selected as this only provides total packets dropped count per port by ACL services and does not provide details of whether it's dropped by ACL rules or for some other reasons.

### Usage

### Features to Install

odl-netvirt-openstack



## REST API

### Get ACL statistics

Following API gets ACL statistics for given list of ports.

**Method:** POST

**URI:** /operations/acl-live-statistics:get-acl-port-statistics

**Parameters:**

Parameter	Type	Possible Values	Comments
"direction"	Enum	ingress/egress/both	Required
"interface-names"	Array [UUID String]	[<UUID String>,<UUID String>,.. ]	Required (1,N)

**Example:**

```
{
  "input": {
    "direction": "both",
    "interface-names": [
      "4ae8cd92-48ca-49b5-94e1-b2921a2661c5",
      "6c53df3a-3456-11e5-a151-feff819cdc9f"
    ]
  }
}
```

**Possible Responses:**

**RPC Success:**

```
{
  "output": {
    "acl-port-stats": [
      {
        "interface-name": "4ae8cd92-48ca-49b5-94e1-b2921a2661c5",
        "acl-drop-stats": [
          {
            "direction": "ingress",
            "bytes": {
              "invalid-drop-count": "0",
              "drop-count": "300"
            },
            "packets": {
              "invalid-drop-count": "0",
              "drop-count": "4"
            }
          },
          {
            "direction": "egress",
            "bytes": {
              "invalid-drop-count": "168",
              "drop-count": "378"
            },
            "packets": {
              "invalid-drop-count": "2",

```

```
        "drop-count": "9"
      }
    ]
  },
  {
    "interface-name": "6c53df3a-3456-11e5-a151-feff819cdc9f",
    "acl-drop-stats": [
      {
        "direction": "ingress",
        "bytes": {
          "invalid-drop-count": "1064",
          "drop-count": "1992"
        },
        "packets": {
          "invalid-drop-count": "18",
          "drop-count": "23"
        }
      },
      {
        "direction": "egress",
        "bytes": {
          "invalid-drop-count": "462",
          "drop-count": "476"
        },
        "packets": {
          "invalid-drop-count": "11",
          "drop-count": "6"
        }
      }
    ]
  }
]
```

**RPC Success (with error for one of the interface):**

```
{
  "output":
  {
    "acl-port-stats": [
      {
        "interface-name": "4ae8cd92-48ca-49b5-94e1-b2921a2661c5",
        "acl-drop-stats": [
          {
            "direction": "ingress",
            "bytes": {
              "invalid-drop-count": "0",
              "drop-count": "300"
            },
            "packets": {
              "invalid-drop-count": "0",
              "drop-count": "4"
            }
          },
          {
            "direction": "egress",
            "bytes": {
              "invalid-drop-count": "168",
              "drop-count": "378"
            },
            "packets": {
              "invalid-drop-count": "0",
              "drop-count": "0"
            }
          }
        ]
      }
    ]
  }
}
```

```

        "packets": {
            "invalid-drop-count": "2",
            "drop-count": "9"
        },
        {
            "interface-name": "6c53df3a-3456-11e5-a151-feff819cdc9f",
            "error": {
                "error-message": "Interface not found in datastore."
            }
        }
    ]
}

```

**Note:** Below are error messages for the interface:

1. "Interface not found in datastore."
2. "Failed to find device for the interface."
3. "Unable to retrieve drop counts due to error: <<error message>>"
4. "Unable to retrieve drop counts as interface is not configured for statistics collection."
5. "Operation not supported for ACL <<Stateless/Transparent/Learn>> mode"

## CLI

No CLI being added for this feature

## Implementation

### Assignee(s)

**Primary assignee:** <Somashekar Byrappa>

**Other contributors:** <Shashidhar R>

### Work Items

1. Adding new drop rules per port (in table 41 and 252)
2. Yang changes
3. Supporting new RPC

### Dependencies

This doesn't add any new dependencies.

This feature has dependency on below bug reported in OF Plugin:

[Bug 7232 - Problem observed with "get-flow-statistics" RPC call](#)

## Testing

### Unit Tests

Following test cases will need to be added/expanded

1. Verify ACL STAT RPC with single Neutron port
2. Verify ACL STAT RPC with multiple Neutron ports
3. Verify ACL STAT RPC with invalid Neutron port
4. Verify ACL STAT RPC with mode set to “transparent/learn/stateless”

Also, existing unit tests will be updated to include new drop flows.

### Integration Tests

Integration tests will be added, once IT framework is ready

### CSIT

Following test cases will need to be added/expanded

1. Verify ACL STAT RPC with single Neutron port with different directions (ingress, egress, both)
2. Verify ACL STAT RPC with multiple Neutron ports with different directions (ingress, egress, both)
3. Verify ACL STAT RPC with invalid Neutron port
4. Verify ACL STAT RPC with combination of valid and invalid Neutron ports
5. Verify ACL STAT RPC with combination of Neutron ports with few having port-security-enabled as true and others having false

### Documentation Impact

This will require changes to User Guide. User Guide needs to be updated with details about new RPC being supported and also about its REST usage.

### References

N.A.

---

**Note:** This work is licensed under a Creative Commons Attribution 3.0 Unported License. <http://creativecommons.org/licenses/by/3.0/legalcode>

---

#### Table of Contents

- *ACL Remote ACL - Indirection Table to Improve Scale*
  - *Problem description*

- \* *Use Cases*
- *Proposed change*
  - \* *Pipeline changes*
  - \* *Yang changes*
  - \* *Configuration impact*
  - \* *Clustering considerations*
  - \* *Other Infra considerations*
  - \* *Security considerations*
  - \* *Scale and Performance Impact*
  - \* *Targeted Release*
  - \* *Alternatives*
- *Usage*
  - \* *Features to Install*
  - \* *REST API*
  - \* *CLI*
- *Implementation*
  - \* *Assignee(s)*
  - \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.3 ACL Remote ACL - Indirection Table to Improve Scale

ACL Remote ACL Indirection patches: [https://git.opendaylight.org/gerrit/#/q/topic:remote\\_acl\\_indirection](https://git.opendaylight.org/gerrit/#/q/topic:remote_acl_indirection)

This spec is to enhance the initial implementation of ACL remote ACLs filtering which was released in Boron. The Boron release added full support for remote ACLs, however the current implementation does not scale well in terms of flows. The Carbon release will update the implementation to introduce a new indirection table for ACL rules with remote ACLs, to reduce the number of necessary flows, in cases where the port is associated with a single ACL. Due to the complication of supporting multiple ACLs on a single port, the current implementation will stay the same for these cases.

## Problem description

Today, for each logical port, an ACL rule results in a flow in the ACL table (ACL2). When a remote ACL is configured on this rule, this flow is multiplied for each VM in the remote ACL, resulting in a very large number of flows.

For example, consider we have:

- 100 computes
- 50 VMs on each compute (5000 VMs total),
- All VMs are in a SG (SG1)
- This SG has a security rule configured on it with remote SG=SG1 (it is common to set the remote SG as itself, to set rules within the SG).

This would result in  $50 * 5000 = 250,000$  flows on each compute, and 25M flows in ODL MDSAL (!).

## Use Cases

Neutron configuration of security rules, configured with remote SGs. This optimization will be relevant only when there is a single security group that is associated with the port. In case more than one security group is associated with the port - we will fallback to the current implementation which allows full functionality but with possible flow scaling issues.

Rules with a remote ACL are used to allow certain types of packets only between VMs in certain security groups. For example, configuring rules with the parent security group also configured as a remote security group, allows to configure rules applied only for traffic between VMs in the same security group.

This will be done in the ACL implementation, so any ACL configured with a remote ACL via a different northbound or REST would also be handled.

## Proposed change

This blueprint proposes adding a new indirection table in the ACL service in each direction, which will attempt to match the “remote” IP address associated with the packet (“dst\_ip” in Ingress ACL, “src\_ip” in Egress ACL), and set the ACL ID as defined by the ietf-access-control-list in the metadata. This match will also include the ELAN ID to handle ports with overlapping IPs.

These flows will be added to the ACL2 table. In addition, for each such ip->SG flow inserted in ACL2, we will insert a single SG metadata match in ACL3 for each SG rule on the port configured with this remote SG.

If the IP is associated with multiple SGs - it is impossible to do a 1:1 matching of the SG, so we will not set the metadata at this time and fallback to the current implementation of matching all possible IPs in the ACL table - for this ACL2 will have a default flow passing the unmatched packets to ACL3 with an empty metadata SG\_ID write (e.g. 0x0), to prevent potential garbage in the metadata SG ID.

This means that on transition from a single SG on the port to multiple SG (and back), we would need to remove/add these flows from ACL2, and insert the correct rules into ACL3.

### ACL1 (211/241):

- This is the ACL that has default allow rules - it is left untouched, and usually goes to ACL2.

### ACL2 (212/242):

- For each port with a single SG - we will match on the IPs and the ELAN ID (for tenant awareness) here, and set the SG ID in the metadata, before going to the ACL3 table.

- For any port with multiple SGs (or with no SG) - an empty value (0x0) will be set as the SG ID in the metadata, to avoid potential garbage in the SG ID, and goto ACL3 table.

#### ACL3 (213/243):

- For each security rule that *doesn't have* a remote SG, we keep the behavior the same: ACL3 matches on rule, and resubmits to dispatcher if there is a match (Allow). The SG ID in the metadata will **not** be matched.
- For each security rule that *does have* a remote SG, we have two options:
  - For ports belonging to the remote SG that are associated with a single SG - there will be a single flow per rule, matching the SG ID from the metadata (in addition to the other rule matches) and allowing the packet.
  - For ports belonging to the remote SG that are associated with multiple SGs - the existing implementation will stay the same, multiplying the rule with all possible IP matches from the remote security groups.

Considering the example from the problem description above, the new implementation would result in a much reduced number of flows:

5000+50 = 5050 flows on each compute, and 505,000 flows in ODL MDSAL.

As noted above, this would require using part of the metadata for writing/matching of an ACL ID. We would likely require at least 12 bits for this, to support up to 4K SGs, where 16 bits to support up to 65K would be ideal. If the metadata bits are not available, we can use a register for this purpose (16 bits).

In addition, the dispatcher will set the ELAN ID in the metadata before entering the ACL services, to allow tenant aware IP to SG detection, supporting multi-tenants with IP collisions.

#### Pipeline changes

ACL3 will be added, and the flows in ACL2/ACL3 will be modified as noted above in the proposed change:

Table	Match	Action
Dispatcher	metadata=service_id:ACL	write_metadata:(elan_id=ELAN, service_id=NEXT), goto_table:ACL1
ACL1 (211/241)	goto_table:ACL2	
ACL2 (212/242)	metadata=ELAN_ID, ip_src/dst=VM1_IP	write_metadata:(remote_acl=id), goto_table:ACL3
ACL2 (212/242)	metadata=ELAN_ID, ip_src/dst=VM2_IP	write_metadata:(remote_acl=id), goto_table:ACL3
...		
ACL2 (212/242)		goto_table:ACL3
ACL3 (213/243)	metadata=lport, <acl_rule>	resubmit(DISPATCHER) <sup>(X)</sup>
ACL3 (213/243)	metadata=lport+remote_acl, <acl_rule>	resubmit(DISPATCHER) <sup>(XX)</sup>
ACL3 (213/243)	metadata=lport,ip_src/dst=VM1_IP, <acl_rule>	resubmit(DISPATCHER) <sup>(XXX)</sup>
ACL3 (213/243)	metadata=lport,ip_src/dst=VM2_IP, <acl_rule>	resubmit(DISPATCHER) <sup>(XXX)</sup>
...		

(X) These are the regular rules, not configured with any remote SG.

(XX) These are the proposed rules with the optimization - assuming the lport is using a single ACL.

(XXX) These are the remote SG rules in the current implementation, which we will fall back to if the lport has multiple ACLs.

**Table Numbering:**

Currently the Ingress ACLs use tables *40,41,42* and the Egress ACLs use tables *251,252,253*.

Table 43 is already proposed to be taken by SNAT, and table 254 is considered invalid by OVS. To overcome this and align Ingress/Egress with symmetric numbering, I propose the following change:

- Ingress ACLs: 211, 212, 213, 214
- Egress ACLs: 241, 242, 243, 244

ACL1: INGRESS/EGRESS\_ACL\_TABLE ACL2: INGRESS/EGRESS\_ACL\_REMOTE\_ACL\_TABLE ACL3: INGRESS/EGRESS\_ACL\_FILTER\_TABLE

ACL4 is used only for Learn implementation for which an extra table is required.

**Yang changes**

None.

**Configuration impact**

None.

**Clustering considerations**

None.

**Other Infra considerations**

None.

**Security considerations**

None.

**Scale and Performance Impact**

See example in description. The scale of the flows will be drastically reduced when using remote ACLs.

**Targeted Release**

Carbon



## Alternatives

For fully optimized support in all scenarios for remote SGs, meaning including support for ports with multiple ACLs on them, we did consider implementing a similar optimization.

However, for this to happen due to OpenFlow limitations we would need to introduce an internal dispatcher inside the ACL services, meaning we loop the ACL service multiple times, each time setting a different metadata SG value for the port.

For another approach we could use a bitmask, but this would limit the number of possible SGs to be the number of bits in the mask, which is much too low for any reasonable use case.

## Usage

Any configuration of ACL rules with remote ACLs will receive this optimization if the port is using a single SG.

Functionality should remain as before in any case.

## Features to Install

Install the ODL Karaf feature for NetVirt (no change):

- odl-netvirt-openstack

## REST API

None.

## CLI

Refer to the Neutron CLI Reference<sup>1</sup> for the Neutron CLI command syntax for managing Security Rules with Remote Security Groups.

## Implementation

### Assignee(s)

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

Primary assignee:

- Alon Kochba <alonko@hpe.com>
- Aswin Suryanarayanan <asuryana@redhat.com>

Other contributors:

- ?

---

<sup>1</sup> Neutron Security Groups <http://docs.openstack.org/user-guide/cli-nova-configure-access-security-for-instances.html>

## Work Items

Task list in Carbon Trello

## Dependencies

None.

## Testing

## Unit Tests

## Integration Tests

## CSIT

We should add tests verifying remote SG configuration functionality. There should be at least:

- One security rule allowing ICMP traffic between VMs in the same SG.
- One positive test, checking ICMP connectivity works between two VMs using the same SG.
- One negative test, checking ICMP connectivity does not work between two VMs, one using the SG configured with the rule above, and the other using a separate security group with all directions allowed.

## Documentation Impact

None.

## References

### Table of Contents

- *ACL - Reflecting the ACL changes on existing traffic*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*

- \* *Targeted Release*
- \* *Alternatives*
- *Usage*
  - \* *Features to Install*
  - \* *REST API*
  - \* *CLI*
- *Implementation*
  - \* *Assignee(s)*
  - \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.4 ACL - Reflecting the ACL changes on existing traffic

ACL patches: <https://git.opendaylight.org/gerrit/#/q/topic:acl-reflection-on-existing-traffic>

This spec describes the new implementation for applying ACL changes on existing traffic.

In current ACL implementation, once a connection had been committed to the connection tracker, the connection would continue to be allowed, even if the policy defined in the ACL table has changed. This spec will explain the new approach that ensures ACL policy changes will affect existing connections as well. This approach will improve the pipeline behaviour in terms of reliable traffic between the VMs.

#### Problem description

When the traffic between two VMs starts, the first packet will match the actual SG flow, which commits the packets in connection tracker. It changes the state of the packets to established. Further traffic will match the global conntrack flow and go through the connection tracker straightly. This will continue until we terminate the established traffic.

When a rule is removed from the VM, the ACL flow getting removed from the respective tables. But, the already established traffic is still working, because the connection still exists as 'committed' in the conntrack tracker.

For example, consider the below scenario which explains the problem in detail,

- Create a VM and associate the rule which allows ICMP
- Ping the DHCP server from the VM
- Remove the ICMP rule and check the ongoing traffic

When we remove the ICMP rule, the respective ICMP flow getting removed from the respective table (For egress, table 213 and For Ingress, table 243). But, Since the conntrack flow having high priority than the SG flow, the packets are matched by the conntrack flow and the live traffic is unaware of the flow removal.

The traffic between the VMs should be reliable and it should be succeeded accordance with SG flow. When a SG rule is removed from the VM, the packets of ongoing traffic should be dropped.

## Use Cases

**The new ACL implementation will affect the below use cases,**

- VM Creation/Deletion with SG
- SG Rule addition and removal to/from existing SG associated to ports

## Proposed change

This spec proposes the fix that requires a new table (210/240) in the existing pipeline.

In this approach, we will use the “ct\_mark” flag of connection tracker. The default value of ct\_mark is zero.

- ct\_mark=0 matches the packet in new state
- ct\_mark=1 matches the packet in established state

For every new traffic, the ct\_mark value will be zero. When the traffic begins, the first packet of every new traffic will be matched by the respective SG flow which commits the packets into the connection tracker and changes the ct\_mark value to 1. So, every packets of established traffic will have the ct\_mark value as 1.

In conntrack flow, we will have a ct\_mark=1 match condition. After first packet committed to the connection tracker, further packets of established traffic will be matched by the conntrack flow straightly.

**In every SG flow, we will have below changes,** “table=213/243, priority=3902, ct\_state=+trk ,icmp,reg6=0x200/0xfffff00 actions=ct(commit,zone=6001, exec(set\_field:0x1->ct\_mark)),resubmit(,17/220)”

- The SG flow will match the packets which are in tracked state. It will commit the packet into the connection tracker. It will change the ct\_mark value to 1.
- When a VM having duplicate flows, the removal of one flow should not affect the existing traffic.

For example, consider a VM having ingress ICMP and Other protocol (ANY) rule. Ping the VM from the DHCP server. Removal of ingress ICMP rule from the VM should not affect the existing traffic. Because the Other protocol ANY flow will match the established packets of existing ICMP traffic and should make the communication possible. To make the communication possible in above specific scenarios, we should match the established packets in every SG flow. So, We will remove the “+new” check from the ct\_state condition of every ACL flow to recommit the established packets again into the conntrack.

**In conntrack flow,** “table=213/243, priority=62020,ct\_state=-new+est-rel-inv+trk, ct\_mark=0x1 actions=resubmit(,17/220)” “table=213/243, priority=62020,ct\_state=-new-est+rel-inv+trk, ct\_mark=0x1 actions=resubmit(,17/220)”

- The conntrack flow will match the packet which are in established state.
- For every new traffic, the first packet will be matched by the SG flow, which will change the ct\_mark value to 1. So, further packets will match the conntrack flow straightly.

**In default drop flow of table 213/243,** “table=213, n\_packets=0, n\_bytes=0, priority=50, ct\_state=+trk ,meta-data=0x2000000000/0xfffff00000000000 actions=drop” “table=243, n\_packets=6, n\_bytes=588, priority=50, ct\_state=+trk ,reg6=0x300/0xfffff00 actions=drop”

- For every VM, we are having a default drop flow to measure the drop statistics of particular VM. So, we will remove the “+new” state check from the ct\_state to measure the drop counts accurately.

Deletion of SG flow will add the below flow with configured hard time out in the table 212/242.

[1] “table=212/242, n\_packets=73, n\_bytes=7154, priority=40,icmp,reg6=0x200/0xfffff00,ct\_mark=1 actions=ct(commit, zone=5500, **exec(set\_field:0x0->ct\_mark)**),goto\_table:ACL4”

- It will match the ct\_mark value with the one and change the ct\_mark to zero.

The below tables describes the default hard time out of each protocol as configured in the contrack.

Protocol	Time out (secs)
ICMP	30
TCP	18000
UDP	180

Please refer the Pipeline Changes for table information.

For Egress, Dispatcher table (table 17) will forward the packets to the new table 210 where we will check the source match. It will forward the packet to 211 to match the destination of the packets. After the destination of the packet verified, The packets will forward to the table 212. New flow in the table, will match the ct\_mark value and forward the packets to the 213 table.

**Similarly, for Ingress, the packets will be forwarded through, Dispatcher table (220) >> New table (240) >> 241 >> 242 >> 243.**

In dispatcher flows, we will have the below changes which will change the table 211/241 from the goto\_table action to the new table 210/240.

“table=17, priority=10,metadata=0x20000000000/0xffffffff0000000000 actions=write\_metadata:0x900002157f000000/0xfffffffffffffffe, **goto\_table:210**”

“table=220, priority=6,reg6=0x200 actions=load:0x90000200->NXM\_NX\_REG6[],write\_metadata:0x157f000000/0xfffffffffffffe, **goto\_table:240**”

Deletion of SG rule will add a new flow in the table 212/242 as mentioned above. The first packet after SG got deleted, will match the above new flow and will change the ct\_mark value to zero. So this packet will not match the contrack flow and will check the ACL4 table whether it having any other flows to match this packet. If the SG flow found, the packet will be matched and change the ct\_mark value 1.

If we restore the SG rule again, we will delete the added flow [1] from the 212/242 table, so the packets of existing traffic will match the newly added SG flow in ACL4 table and proceed successfully.

Sample flows to be installed in each scenario,

#### SG rule addition

**SG flow: [ADD]** “table=213/243, n\_packets=33, n\_bytes=3234, priority=62021, **ct\_state=+trk**, icmp, reg6=0x200/0xfffff00 actions=ct(commit,zone=6001, **exec(set\_field:0x1->ct\_mark)**),resubmit(,17/220)”

**Contrack flow: [DEFAULT]** “table=213/243, n\_packets=105, n\_bytes=10290, priority=62020,ct\_state=-new+est-rel-inv+trk, **ct\_mark=0x1** actions=resubmit(,17/220)”

#### SG Rule deletion

**SG flow: [DELETE]** “table=213/243, n\_packets=33, n\_bytes=3234, priority=62021, ct\_state=+trk,icmp,reg6=0x200/0xfffff00 actions=ct(commit,zone=6001,exec(set\_field:0x1->ct\_mark)),resubmit(,17/220)”

**New flow: [ADD]** “table=212/242, n\_packets=73, n\_bytes=7154, priority=62021, **ct\_mark=0x1**,icmp,reg6=0x200/0xfffff00 actions=ct(commit, **exec(set\_field:0x0->ct\_mark)**),goto\_table:213/243”

#### Rule Restore

**SG flow: [ADD]** “table=213/243, n\_packets=33, n\_bytes=3234, priority=62021, ct\_state=+trk, icmp,reg6=0x200/0xfffff00 actions=ct(commit,zone=6001,exec(set\_field:0x1->ct\_mark)),resubmit(,17/220)”

**New flow: [DELETE]** “table=212/242, n\_packets=73, n\_bytes=7154, priority=62021,ct\_mark=0x1,icmp,reg6=0x200/0xfffff00 actions=ct(commit,exec(set\_field:0x0->ct\_mark)),goto\_table:213/243”

The new tables (210/240) will matches the source and the destination of the packets respectively. So, a default flow will be added in the table 210/240 with least priority to drop the packets.

“table=210/240, n\_packets=1, n\_bytes=98, priority=0 actions=drop”

#### Flow Sample:

Egress flows before the changes,

```
cookie=0x6900000, duration=30.590s, table=17, n_packets=108,
n_bytes=10624, priority=10,metadata=0x20000000000/0xfffff00000000000
actions=write_metadata:0x9000021389000000/0xfffffffffffffe,goto_table:211
cookie=0x6900000, duration=30.247s, table=211, n_packets=0, n_bytes=0, pri-
ority=61010,ipv6,d1_src=fa:16:3e:93:dc:92,ipv6_src=fe80::f816:3eff:fe93:dc92
actions=ct(table=212,zone=5001) cookie=0x6900000, dura-
tion=30.236s, table=211, n_packets=96, n_bytes=9312, pri-
ority=61010,ip,d1_src=fa:16:3e:93:dc:92,nw_src=10.100.5.3 ac-
tions=ct(table=212,zone=5001) cookie=0x6900000, duration=486.527s,
table=211, n_packets=2, n_bytes=180, priority=0 actions=drop
cookie=0x6900000, duration=30.157s, table=212, n_packets=0, n_bytes=0, prior-
ity=50,ipv6,metadata=0x1389000000/0xffff000000,ipv6_dst=fe80::f816:3eff:fe93:dc92
actions=write_metadata:0x2/0xfffffe,goto_table:212 cookie=0x6900000,
duration=30.152s, table=212, n_packets=0, n_bytes=0, prior-
ity=50,ip,metadata=0x1389000000/0xffff000000,nw_dst=10.100.5.3 ac-
tions=write_metadata:0x2/0xfffffe,goto_table:212 cookie=0x6900000,
duration=486.527s, table=212, n_packets=96, n_bytes=9312, prior-
ity=0 actions=goto_table:212 cookie=0x6900000, duration=486.056s,
table=213, n_packets=80, n_bytes=8128, priority=62020,ct_state=-
new+est-rel-inv+trk actions=resubmit(,17) cookie=0x6900000, dura-
tion=485.948s, table=213, n_packets=0, n_bytes=0, priority=62020,ct_state=-
new-est+rel-inv+trk actions=resubmit(,17) cookie=0x6900001, du-
ration=30.184s, table=213, n_packets=0, n_bytes=0, prior-
ity=62015,ct_state=+inv+trk,metadata=0x20000000000/0xfffff00000000000
actions=drop cookie=0x6900000, duration=30.177s, ta-
ble=213, n_packets=16, n_bytes=1184, prior-
ity=1000,ct_state=+new+trk,ip,metadata=0x20000000000/0xfffff00000000000
actions=ct(commit,zone=5001),resubmit(,17) cookie=0x6900000,
duration=30.168s, table=213, n_packets=0, n_bytes=0, prior-
ity=1001,ct_state=+new+trk,ipv6,metadata=0x20000000000/0xfffff00000000000
actions=ct(commit,zone=5001),resubmit(,17) cookie=0x6900001,
duration=30.207s, table=213, n_packets=0, n_bytes=0, prior-
ity=50,ct_state=+new+trk,metadata=0x20000000000/0xfffff00000000000 ac-
tions=dro
```

After the changes, flows will be,

```
cookie=0x6900000, duration=30.590s, table=17, n_packets=108,
n_bytes=10624, priority=10,metadata=0x20000000000/0xfffff00000000000 ac-
tions=write_metadata:0x9000021389000000/0xfffffffffffffe,goto_table:210
cookie=0x6900000, duration=30.247s, table=210, n_packets=0, n_bytes=0, pri-
```

```

ority=61010,ipv6,dl_src=fa:16:3e:93:dc:92,ipv6_src=fe80::f816:3eff:fe93:dc92
actions=ct(table=211,zone=5001)          cookie=0x6900000,          dura-
tion=30.236s,          table=210,          n_packets=96,          n_bytes=9312,          pri-
ority=61010,ip,dl_src=fa:16:3e:93:dc:92,nw_src=10.100.5.3          ac-
tions=ct(table=211,zone=5001)          cookie=0x6900000,          duration=486.527s,
table=210,          n_packets=2,          n_bytes=180,          priority=0          actions=drop
cookie=0x6900000, duration=30.157s, table=211, n_packets=0, n_bytes=0, prior-
ity=50,ipv6,metadata=0x1389000000/0xffff000000,ipv6_dst=fe80::f816:3eff:fe93:dc92
actions=write_metadata:0x2/0xfffffe,goto_table:212          cookie=0x6900000,
duration=30.152s,          table=211,          n_packets=0,          n_bytes=0,          prior-
ity=50,ip,metadata=0x1389000000/0xffff000000,nw_dst=10.100.5.3          ac-
tions=write_metadata:0x2/0xfffffe,goto_table:212          cookie=0x6900000,          du-
ration=486.527s,          table=211,          n_packets=96,          n_bytes=9312,          prior-
ity=0          actions=goto_table:212          cookie=0x6900000,          duration=486.527s,          ta-
ble=212,          n_packets=96,          n_bytes=9312,          priority=0          actions=goto_table:213
cookie=0x6900000, duration=486.056s, table=213, n_packets=80, n_bytes=8128,
priority=62020,ct_state=-new+est-rel-inv+trk,ct_mark=0x1          actions=resubmit,(17)
cookie=0x6900000, duration=485.948s, table=213, n_packets=0, n_bytes=0,
priority=62020,ct_state=-new-est+rel-inv+trk,ct_mark=0x1          actions=resubmit,(17)
cookie=0x6900001, duration=30.184s, table=213, n_packets=0, n_bytes=0, prior-
ity=62015,ct_state=+inv+trk,metadata=0x20000000000/0xffff000000000000 actions=drop
cookie=0x6900000, duration=30.177s, table=213, n_packets=16, n_bytes=1184,
priority=1000,ct_state=+trk,ip,metadata=0x20000000000/0xffff000000000000
actions=ct(commit,zone=5001,exec(set_field:0x1->ct_mark)),resubmit,(17)
cookie=0x6900000, duration=30.168s, table=213, n_packets=0, n_bytes=0, pri-
ority=1001,ct_state=+new+trk,ipv6,metadata=0x20000000000/0xffff000000000000
actions=ct(commit,zone=5001),resubmit,(17)          cookie=0x6900001,          du-
ration=30.207s,          table=213,          n_packets=0,          n_bytes=0,          prior-
ity=50,ct_state=+trk,metadata=0x20000000000/0xffff000000000000 actions=drop

```

**New flow will be installed in table 212 when we delete SG rule, “cookie=0x6900000, duration=30.177s, table=212, n\_packets=16, n\_bytes=1184, priority=1000,ct\_state=+trk,ip,metadata=0x20000000000/0xffff000000000000,ct\_mark=1,idle\_timeout=1800 actions=ct(commit,zone=5001,exec(set\_field:0x0->ct\_mark)),goto\_table:213”**

Similarly, the ingress related flows will have the same changes as mentioned above.

## Pipeline changes

### The propose changes includes:

- New tables 210 and 240
- Re-purposed tables 211, 212, 241, 242

The propose will re-purpose the table 211 and 212 of egress, table 241 and 242 of ingress.

Currently, for egress, we are using the table 211 for source match and 212 for destination match. In new propose, we will use the new table 210 for source match, table 211 for destination match and table 212 for new flow installation when we delete the SG flow.

**For Egress, the traffic will use the tables in following order, 17 >> 210 >> 211 >> 212 >> 213.**

Similarly, for ingress, currently we are using the table 241 for destination match and 242 for source match. In new propose, we will use the new table 240 for destination match, table 241 for source match and table 242 for new flow installation when we delete the SG flow.

**For Ingress, the traffic will use the tables in following order, 220 >> 240 >> 241 >> 242 >> 243**

flow will be added in table 212/242, and the match condition of ACL4 flows will be modified as noted above in the proposed change:

Table	Match	Action
Dispatcher	metadata=service_id:ACL	write_metadata:(elan_id=ELAN, service_id=NEXT), goto_table:210/240 (ACL1)
ACL1 (210/240)		goto_table:ACL2
...		
ACL2 (211/241)		goto_table:ACL3
ACL3 (212/242)	ip,ct_mark=0x1,reg6=0x200/0xfffff00	(set_field:0x0->ct_mark), goto_table:ACL4
ACL3 (212/242)		goto_table:ACL4
ACL4 (213/243)	ct_state=-new+est-rel- inv+trk,ct_mark=0x1	resubmit,(DISPATCHER)
ACL4 (213/243)	ct_state=+trk,priority=3902,ip,reg6=0x200/0xfffff00	(set_field:0x0->ct_mark, resubmit,(DISPATCHER)
ACL4 (213/243)	ct_state=+trk, reg6=0x200/0xfffff00	drop
...		

## Yang changes

The nicira-action.yang and the openflowplugin-extension-nicira-action.yang needs to be updated with ct\_mark action. The action structure shall be

```
grouping ofj-nx-action-contrack-grouping {
  container nx-action-contrack {
    leaf flags {
      type uint16;
    }
    leaf zone-src {
      type uint32;
    }
    leaf contrack-zone {
      type uint16;
    }
    leaf recirc-table {
      type uint128;
    }
    leaf experimenter-id {
      type of:experimenter-id;
    }
    list ct-actions{
      uses ofpact-actions;
    }
  }
}
```

The nicira-match.yang and the openflowplugin-extension-nicira-match.yang needs to be updated with the ct\_mark match.



```

grouping ofj-nxm-nx-match-ct-mark-grouping{
    container ct-mark-values {
        leaf ct-mark {
            type uint32;
        }
        leaf mask {
            type uint32;
        }
    }
}

```

### Configuration impact

None.

### Clustering considerations

None.

### Other Infra considerations

None.

### Security considerations

None.

### Scale and Performance Impact

When we delete the SG rule from the VM, A new flow will be added in the flow table 212 to flip the value of ct\_mark of ongoing traffics. This flow will have a time out based on the protocol as mentioned in the proposed changes section. The packets of ongoing traffic will be recommitted and will do the set filed of ct\_mark until the flow reaches the time out.

### Targeted Release

Carbon

### Alternatives

While deleting a SG flow from the flow table, we will add a DROP flow with the highest priority in the ACL4 table. This DROP flow will drop the packets and it will stop the existing traffic. Similarly, when we restore the same rule again, we will delete the DROP flow from the ACL4 table which will enable the existing traffic.

But this approach will be effective only if the VM do not have any duplicate flows. With the current ACL implementation, if we associate two SGs which having similar set of SG rule, netvirt will install the two set of flows with different priority for the same VM.

As per above approach, if we dissociate any one of SG from the VM, It will add the DROP flow in ACL4 table which will stop the existing traffic irrespective of there is still another flow available in ACL4, to make the traffic possible.

### Usage

Traffic between VMs will work accordance with the SG flow existence in the flow table.

### Features to Install

Install the ODL Karaf feature for NetVirt (no change):

- odl-netvirt-openstack

### REST API

None.

### CLI

Refer to the Neutron CLI Reference<sup>1</sup> for the Neutron CLI command syntax for managing Security Rules.

### Implementation

#### Assignee(s)

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

Primary assignee:

- VinothB <vinothb@hcl.com>
- Balakrishnan Karuppasamy <balakrishnan.ka@hcl.com>

Other contributors:

- ?

#### Work Items

None

#### Dependencies

None.

---

<sup>1</sup> Neutron Security Groups <http://docs.openstack.org/user-guide/cli-nova-configure-access-security-for-instances.html>

## Testing

### Unit Tests

### Integration Tests

### CSIT

We should add tests verifying ACL change reflection on existing traffic. There should be at least:

- One security rule allowing ICMP traffic between VMs in the same SG.
- One positive test, checking ICMP connectivity works between two VMs using the same SG. Delete all the rules from the SG without disturbing the already established traffic. It should stop the traffic.
- One positive test, checking ICMP connectivity works between two VMs, one using the SG, configured with the ICMP rule. Delete and restore the ICMP rule immediately. This should stop and resume the ICMP traffic after restoring the ICMP rule.
- One positive test, checking ICMP connectivity between VMs, using the SG, configured with ICMP ALL and Other protocol ANY rule. Delete the ICMP rule from the SG, It should not stop the ICMP traffic.
- One negative test, checking ICMP connectivity between two VMs, one using the SG, configured with the ICMP and TCP rules above, and delete the TCP rule. This should not affect the ICMP traffic.

### Documentation Impact

None.

### References

#### Table of Contents

- *Conntrack Based SNAT*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*

- *Usage*
  - \* *Create External Network*
  - \* *Create Internal Network*
  - \* *Create Router*
  - \* *Features to Install*
  - \* *REST API*
  - \* *CLI*
- *Implementation*
  - \* *Assignee(s)*
  - \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.5 Conntrack Based SNAT

[https://git.opendaylight.org/gerrit/#/q/topic:snat\\_conntrack](https://git.opendaylight.org/gerrit/#/q/topic:snat_conntrack)

The ovs conntrack based SNAT implements Source Network Address Translation using openflow rules by leveraging ovs-netfilter integration.

#### Problem description

Today SNAT is done in OpenDaylight netvirt using controller punting and thus controller installing the rules for inbound and outbound NAPT. This causes significant delay as the first packet of all the new connections needs to go through the controller. The number of flows grows linearly with the increase in the vms. Also the current implementation does not support ICMP.

The current algorithm for selecting the NAPT switch does not work well with conntrack based SNAT. For a NAPT switch to remain as designated NAPT switch, it requires at least one port from any of the subnets present in the router. When such a port ceases to exist a new NAPT switch will be elected. With the controller based implementation the failover is faster as the NAT flows are reinstalled to the new NAPT switch and should not lead to termination of existing connection. With the conntrack based approach, the translation will be lost and the newly elected switch will have to redo the translation. This will lead to connection timeout for TCP like connections. So the re-election needs to be prevented unless switch is down. Also the current implementation tends to select the node running the DHCP agent as the designated NAPT switch as the DHCP port is the first port created for a subnet.

## Use Cases

The following use case will be realized by the implementation

**External Network Access** The SNAT enables the VM in a tenant network access the external network without using a floating ip. It uses NAPT for sharing the external ip address across multiple VMs that share the same router gateway.

## Proposed change

The proposed implementation uses linux netfilter framework to do the NAPT (Network Address Port Translation) and for tracking the connection. The first packet of a traffic will be committed to the netfilter for translation along with the external ip. The subsequent packets will use the entry in the netfilter for inbound and outbound translation. The router id will be used as the zone id in the netfilter. Each zone tracks the connection in its own table. The rest of the implementation for selecting the designated NAPT switch and non designated switches will remain the same. The pipeline changes will happen in the designated switch. With this implementation we will be able to do translation for icmp as well.

The openflow plugin needs to support new set of actions for conntrack based NAPT. This shall be added in the nicira plugin extension of OpenFlow plugin.

The new implementation will not re-install the existing NAT entries to the new NAPT switch during fail-over. Also spec does not cover the use case of having multiple external subnets in the same router.

The HA framework will have a new algorithm to elect the designated NAPT switch. The new logic will be applicable only if the conntrack mode is selected. The switch selection logic will also be modified to use round robin logic with weights associated with each switch. It will not take into account whether a port belonging to a subnet in the router is present in the switch. The initial weight of all the switches shall be 0 and will be incremented by 1 when the switch is selected as the designated NAPT. The weights shall be decremented by 1 when the router is deleted. At any point of time the switch with the lowest weight will be selected as the designated NAPT switch for a new router. If there are multiple the first one with the lowest weight will be selected. A pseudo port will be added in the switch which is selected as the designated NAPT switch. This port will be deleted only when the switch cease to be a designated NAPT switch. This helps the switch to maintain the remote flows even when there are no ports in the router subnet in the switch. Only if the switch hosting the designated NAPT switch is down a new NAPT switch will be elected.

## Pipeline changes

The ovs based NAPT flows will replace the controller based NAPT flows. The changes are limited to the designated switch for the router. Below is the illustration for flat external network.

### Outbound NAPT

Table 26 (PSNAT Table) => submits the packet to netfilter to check whether it is an existing connection. Resubmits the packet back to 46.

Table 46 (NAPT OUTBOUND TABLE) => if it is an established connection, it indicates the translation is done and the packet is forwarded to table 47 after writing the external network metadata.

If it is a new connection the connection will be committed to netfilter and this entry will be used for NAPT. The translated packet will be resubmitted to table 47. The external network metadata will be written before sending the packet to netfilter.

Table 47 (NAPT FIB TABLE) => The translated packet will be sent to the egress group.

Sample Flows

```

table=26, priority=5, ip, metadata=0x222e2/0xfffffffffe actions=ct (table=46, zone=5003, nat)
table=46, priority=6, ct_state+=snat, ip, metadata=0x222e2/0xfffffffffe actions=set_
↪field:0x222e0->metadata, resubmit(, 47)
table=46, priority=5, ct_state+=new+trk, ip, metadata=0x222e2/0xfffffffffe actions=set_
↪field:0x222e0->metadata, ct (commit, table=47, zone=5003, nat (src=192.168.111.21))
table=47, n_packets=0, n_bytes=0, priority=6, ct_state+=snat, ip, nw_src=192.168.111.21_
↪actions=group:200000

```

### Inbound NATP

Table 44 (NAPT INBOUND Table)=> submits the packet to netfilter to check for an existing connection after changing the metadata to that of the internal network. The packet will be submitted back to table 47.

Table 47 (NAPT FIB TABLE) => The translated packet will be submitted back to table 21.

### Sample Flows

```

table=21, priority=42, ip, metadata=0x222e0/0xfffffffffe, nw_dst=192.168.111.21_
↪actions=resubmit(, 44)
table=44, priority=10, ip, metadata=0x222e0/0xfffffffffe, nw_dst=192.168.111.21_
↪actions=set_field:0x222e2->metadata, ct (table=47, zone=5003, nat)
table=47, priority=5, ct_state+=dnat, ip actions=resubmit(, 21)

```

## Yang changes

The nicira-action.yang and the openflowplugin-extension-nicira-action.yang needs to be updated with nat action. The action structure shall be

```

typedef nx-action-nat-range-present {
    type enumeration {
        enum NX_NAT_RANGE_IPV4_MIN {
            value 1;
            description "IPv4 minimum value is present";
        }
        enum NX_NAT_RANGE_IPV4_MAX {
            value 2;
            description "IPv4 maximum value is present";
        }
        enum NX_NAT_RANGE_IPV6_MIN {
            value 4;
            description "IPv6 minimum value is present in range";
        }
        enum NX_NAT_RANGE_IPV6_MAX {
            value 8;
            description "IPv6 maximum value is present in range";
        }
        enum NX_NAT_RANGE_PROTO_MIN {
            value 16;
            description "Port minimum value is present in range";
        }
        enum NX_NAT_RANGE_PROTO_MAX {
            value 32;
            description "Port maximum value is present in range";
        }
    }
}

```

```

typedef nx-action-nat-flags {
    type enumeration {
        enum NX_NAT_F_SRC {
            value 1;
            description "Source nat is selected ,Mutually exclusive with NX_NAT_F_DST
↪";
        }
        enum NX_NAT_F_DST {
            value 2;
            description "Destination nat is selected";
        }
        enum NX_NAT_F_PERSISTENT {
            value 4;
            description "Persistent flag is selected";
        }
        enum NX_NAT_F_PROTO_HASH {
            value 8;
            description "Hash mode is selected for port mapping, Mutually exclusive_
↪with
            NX_NAT_F_PROTO_RANDOM ";
        }
        enum NX_NAT_F_PROTO_RANDOM {
            value 16;
            description "Port mapping will be randomized";
        }
    }
}

grouping ofj-nx-action-conntrack-grouping {
    container nx-action-conntrack {
        leaf flags {
            type uint16;
        }
        leaf zone-src {
            type uint32;
        }
        leaf conntrack-zone {
            type uint16;
        }
        leaf recirc-table {
            type uint8;
        }
        leaf experimenter-id {
            type ofc:experimenter-id;
        }
        list ct-actions{
            uses ofpact-actions;
        }
    }
}

grouping ofpact-actions {
    description
        "Actions to be performed with conntrack.";
    choice ofpact-actions {
        case nx-action-nat-case {
            container nx-action-nat {
                leaf flags {

```

```
        type uint16;
    }
    leaf range_present {
        type uint16;
    }
    leaf ip-address-min {
        type inet:ip-address;
    }
    leaf ip-address-max {
        type inet:ip-address;
    }
    leaf port-min {
        type uint16;
    }
    leaf port-max {
        type uint16;
    }
}
}
```

For the new configuration knob a new yang nat-service-config shall be added in NAT service, with the container for holding the NAT mode configured. It will have two options controller and conntrack, with controller being the default.

```
container nat-service-config {
    config true;
    leaf nat-mode {
        type enumeration {
            enum "controller";
            enum "conntrack";
        }
        default "controller";
    }
}
```

## Configuration impact

The proposed change requires the NAT service to provide a configuration knob to switch between the controller based/conntrack based implementation. A new configuration file netvirt-nat-service-config.xml shall be added with default value controller.

```
<nat-service-config xmlns="urn:opendaylight:netvirt:nat-service-config">
  <nat-mode>controller</nat-mode>
</nat-service-config>
```

The dynamic update of nat-mode will not be supported. To change the nat-mode the controller cluster needs to be restarted after changing the nat-mode. On restart the NAT translation lifecycle will be reset and after the controller comes up in the updated nat-mode, a new set of switches will be elected as designated NAPT switches and it can be different from the ones that were forwarding traffic earlier.

## Clustering considerations

NA



## Other Infra considerations

The implementation requires ovs2.6 with the kernel module installed. OVS currently does not support SNAT connection tracking for dpdk datapath. It would be supported in some future release.

## Security considerations

NA

## Scale and Performance Impact

The new SNAT implementation is expected to improve the performance when compared to the existing one and will reduce the flows in ovs pipeline.

## Targeted Release

Carbon

## Alternatives

An alternative implementation of X NAT switches was discussed, which will not be a part of this document but will be considered as a further enhancement.

## Usage

### Create External Network

Create an external flat network and subnet

```
neutron net-create ext1 --router:external --provider:physical_network public --
↪provider:network_type flat
neutron subnet-create --allocation-pool start=<start-ip>,end=<end-ip> --gateway=<gw-
↪ip> --disable-dhcp --name subext1 ext1 <subnet-cidr>
```

### Create Internal Network

Create an internal n/w and subnet

```
neutron net-create vx-net1 --provider:network_type vxlan
neutron subnet-create vx-net1 <subnet-cidr> --name vx-subnet1
```

### Create Router

Create a router and add an interface to internal n/w. Set the external n/w as the router gateway.

```
neutron router-create router1
neutron router-interface-add router1 vx-subnet1
neutron router-gateway-set router1 ext1
nova boot --poll --flavor m1.tiny --image $(nova image-list | grep 'uec\s' | awk '
↪{print $2}' | tail -1) --nic net-id=$(neutron net-list | grep -w vx-net1 | awk '
↪{print $2}') vmvx2
```

## Features to Install

odl-netvirt-openstack

## REST API

NA

## CLI

A new command line, `display-napt-switch`, will be added to display the current designated NAPT switch selected for each router. It shall show the below info.

```
router id | Host Name of designated NAPT switch | Management Ip of the designated_
↪NAPT switch
```

## Implementation

### Assignee(s)

Aswin Suryanarayanan <asuryana@redhat.com>

### Work Items

<https://trello.com/c/DMLsrLfQ/9-snat-decentralized-ovs-nat-based>

- Write a framework which can support multiple modes of NAT implementation.
- Add support in openflow plugin for conntrack nat actions.
- Add support in genius for conntrack nat actions.
- Add a config parameter to select between controller based and conntrack based.
- Add the flow programming for SNAT in netvirt.
- Add the new HA framework.
- Add the command to display the designated NAPT switch.
- Write Unit tests for conntrack based snat.

## Dependencies

NA

## Testing

### Unit Tests

Unit test needs to be added for the new snat mode. It shall use the component tests framework

### Integration Tests

Integration tests needs to be added for the conntrack snat flows.

### CSIT

Run the CSIT with conntrack based SNAT configured.

### Documentation Impact

Necessary documentation would be added on how to use this feature.

## References

### Table of Contents

- *Cross site connectivity with federation service*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*
  - *Usage*
    - \* *Features to Install*
    - \* *REST API*
    - \* *CLI*
  - *Implementation*

- \* *Assignee(s)*
- \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.6 Cross site connectivity with federation service

<https://git.opendaylight.org/gerrit/#/q/topic:federation-plugin>

Enabling neutron networks to expand beyond a single OpenStack instance to allow L2 switching and L3 routing between sites. Sites may be geographically remote or partitioned in a single data center.

Each site is deployed with independent local ODL cluster. The clusters communicate using the federation infrastructure [2] in order to publish MDSAL events whenever routable entities e.g. VM instances are added/removed from remote sites.

VxLAN tunnels are used to form the overlay for cross site communication between OpenStack compute nodes.

#### Problem description

Today, communication between VMs in remote sites is based on BGP control plane and requires DC-GW. Overlay network between data centers is based on MPLSoverGRE or VxLAN if the DC-GW supports EVPN RT5 [4]. The purpose of this feature is to allow inter-DC communication independent from BGP control plane and DC-GW.

#### Use Cases

This feature will cover the following use cases:

##### L2 switching use cases

- L2 Unicast frames exchanged between VMs sharing federated neutron network between OVS datapaths in remote sites
- L2 Unicast frames exchanged between VM and PNF sharing federated neutron network between OVS and HWVTEP datapath in remote sites
- L2 Broadcast frames exchanged between VMs sharing federated neutron network between OVS datapaths in remote sites
- L2 Broadcast frames exchanged between VM and PNF sharing federated neutron network between OVS and HWVTEP datapath in remote sites

## L3 forwarding use cases

- L3 traffic exchanged between VMs sharing federated neutron router between OVS datapaths in remote sites

## Proposed change

For Carbon release, cross-site connectivity will be based on the current HPE downstream federation plugin code-base. This plugin implements the federation service API [3] to synchronize the following MDSAL subtrees between connected sites:

- config/ietf-interfaces:interfaces
- config/elan:elan-interfaces
- config/l3vpn:vpn-interfaces
- config/network-topology:network-topology/topology/ovsdb:1
- operational/network-topology:network-topology/topology/ovsdb:1
- config/network-topology:network-topology/topology/hwvtep:1
- operational/network-topology:network-topology/topology/hwvtep:1
- config/opendaylight-inventory:nodes
- operational/opendaylight-inventory:nodes
- config/neutron:neutron/l2gateways
- config/neutron:neutron/l2gatewayConnections

The provisioning of connected networks between remote sites is out of the scope of this spec and described in [6].

Upon receiving a list of shared neutron networks and subnets, the federation plugin will propagate MDSAL entities from all of the subtrees detailed above to remote sites based on the federation connection definitions. The federated entities will be transformed to match the target network/subnet/router details in each remote site.

For example, ELAN interface will be federated with elan-instance-name set to the remote site elan-instance-name. VPN interface will be federated with the remote site vpn-instance-name i.e. router-id and remote subnet-id contained in the primary VPN interface adjacency.

This would allow remotely federated entities a.k.a shadow entities to be handled the same way local entities are handled thus shadow entities will appear as if they were local entities in remote sites. As a result, the following pipeline elements will be added for shadow entities on all compute nodes in each connected remote site:

- ELAN remote DMAC flow for L2 unicast packets to remote site
- ELAN remote broadcast group buckets for L2 multicast packets to remote site
- FIB remote nexthop flow for L3 packet to remote site

The following limitations exist for the current federation plugin implementation:

- Federated networks use VxLAN network type and the same VNI is used across sites.
- The IP addresses allocated to VM instances in federated subnets do not overlap across sites.
- The neutron-configured VNI will be passed on the wire for inter-DC L2/L3 communication between VxLAN networks. The implementation is described in [5].

As part of Nitrogen, the federation plugin is planned to go through major redesign. The scope and internals have not been finalized yet but this spec might be a good opportunity to agree on an alternate solution.

Some initial thoughts:

- For L3 cross site connectivity, it seems that federating the FIB vrf-entry associated with VMs in connected networks should be sufficient to form remote nexthop connectivity across sites.
- In order to create VxLAN tunnels to remote sites, it may be possible to use the external tunnel concept instead of creating internal tunnels that are dependent on federation of the OVS topology nodes from remote sites.
- L2 cross site connectivity is the most challenging part for federation of MAC addresses of both VM instances and PNFs connected to HWVTEP. If the ELAN model could be enhanced to have remote-mac-entry model containing MAC address, ELAN instance name and remote TEP ip, it would be possible to federate such entity to remote sites in order to create remote DMAC flows for cases of remote VM instances and PNFs connected HWVTEP in remote sites.

## Pipeline changes

No new pipeline changes are introduced as part of this feature. The pipeline flow between VM instances in remote sites is similar to the current implementation of cross compute intra-DC traffic since the realization of remote compute nodes is similar to local ones.

## Yang changes

The following new yang models will be introduced as part of the federation plugin API bundle:

### Federation Plugin Yang

Marking for each federated entity using shadow-properties augmentation

```
module federation-plugin {
  yang-version 1;
  namespace "urn:opendaylight:netvirt:federation:plugin";
  prefix "federation-plugin";

  import yang-ext {
    prefix ext;
    revision-date "2013-07-09";
  }

  import ietf-yang-types {
    prefix yang;
  }

  import network-topology {
    prefix topo;
  }

  import opendaylight-inventory {
    prefix inv;
  }

  import ietf-interfaces {
    prefix if;
  }

  import elan {
    prefix elan;
  }
}
```

```

}

import l3vpn {
    prefix l3vpn;
}

import neutronvpn {
    prefix nvpn;
}

revision "2017-02-19" {
    description "Federation plugin model";
}

grouping shadow-properties {
    leaf shadow {
        type boolean;
        description "Represents whether this is a federated entity";
    }
    leaf generation-number {
        type int32;
        description "The current generation number of the federated entity";
    }
    leaf remote-ip {
        type string;
        description "The IP address of the original site of the federated entity";
    }
}

augment "/topo:network-topology/topo:topology/topo:node" {
    ext:augment-identifier "topology-node-shadow-properties";
    uses shadow-properties;
}

augment "/inv:nodes/inv:node" {
    ext:augment-identifier "inventory-node-shadow-properties";
    uses shadow-properties;
}

augment "/if:interfaces/if:interface" {
    ext:augment-identifier "if-shadow-properties";
    uses shadow-properties;
}

augment "/elan:elan-interfaces/elan:elan-interface" {
    ext:augment-identifier "elan-shadow-properties";
    uses shadow-properties;
}

augment "/l3vpn:vpn-interfaces/l3vpn:vpn-interface" {
    ext:augment-identifier "vpn-shadow-properties";
    uses shadow-properties;
}
}

```

## Federation Plugin Manager Yang

Management of federated networks and routed RPCs subscription

```
module federation-plugin-manager {
  yang-version 1;
  namespace "urn:opendaylight:netvirt:federation:plugin:manager";
  prefix "federation-plugin-manager";

  import yang-ext {
    prefix ext;
    revision-date "2013-07-09";
  }

  import ietf-yang-types {
    prefix yang;
  }

  revision "2017-02-19" {
    description "Federation plugin model";
  }

  identity mgr-context {
    description "Identity for a routed RPC";
  }

  container routed-container {
    list route-key-item {
      key "id";
      leaf id {
        type string;
      }

      ext:context-instance "mgr-context";
    }
  }

  container federated-networks {
    list federated-network {
      key self-net-id;
      uses federated-nets;
    }
  }

  container federation-generations {
    description
      "Federation generation information for a remote site.";
    list remote-site-generation-info {
      max-elements "unbounded";
      min-elements "0";
      key "remote-ip";
      leaf remote-ip {
        mandatory true;
        type string;
        description "Remote site IP address.";
      }
      leaf generation-number {
        type int32;
      }
    }
  }
}
```



```

        description "The current generation number used for the remote site.";
    }
}

grouping federated-nets {
    leaf self-net-id {
        type string;
        description "UUID representing the self net";
    }
    leaf self-subnet-id {
        type yang:uuid;
        description "UUID representing the self subnet";
    }
    leaf self-tenant-id {
        type yang:uuid;
        description "UUID representing the self tenant";
    }
    leaf subnet-ip {
        type string;
        description "Specifies the subnet IP in CIDR format";
    }
}

list site-network {
    key id;
    leaf id {
        type string;
        description "UUID representing the site ID (from xsite manager)";
    }
    leaf site-ip {
        type string;
        description "Specifies the site IP";
    }
    leaf site-net-id {
        type string;
        description "UUID of the network in the site";
    }
    leaf site-subnet-id {
        type yang:uuid;
        description "UUID of the subnet in the site";
    }
    leaf site-tenant-id {
        type yang:uuid;
        description "UUID of the tenant holding this network in the site";
    }
}
}
}

```

## Federation Plugin RPC Yang

FederationPluginRpcService yang definition for update-federated-networks RPC

```

module federation-plugin-rpc {
    yang-version 1;
    namespace "urn:opendaylight:netvirt:federation:plugin:rpc";

```

```

prefix "federation-plugin-rpc";

import yang-ext {
    prefix ext;
    revision-date "2013-07-09";
}

import ietf-yang-types {
    prefix yang;
}

import federation-plugin-manager {
    prefix federation-plugin-manager;
}

revision "2017-02-19" {
    description "Federation plugin model";
}

rpc update-federated-networks {
    input {
        list federated-networks-in {
            key self-net-id;
            uses federation-plugin-manager:federated-nets;
            description "Contain all federated networks in this site that are still
↪be considered                                connected, a federated network that does not appear will
                                                disconnected";
        }
    }
}

```

## Federation Plugin routed RPC Yang

Routed RPCs will be used only within the cluster to route connect/disconnect requests to the federation cluster singleton.

```

module federation-plugin-routed-rpc {
    yang-version 1;
    namespace "urn:opendaylight:netvirt:federation:plugin:routed:rpc";
    prefix "federation-plugin-routed-rpc";

    import yang-ext {
        prefix ext;
        revision-date "2013-07-09";
    }

    import ietf-yang-types {
        prefix yang;
    }

    import federation-plugin-manager {
        prefix federation-plugin-manager;
    }
}

```

```

revision "2017-02-19" {
    description "Federation plugin model";
}

rpc update-federated-networks {
    input {
        leaf route-key-item {
            type instance-identifier;
            ext:context-reference federation-plugin-manager:mgr-context;
        }

        list federated-networks-in {
            key self-net-id;
            uses federation-plugin-manager:federated-nets;
        }
    }
}

```

### Configuration impact

None.

### Clustering considerations

The federation plugin will be active only on one of the ODL instances in the cluster. The cluster singleton service infrastructure will be used in order to register the federation plugin routed RPCs only on the selected ODL instance.

### Other Infra considerations

None

### Security considerations

None

### Scale and Performance Impact

None

### Targeted Release

Carbon

### Alternatives

None

## Usage

### Features to Install

odl-netvirt-federation

This is a new feature that will load odl-netvirt-openstack and the federation service features. It will not be installed by default and requires manual startup using `karaf feature:install` command.

### REST API

Connecting neutron networks from remote sites

**URL:** restconf/operations/federation-plugin-manager:update-federated-networks

#### Sample JSON data

```
{
  "input": {
    "federated-networks-in": [
      {
        "self-net-id": "c4976ee7-c5cd-4a5e-9cf9-261f28ba7920",
        "self-subnet-id": "93dee7cb-ba25-4318-b60c-19a15f2c079a",
        "subnet-ip": "10.0.123.0/24",
        "site-network": [
          {
            "id": "c4976ee7-c5cd-4a5e-9cf9-261f28ba7922",
            "site-ip": "10.0.43.146",
            "site-net-id": "c4976ee7-c5cd-4a5e-9cf9-261f28ba7921",
            "site-subnet-id": "93dee7cb-ba25-4318-b60c-19a15f2c079b",
          }
        ]
      }
    ]
  }
}
```

### CLI

None.

### Implementation

#### Assignee(s)

**Primary assignee:** Tali Ben-Meir <tali@hpe.com>

**Other contributors:** Guy Sela <guy.sela@hpe.com>

Shlomi Alfasi <shlomi.alfasi@hpe.com>

Yair Zinger <yair.zinger@hpe.com>

## Work Items

Trello card <https://trello.com/c/mgdUO6xx/154-federation-plugin-for-netvirt>

Since the code was already implemented in downstream no work items will be defined

## Dependencies

This feature will be implemented in 2 new bundles - `federation-plugin-api` and `federation-plugin-impl` the implementation will be dependent on `federation-service-api` [3] bundle from OpenDaylight federation project.

The new karaf feature `odl-netvirt-federation` will encapsulate the `federation-plugin api` and `impl` bundles and will be dependant on the followings features:

- `federation-with-rabbit` from federation project
- `odl-netvirt-openstack` from netvirt project

## Testing

### Unit Tests

End-to-end component service will test the federation plugin on top of the federation service.

### Integration Tests

None

## CSIT

The CSIT infrastructure will be enhanced to support connect/disconnect operations between sites using `update-federated-networks` RPC call.

A new federation suite will test L2 and L3 connectivity between remote sites and will be based on the existing L2/L3 connectivity suites. CSIT will load sites A,B and C in 1-node/3-node deployment options to run the following tests:

### 1 Install odl-netvirt-federation feature

- Basic L2 connectivity test within the site
- Basic L3 connectivity test within the site
- L2 connectivity between sites - expected to fail since sites are not connected
- L3 connectivity between sites - expected to fail since sites are not connected

## 2 Connect sites A,B

- Basic L2 connectivity test within the site
- L2 connectivity test between VMs in sites A,B
- L2 connectivity test between VMs in sites A,C and B,C - expected to fail since sites are not connected
- Basic L3 connectivity test within the site
- L3 connectivity test between VMs in sites A,B
- L3 connectivity test between VMs in sites A,C and B,C - expected to fail since sites are not connected

## 3 Connect site C to A,B

- L2 connectivity test between VMs in sites A,B B,C and A,C
- L3 connectivity test between VMs in sites A,B B,C and A,C
- Connectivity test between VMs in non-federated networks in sites A,B,C - expected to fail

## 4 Disconnect site C from A,B

- Repeat the test steps from 2 after C disconnect. Identical results expected.

## 5 Disconnect sites A,B

- Repeat the test steps from 1 after A,B disconnect. Identical results expected.

## 6 Federation cluster test

- Repeat test steps 1-5 while rebooting the ODLs between the steps similarly to the existing cluster suite.

## Documentation Impact

None.

## References

- [1] OpenDaylight Documentation Guide
- [2] Federation project
- [3] Federation service API
- [4] Support of VxLAN based connectivity across Datacenters
- [5] VNI based L2 switching, L3 forwarding and NATing
- [6] Cross site manager presentation ODL Summit 2016

## Table of Contents

- *DHCP Server Dynamic Allocation Pool*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*
  - *Usage*
    - \* *Features to Install*
    - \* *REST API*
    - \* *CLI*
  - *Implementation*
    - \* *Assignee(s)*
    - \* *Work Items*
  - *Dependencies*
  - *Testing*
    - \* *Unit Tests*
    - \* *Integration Tests*
    - \* *CSIT*
  - *Documentation Impact*
  - *References*

### 1.1.7 DHCP Server Dynamic Allocation Pool

[gerrit filter: [https://git.opendaylight.org/gerrit/#/q/topic:dhcp\\_server\\_pool](https://git.opendaylight.org/gerrit/#/q/topic:dhcp_server_pool)]

Extension of the ODL based DHCP server, which add support for dynamic address allocation to end point users, that are not controlled (known) by OpenStack Neutron. Each DHCP pool can be configured with additional information such as DNS servers, lease time (not yet), static allocations based on MAC address, etc.

The feature supports IPv4 only.

## Problem description

In a non-neutron northbounds environment e.g. SD-WAN solution (unimgr), there is currently no dynamic DHCP service for end-points or networks that are connected to OVS. Every DHCP packet that is received by the controller, the controller finds the neutron port based on the inport of the packet, extracts the ip which was allocated by neutron for that vm, and replies using that info. If the dhcp packet is from a non-neutron port, the packet won't even reach the controller.

## Use Cases

a DHCP packet that is received by the odl, from a port that is managed by Netvirt and was configured using the netvirt API, rather than the neutron API, in a way that there is no pre-allocated IP for network interfaces behind that port - will be handled by the DHCP dynamic allocation pool that is configured on the network associated with the receiving OVS port.

## Proposed change

We wish to forward to the controller, every dhcp packet coming from a non-neutron port as well (as long as it is configured to use the controller dhcp). Once a DHCP packet is recieved by the controller, the controller will check if there is already a pre-allocated address by checking if packet came from a neutron port. if so, the controller will reply using the information from the neutron port. Otherwise, the controller will find the allocation pool for the network which the packet came from and will allocate the next free ip. The operation of each allocation pool will be managed through the Genius ID Manager service that will support the allocation and release of IP addresses (ids), persistent mapping across controller restarts and more. Neutron IP allocations will be added to the relevant pools to avoid allocation of the same addresses.

The allocation pool DHCP server will support:

- DHCP methods: Discover, Request, Release, Decline and Inform (future)
- Allocation of a dynamic or specific (future) available IP address from the pool
- (future) Static IP address allocations
- (future) IP Address Lease Time + Rebinding and Renewal Time
- Classless Static Routes for each pool
- Domain names (future) and DNS for each pool
- (future) Probe an address before allocation
- (future) Relay agents

## Pipeline changes

This new rule in table 60 will be responsible for forwarding dhcp packets to the controller:

```
cookie=0x6800000, duration=121472.576s, table=60, n_packets=1, n_bytes=342, _  
→priority=49, udp, tp_src=68, tp_dst=67 actions=CONTROLLER:65535
```

## Yang changes

New YANG model to support the configuration of the DHCP allocation pools and allocations, per network and subnet.



- Allocation-Pool: configuration of allocation pool parameters like range, gateway and dns servers.
- Allocation-Instance: configuration of static IP address allocation and Neutron pre-allocated addresses, per MAC address.

Listing 1.3: dhcp\_allocation\_pool.yang

```

container dhcp_allocation_pool {
  config true;
  description "contains DHCP Server dynamic allocations";

  list network {
    key "network-id";
    leaf network-id {
      description "network (vlan-instance) id";
      type string;
    }
  }
  list allocation {
    key "subnet";
    leaf subnet {
      description "subnet for the dhcp to allocate ip addresses";
      type inet:ip-prefix;
    }
  }

  list allocation-instance {
    key "mac";
    leaf mac {
      description "requesting mac";
      type yang:phys-address;
    }
    leaf allocated-ip {
      description "allocated ip address";
      type inet:ip-address;
    }
  }
}

list allocation-pool {
  key "subnet";
  leaf subnet {
    description "subnet for the dhcp to allocate ip addresses";
    type inet:ip-prefix;
  }
  leaf allocate-from {
    description "low allocation limit";
    type inet:ip-address;
  }
  leaf allocate-to {
    description "high allocation limit";
    type inet:ip-address;
  }
  leaf gateway {
    description "default gateway for dhcp allocation";
    type inet:ip-address;
  }
  leaf-list dns-servers {
    description "dns server list";
    type inet:ip-address;
  }
  list static-routes {

```

```
        description "static routes list for dhcp allocation";
        key "destination";
        leaf destination {
            description "destination in CIDR format";
            type inet:ip-prefix;
        }
        leaf nexthop {
            description "router ip address";
            type inet:ip-address;
        }
    }
}
}
```

## Configuration impact

The feature is activated in the configuration (disabled by default).

adding **dhcp-dynamic-allocation-pool-enabled** leaf to dhcpservice-config:

Listing 1.4: dhcpservice-config.yang

```
container dhcpservice-config {
    leaf controller-dhcp-enabled {
        description "Enable the dhcpservice on the controller";
        type boolean;
        default false;
    }

    leaf dhcp-dynamic-allocation-pool-enabled {
        description "Enable dynamic allocation pool on controller dhcpservice";
        type boolean;
        default false;
    }
}
```

and netvirt-dhcpservice-config.xml:

```
<dhcpservice-config xmlns="urn:opendaylight:params:xml:ns:yang:dhcpservice:config">
  <controller-dhcp-enabled>false</controller-dhcp-enabled>
  <dhcp-dynamic-allocation-pool-enabled>false</dhcp-dynamic-allocation-pool-enabled>
</dhcpservice-config>
```

## Clustering considerations

Support clustering.

## Other Infra considerations

None.

## Security considerations

None.

## Scale and Performance Impact

None.

## Targeted Release

Carbon.

## Alternatives

Implement and maintain an external DHCP server.

## Usage

### Features to Install

This feature can be used by installing odl-netvirt-openstack. This feature doesn't add any new karaf feature.

## REST API

Introducing a new REST API for the feature

### Dynamic allocation pool

**URL:** /config/dhcp\_allocation\_pool:dhcp\_allocation\_pool/

### Sample JSON data

```
{ "dhcp_allocation_pool": {
  "network": [
    {
      "network-id": "d211a14b-e5e9-33af-89f3-9e43a270e0c8",
      "allocation-pool": [
        {
          "subnet": "10.1.1.0/24",
          "dns-servers": [
            "8.8.8.8"
          ],
          "gateway": "10.1.1.1",
          "allocate-from": "10.1.1.2",
          "allocate-to": "10.1.1.200",
          "static-routes": [
            {
              "destination": "5.8.19.24/16",
              "nexthop": "10.1.1.254"
            }
          ]
        }
      ]
    }
  ]
}
```

```
    ]  
  ]}}}}
```

## Static address allocation

**URL:** /config/dhcp\_allocation\_pool:dhcp\_allocation\_pool/

### Sample JSON data

```
{ "dhcp_allocation_pool": {  
  "network": [  
    {  
      "network-id": "d211a14b-e5e9-33af-89f3-9e43a270e0c8",  
      "allocation": [  
        {  
          "subnet": "10.1.1.0/24",  
          "allocation-instance": [  
            {  
              "mac": "fa:16:3e:9d:c6:f5",  
              "allocated-ip": "10.1.1.2"  
            }  
          ]  
        }  
      ]  
    }  
  ]  
}}
```

## CLI

None.

## Implementation

### Assignee(s)

**Primary assignee:** Shai Haim (shai.haim@hpe.com)

**Other contributors:** Alex Feigin (alex.feigin@hpe.com)

## Work Items

Here is the link for the Trello Card: <https://trello.com/c/0mgGyJuV/153-dhcp-server-dynamic-allocation-pool>

## Dependencies

None.

## Testing

### Unit Tests

N.A.

## Integration Tests

N.A.

## CSIT

N.A.

## Documentation Impact

??

## References

### Table of Contents

- *Discovery of directly connected PNFs in Flat/VLAN provider networks*
  - *Problem description*
    - \* *Subnet-Route*
    - \* *Aliveness monitor*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Subnet-route*
    - \* *Communication between VMs in tenant networks and PNFs in provider networks.*
    - \* *Communication between VMs and PNFs in different tenant networks.*
    - \* *ARP messages*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*
  - *Usage*
    - \* *Create external network with a subnet*
    - \* *Create internal networks with subnets*

- \* *Create a router instance and connect it to an internal subnet and an external subnet*
- \* *Create a router instance and connect it to two internal subnets*
- \* *Features to Install*
- \* *REST API*
- \* *CLI*
- *Implementation*
  - \* *Assignee(s)*
  - \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.8 Discovery of directly connected PNFs in Flat/VLAN provider networks

[https://git.opendaylight.org/gerrit/#/q/topic:directly\\_connected\\_pnf\\_discovery](https://git.opendaylight.org/gerrit/#/q/topic:directly_connected_pnf_discovery)

This feature enables discovering and directing traffic to Physical Network Functions (PNFs) in Flat/VLAN provider and tenant networks, by leveraging Subnet-Route feature.

#### Problem description

PNF is a device which has not been created by Openstack but connected to the hypervisors L2 broadcast domain and configured with ip from one of the neutron subnets.

Ideally, L2/L3 communication between VM instances and PNFs on flat/VLAN networks would be routed similarly to inter-VM communication. However, there are two main issues preventing direct communication to PNFs.

- L3 connectivity of tenant network and VLAN provider network, between VMs and PNFs. A VM is located in a tenant network, A PNF is located in a provider network (external network). Both networks are connected via a router. The only way for VMs to communicate with a PNF is via additional hop which is the external gateway, instead of directly.
- L3 connectivity between VMs and PNFs in two different tenant networks, connected by a router, which is not supported and has two problems. First, traffic initiated from a VMs towards a PNF is dropped because there isn't an appropriate rule in FIB table (table 21) to route that traffic. Second, in the other direction, PNFs are not able to resolve their default gateway.

We want to leverage the Subnet-Route and Aliveness-Monitor features in order to address the above issues.

## Subnet-Route

Today, Subnet-Route feature enables ODL to route traffic to a destination IP address, even for ip addresses that have not been statically configured by OpenStack, in the FIB table. To achieve that, the FIB table contains a flow that match all IP packets in a given subnet range. How that works?

- A flow is installed in the FIB table, matching on subnet prefix and vpn-id of the network, with a goto-instruction directing packets to table 22. There, packets are punted to the controller.
- ODL hold the packets, and initiate an ARP request towards the destination IP.
- Upon receiving ARP reply, ODL installs exact IP match flow in FIB table to direct all further traffic towards the newly learnt MAC of the destination IP

Current limitations of Subnet-Route feature:

- Works for BGPVPN only
- May cause traffic lost due to “swallowing” the packets punted from table 22.
- Uses the source MAC and source IP from the punted packet.

## Aliveness monitor

After ODL learns a mac that is associated with an ip address, ODL schedule an arp monitor task, with the purpose of verifying that the device is still alive and responding. This is done by periodically sending arp requests to the device.

Current limitation: Aliveness monitor was not designed for monitoring devices behind flat/VLAN provider network ports.

## Use Cases

- **L3 connectivity of tenant network and VLAN provider network, between VMs and PNFs.**
  - VMs in a private network, PNFs in external network
- L3 connectivity between VMs and PNFs in a two different tenant networks.

## Proposed change

### Subnet-route

- Upon OpenStack configuration of a Subnet in a provider network, a new vrf entry with subnet-route augmentation will be created.
- Upon associataion of neutron router with a subnet in a tenant network, a new vrf entry with subnet-route augmentation will be created.
- Upon receiving ARP reply, install exact IP match flow in FIB table to direct all further traffic towards the newly resolved PNF, on all relevant computes nodes, which will be discussed later
- Packets that had been punted to controller will be resubmitted to the openflow pipeline after installation of exact match flow.

### Communication between VMs in tenant networks and PNFs in provider networks.

In this scenario a VM in a private tenant network wants to communicate with a PNF in the (external) provider network

- The controller will hold the packets, and initiate an ARP request towards the PNF IP. an ARP request will have source MAC and IP the router gateway and will be sent from the NAPT switch.
- ARP packets will be punted from the NAPT switch only.
- Upon receiving ARP reply, install exact IP match flow in FIB table to direct all further traffic towards the newly resolved PNF, on all compute nodes that are associated with the external network.
- leveraging Aliveness monitor feature to monitor PNFs. The controller will send ARP requests from the NAPT switch.

### Communication between VMs and PNFs in different tenant networks.

In this scenario a VM and a PNF, in different private networks of the same tenant, wants to communicate. For each subnet prefix, a designated switch will be chosen to communicate directly with the PNFs in that subnet prefix. That means sending ARP requests to the PNFs and receiving their traffic.

**Note: IP traffic from VM instances will retain the src MAC of the VM instance, instead of replacing it with the router-interface-mac, in order to prevent MAC movements in the underlay switches. This is a limitation until NetVirt supports a MAC per hypervisor implementation.**

- A subnet flow will be installed in the FIB table, matching the subnet prefix and vpn-id of the router.
- ARP request will have a source MAC and IP of the router interface, and will be sent via the provider port in the designated switch.
- ARP packets will be punted from the designated switch only.
- Upon receiving an ARP reply, install exact IP match flow in FIB table to direct all further traffic towards the newly resolved PNF, on all computes related to the router
- ARP responder flow: a new ARP responder flow will be installed in the designated switch This flow will response for ARP requests from a PNF and the response MAC will be the router interface MAC. This flow will use the LPort-tag of the provider port.
- Split Horizon protection disabling: traffic from PNFs, arrives to the primary switch(via a provider port) due to the ARP responder rule described above, and will need to be directed to the proper compute of the designated VM (via a provider port). This require disabling the split horizon protection. In order to protects against infinite loops, the packet TTL will be decreased.
- leveraging Aliveness monitor, the controller will send ARP requests from the designated switch.

### ARP messages

ARP messages in the Flat/Vlan provider and tenant networks will be punted from a designated switch, in order to avoid a performance issue in the controller, of dealing with broadcast packets that may be received in multiple provider ports. In external networks this switch is the NAPT switch.

### Pipeline changes

First use-case depends on hairpinning spec [2], the flows presented here reflects that dependency.



## Egress traffic from VM with floating IP to an unresolved PNF in external network

- Packets in FIB table after translation to FIP, will match on subnet flow and will be punted to controller from Subnet Route table. Then, ARP request will be generated and be sent to the PNF. No flow changes are required in this part.

Classifier table (0) =>

Dispatcher table (17) l3vpn service: set vpn-id=router-id=>

GW Mac table (19) match: vpn-id=router-id, dst-mac=router-interface-mac =>

FIB table (21) match: vpn-id=router-id=>

Pre SNAT table (26) match: vpn-id=router-id, src-ip=vm-ip set  
vpn-id=ext-subnet-id, src-ip=fip =>

SNAT table (28) match: vpn-id=ext-subnet-id, src-ip=fip set src-mac=fip-mac =>

FIB table (21) match: vpn-id=ext-subnet-id, dst-ip=ext-subnet-ip =>

Subnet Route table (22): => Output to Controller

- After receiving ARP response from the PNF a new exact IP flow will be installed in table 21. No other flow changes are required.

Classifier table (0) =>

Dispatcher table (17) l3vpn service: set vpn-id=router-id=>

GW Mac table (19) match: vpn-id=router-id, dst-mac=router-interface-mac =>

FIB table (21) match: vpn-id=router-id=>

Pre SNAT table (26) match: vpn-id=router-id, src-ip=vm-ip set  
vpn-id=ext-subnet-id, src-ip=fip =>

SNAT table (28) match: vpn-id=ext-subnet-id, src-ip=fip set src-mac=fip-mac =>

FIB table (21) match: vpn-id=ext-subnet-id, dst-ip=pnf-ip, set  
dst-mac=pnf-mac, reg6=provider-lport-tag =>

Egress table (220) output to provider port

## Egress traffic from VM using NAPT to an unresolved PNF in external network

- Ingress-DPN is not the NAPT switch, no changes required. Traffic will be directed to NAPT switch and directed to the outbound NAPT table straight from the internal tunnel table

Classifier table (0) =>

Dispatcher table (17) l3vpn service: set vpn-id=router-id=>

GW Mac table (19) match: vpn-id=router-id, dst-mac=router-interface-mac =>

FIB table (21) match: vpn-id=router-id=>

Pre SNAT table (26) match: vpn-id=router-id=>

NAPT Group output to tunnel port of NAPT switch

- Ingress-DPN is the NAPT switch. Packets in FIB table after translation to NAPT, will match on subnet flow and will be punted to controller from Subnet Route table. Then, ARP request will be generated and be sent to the PNF. No flow changes are required.

Classifier table (0) =>

Dispatcher table (17) l3vpn service: set vpn-id=router-id=>

GW Mac table (19) match: vpn-id=router-id,dst-mac=router-interface-mac=>

FIB table (21) match: vpn-id=router-id=>

Pre SNAT table (26) match: vpn-id=router-id=>

Outbound NAPT table (46) match: src-ip=vm-ip,port=int-port set  
src-ip=router-gw-ip, vpn-id=router-gw-subnet-id,port=ext-port =>

NAPT PFIB tabl (47) match: vpn-id=router-gw-subnet-id=>

FIB table (21) match: vpn-id=ext-subnet-id, dst-ip=ext-subnet-ip=>

Subnet Route table (22) => Output to Controller

- After receiving ARP response from the PNF a new exact IP flow will be installed in table 21. No other changes required.

Classifier table (0) =>

Dispatcher table (17) l3vpn service: set vpn-id=router-id=>

GW Mac table (19) match: vpn-id=router-id,dst-mac=router-interface-mac=>

FIB table (21) match: vpn-id=router-id=>

Pre SNAT table (26) match: vpn-id=router-id=>

Outbound NAPT table (46) match: vpn-id=router-id TBD set vpn-id=external-net-id  
=>

NAPT PFIB table (47) match: vpn-id=external-net-id=>

FIB table (21) match: vpn-id=ext-network-id, dst-ip=pnf-ip set  
dst-mac=pnf-mac, reg6=provider-lport-tag=>

Egress table (220) output to provider port

## Egress traffic from VM in private network to an unresolved PNF in another private network

- Packet from a VM is punted to the controller, no flow changes are required.

Classifier table (0) =>

Dispatcher table (17) l3vpn service: set vpn-id=router-id=>

GW Mac table (19) match: vpn-id=router-id,dst-mac=router-interface-mac=>

FIB table (21) match: vpn-id=router-id dst-ip=subnet-ip=>

Subnet Route table (22): => Output to Controller

- After receiving ARP response from the PNF a new exact IP flow will be installed in table 21.

Classifier table (0) =>  
 Dispatcher table (17) l3vpn service: set vpn-id=router-id=>  
 GW Mac table (19) match: vpn-id=router-id,dst-mac=router-interface-mac=>  
 FIB table (21) match: vpn-id=router-id dst-ip=pnf-ip set dst-mac=pnf-mac,  
 reg6=provider-lport-tag=>  
 Egress table (220) output to provider port

## Ingress traffic to VM in private network from a PNF in another private network

- New flow in table 19, to distinguish our new use-case, in which we want to decrease the TTL of the packet

Classifier table (0) =>  
 Dispatcher table (17) l3vpn service: set vpn-id=router-id=>  
 GW Mac table (19) match: lport-tag=provider-port, vpn-id=router-id,  
 dst-mac=router-interface-mac, set split-horizon-bit = 0, decrease-ttl=>  
 FIB table (21) match: vpn-id=router-id dst-ip=vm-ip set dst-mac=vm-mac  
 reg6=provider-lport-tag=>  
 Egress table (220) output to provider port

## Yang changes

In odl-l3vpn module, adjacency-list grouping will be enhanced with the following field

```
grouping adjacency-list {
  list adjacency {
    key "ip_address";
    ...
    leaf phys-network-func {
      type boolean;
      default false;
      description "Value of True indicates this is an adjacency of a device in a
↪provider network";
    }
  }
}
```

An adjacency that is added as a result of a PNF discovery, is a primary adjacency with an empty next-hop-ip list. This is not enough to distinguish PNF at all times. This new field will help us identify this use-case in a more robust way.

## Configuration impact

A configuration mode will be available to turn this feature ON/OFF.

## Clustering considerations

None

## Other Infra considerations

None

## Security considerations

None

## Scale and Performance Impact

All traffic of PNFs in each subnet-prefix sends their traffic to a designated switch.

## Targeted Release

Carbon

## Alternatives

None

## Usage

### Create external network with a subnet

```
neutron net-create public-net -- --router:external --is-default --provider:network_  
↪type=flat  
--provider:physical_network=physnet1  
neutron subnet-create --ip_version 4 --gateway 10.64.0.1 --name public-subnet1  
↪<public-net-uuid> 10.64.0.0/16  
-- --enable_dhcp=False
```

### Create internal networks with subnets

```
neutron net-create private-net1  
neutron subnet-create --ip_version 4 --gateway 10.0.123.1 --name private-subnet1  
↪<private-net1-uuid>  
10.0.123.0/24  
neutron net-create private-net2  
neutron subnet-create --ip_version 4 --gateway 10.0.124.1 --name private-subnet2  
↪<private-net2-uuid>  
10.0.124.0/24
```

### Create a router instance and connect it to an internal subnet and an external subnet

This will allow communication with PNFs in provider network

```
neutron router-create router1
neutron router-interface-add <router1-uuid> <private-subnet1-uuid>
neutron router-gateway-set --fixed-ip subnet_id=<public-subnet1-uuid> <router1-uuid>
↪<public-net-uuid>
```

## Create a router instance and connect to it to two internal subnets

This will allow East/West communication between VMs and PNFs

```
neutron router-create router1
neutron router-interface-add <router1-uuid> <private-subnet1-uuid>
neutron router-interface-add <router1-uuid> <private-subnet2-uuid>
```

## Features to Install

odl-netvirt-openstack

## REST API

## CLI

## Implementation

## Assignee(s)

**Primary assignee:** Tomer Pearl <tomer.pearl@hpe.com>

**Other contributors:** Yakir Dorani <yakir.dorani@hpe.com>

## Work Items

- Configure subnet-route flows upon ext-net configuration / router association
- Solve traffic lost issues of punted packets from table 22
- Enable aliveness monitoring on external interfaces.
- Add ARP responder flow for L3-PNF
- Add ARP packet-in from primary switch only
- Disable split-horizon and enable TTL decrease for L3-PNF

## Dependencies

This feature depends on hairpinning feature [2]

## Testing

### Unit Tests

Unit tests will be added for the new functionality

### Integration Tests

### CSIT

Will need to see if a PNF could be simulated in CSIT

### Documentation Impact

### References

[1] [https://docs.google.com/presentation/d/1ByvEQXUtlyH-H7Bin6OBJNrHjOv-3hpHYzU6Sf6hDbA/edit#slide=id.g11657174d1\\_0\\_31](https://docs.google.com/presentation/d/1ByvEQXUtlyH-H7Bin6OBJNrHjOv-3hpHYzU6Sf6hDbA/edit#slide=id.g11657174d1_0_31) [2] <http://docs.opendaylight.org/en/latest/submodules/netvirt/docs/specs/hairpinning-flat-vlan.html>

#### Table of Contents

- *ECMP Support for BGP based L3VPN*
  - *Problem description*
    - \* *Use Cases*
  - *High-Level Components:*
  - *Proposed change*
    - \* *Pipeline changes*
      - *Local FIB entry/NextHop Group programming:*
      - *Remote FIB entry/NextHop Group programming:*
    - \* *YANG changes*
      - *L3VPN YANG changes*
      - *ODL-L3VPN YANG changes*
      - *ODL-FIB YANG changes*
    - \* *ECMP forwarding through multiple Compute Node and VMs*
    - \* *ECMP forwarding for dispersed VMs*
    - \* *ECMP forwarding for co-located VMs*
    - \* *ECMP forwarding through two DC-Gateways*
    - \* *ECMP for Intra-DC L3VPN communication*
    - \* *ECMP Path decision based on Internal/External Tunnel Monitoring*
    - \* *GRE tunnel state handling*

- \* *VxLAN tunnel state handling*
- \* *Assumptions*
- \* *Reboot Scenarios*
- \* *Clustering considerations*
- \* *Other Infra considerations*
- \* *Security considerations*
- \* *Scale and Performance Impact*
- \* *Targeted Release*
- \* *Alternatives*
- *Usage*
  - \* *Features to Install*
  - \* *REST API*
- *Implementation*
  - \* *Assignee(s)*
  - \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.9 ECMP Support for BGP based L3VPN

[https://git.opendaylight.org/gerrit/#/q/topic:l3vpn\\_ecmp](https://git.opendaylight.org/gerrit/#/q/topic:l3vpn_ecmp)

This Feature is needed for load balancing of traffic in a cloud and also redundancy of paths for resiliency in cloud.

#### Problem description

The current L3VPN implementation for BGP VPN doesn't support load balancing behavior for external routes through multiple DC-GWs and reaching starting route behind Nova VMs through multiple compute nodes.

This spec provides implementation details about providing traffic load balancing using ECMP for L3 routing and forwarding. The load balancing of traffic can be across virtual machines with each connected to the different compute nodes, DC-Gateways. ECMP also enables fast failover of traffic The ECMP forwarding is required for both inter-DC and intra-DC data traffic types. For inter-DC traffic, spraying from DC-GW to compute nodes & VMs for the traffic entering DC and spraying from compute node to DC-GWs for the traffic exiting DC is needed. For intra-DC traffic, spraying of traffic within DC across multiple compute nodes & VMs is needed. There should be tunnel monitoring (e.g. GRE-KA or BFD) logic implemented to monitor DC-GW /compute node GRE tunnels which helps to determine available ECMP paths to forward the traffic.

## Use Cases

- ECMP forwarding of traffic entering a DC (i.e. Spraying of DC-GW -> OVS traffic across multiple Compute Nodes & VMs). In this case, DC-GW can load balance the traffic if a `static route` can be reachable through multiple NOVA VMs (say VM1 and VM2 connected on different compute nodes) running some networking application (example: vRouter).
- ECMP forwarding of traffic exiting a DC (i.e. Spraying of OVS -> DC-GW traffic across multiple DC Gateways). In this case, a Compute Node can LB the traffic if `external route` can be reachable from multiple DC-GWs.
- ECMP forwarding of intra-DC traffic (i.e. Spraying of traffic within DC across multiple Compute Nodes & VMs) This is similar to UC1, but load balancing behavior is applied on remote Compute Node for intra-DC communication.
- OVS -> DC-GW tunnel status based ECMP for inter and intra-DC traffic. Tunnel status based on monitoring (BFD) is considered in ECMP path set determination.

## High-Level Components:

The following components of the Openstack - ODL solution need to be enhanced to provide ECMP support:

- OpenStack Neutron BGPVPN Driver (for supporting multiple RDs)
- OpenDaylight Controller (NetVirt VpnService)

We will review enhancements that will be made to each of the above components in following sections.

## Proposed change

The following components within OpenDaylight Controller needs to be enhanced:

- NeutronvpnManager
- VPN Engine (VPN Manager and VPN Interface Manager)
- FIB Manager

## Pipeline changes

### Local FIB entry/NextHop Group programming:

A `static route` (example: 100.0.0.0/24) reachable through two VMs connected with same compute node.

Table 0=>Table 17=>Table 19=>Table 21=>LB Group=>Local VM Group=>Table 220

```
cookie=0x80000003, duration=46.020s, table=21, n_packets=0, n_bytes=0, priority=34, ip,
↳ metadata=0x22e4/0xffffffffe, nw_dst=100.0.0.0/24 actions=write_actions(group:150002)
group_id=150002, type=select, bucket=weight:50, actions=group:150001, bucket=weight:50,
↳ actions=group:150000
group_id=150001, type=all, bucket=actions=set_field:fa:16:3e:34:ff:58->eth_dst,
↳ load:0x200->NXM_NX_REG6[], resubmit(, 220)
group_id=150000, type=all, bucket=actions=set_field:fa:16:3e:eb:61:39->eth_dst,
↳ load:0x100->NXM_NX_REG6[], resubmit(, 220)
```



## Remote FIB entry/Nexthop Group programming:

- A static route (example: 10.0.0.1/32) reachable through two VMs connected with different compute node.  
on remote compute node,

Table 0=>Table 17=>Table 19=>Table 21=>LB Group=>VxLAN port

```
cookie=0x8000003, duration=46.020s, table=21, n_packets=0, n_bytes=0, priority=34,
↳ip,metadata=0x222e4/0xffffffff, nw_dst=10.0.0.1 actions=set_field:0xEF->tun_id,
↳group:150003
group_id=150003,type=select,bucket=weight:50,actions=output:1,bucket=weight:50,
↳actions=output:2
```

on local compute node,

Here, From LB group, packets flow through local VM and VxLAN port

Table 0=>Table 17=>Table 19=>Table 21=>LB Group=>Local VM Group=>Table 220

.....=> VxLAN port

```
cookie=0x8000003, duration=46.020s, table=21, n_packets=0, n_bytes=0,
↳priority=34,ip,metadata=0x222e4/0xffffffff, nw_dst=10.0.0.1
↳actions=group:150003
group_id=150003,type=select,bucket=weight:50,group=150001,bucket=weight:50,
↳actions=set_field:0xEF->tun_id, output:2
group_id=150001,type=all,bucket=actions=set_field:fa:16:3e:34:ff:58->eth_dst,
↳load:0x200->NXM_NX_REG6[], resubmit(,220)
```

- An external route (example: 20.0.0.1/32) reachable through two DC-GWs.

Table 0=>Table 17=>Table 19=>Table 21=>LB Group=>GRE port

```
cookie=0x8000003, duration=13.044s, table=21, n_packets=0, n_bytes=0,priority=42,
↳ip,metadata=0x222ec/0xffffffff,nw_dst=20.0.0.1 actions=load:0x64->NXM_NX_REG0[0.
↳.19],load:0xc8->NXM_NX_REG1[0..19],group:150111
group_id=150111,type=select,bucket=weight:50,actions=push_mpls:0x8847, move:NXM_
↳NX_REG0[0..19]->OXM_OF_MPLS_LABEL[],output:3, bucket=weight:50,actions=push_
↳mpls:0x8847,move:NXM_NX_REG1[0..19]->OXM_OF_MPLS_LABEL[],output:4
```

## YANG changes

Changes will be needed in l3vpn.yang, odl-l3vpn.yang and odl-fib.yang to support ECMP functionality.

## L3VPN YANG changes

route-distinguisher type is changed from leaf to leaf-list in vpn-af-config grouping in l3vpn.yang.

Listing 1.5: l3vpn.yang

```
grouping vpn-af-config {
  description "A set of configuration parameters that is applicable to both IPv4
↳and
```

```
IPv6 address family for a VPN instance .";

leaf-list route-distinguisher {
    description "The route-distinguisher command configures a route_
↪distinguisher (RD)
    for the IPv4 or IPv6 address family of a VPN instance.
    Format is ASN:nn or IP-address:nn.";
    config "true";
    type string{
        length "3..21";
    }
}
}
```

## ODL-L3VPN YANG changes

- Add `vrf-id` (RD) in adjacency list in `odl-l3vpn.yang`.

Listing 1.6: `odl-l3vpn.yang`

```
grouping adjacency-list {
    list adjacency{
        key "ip_address";
        leaf-list next-hop-ip-list { type string; }
        leaf ip_address {type string;}
        leaf primary-adjacency {
            type boolean;
            default false;
            description "Value of True indicates this is a primary adjacency";
        }

        leaf label { type uint32; config "false"; } /*optional*/
        leaf mac_address {type string;} /*optional*/
        leaf vrf-id {type string;}
    }
}
```

- `vpn-to-extraroute` have to be updated with multiple RDs (`vrf-id`) when extra route from VMs connected with different compute node and when connected on same compute node, just use same RD and update `nexthop-ip-list` with new VM IP address like below.

Listing 1.7: `odl-l3vpn.yang`

```
container vpn-to-extraroutes {
    config false;
    list vpn-extraroutes {
        key "vpn-name";
        leaf vpn-name {
            type uint32;
        }

        list extra-routes {
            key "vrf-id";
            leaf vrf-id {
                description "The vrf-id command configures a route_
↪distinguisher (RD) for the IPv4
```

```

        or IPv6 address family of a VPN instance or vpn instance name_
↪for
    internal vpn case.";
    type string;
}

list route-paths {
    key "prefix";
    leaf prefix {type string;}
    leaf-list nexthop-ip-list {
        type string;
    }
}
}
}
}

```

- To manage RDs for extra with multiple next hops, the following YANG model is required to advertise (or) withdraw the extra routes with unique NLRI accordingly.

Listing 1.8: odl-l3vpn.yang

```

container extraroute-routedistinguishers-map {
    config true;
    list extraroute-routedistinguishers {
        key "vpnid";
        leaf vpnid {
            type uint32;
        }

        list dest-prefixes {
            key "dest-prefix";
            leaf dest-prefix {
                type string;
                mandatory true;
            }

            leaf-list route-distinguishers {
                type string;
            }
        }
    }
}

```

## ODL-FIB YANG changes

- When Quagga BGP announces route with multiple paths, then it is ODL responsibility to program Fib entries in all compute nodes where VPN instance blueprint is present, so that traffic can be load balanced between these two DC gateways. It requires changes in existing odl-fib.yang model (like below) to support multiple routes for same destination IP prefix.

Listing 1.9: odl-fib.yang

```

grouping vrfEntries {
    list vrfEntry {
        key "destPrefix";
    }
}

```

```
    leaf destPrefix {
        type string;
        mandatory true;
    }

    leaf origin {
        type string;
        mandatory true;
    }

    list route-paths {
        key "nexthop-address";
        leaf nexthop-address {
            type string;
            mandatory true;
        }

        leaf label {
            type uint32;
        }
    }
}
```

- New YANG model to update load balancing next hop group buckets according to VxLAN/GRE tunnel status [Note that these changes are required only if watch\_port in group bucket is not working based on tunnel port liveness monitoring affected by the BFD status]. When one of the VxLAN/GRE tunnel is going down, then retrieve nexthop-key from dpid-l3vpn-lb-nexthops by providing tep-device-ids from src-info and dst-info of StateTunnelList while handling its update DCN. After retrieving next hop key, fetch target-device-id list from l3vpn-lb-nexthops and reprogram VxLAN/GRE load balancing group in each remote Compute Node based on tunnel state between source and destination Compute Node. Similarly, when tunnel comes up, then logic have to be rerun to add its bucket back into Load balancing group.

Listing 1.10: odl-fib.yang

```
container l3vpn-lb-nexthops {
    config false;
    list nexthops {
        key "nexthop-key";
        leaf group-id { type string; }
        leaf nexhop-key { type string; }
        leaf-list target-device-id { type string;
            //dpId or ip-address }
    }
}

container dpid-l3vpn-lb-nexthops {
    config false;
    list dpn-lb-nexthops {
        key "src-dp-id dst-device-id";
        leaf src-dp-id { type uint64; }
        leaf dst-device-id { type string;
            //dpId or ip-address }
        leaf-list nexthop-keys { type string; }
    }
}
```

## ECMP forwarding through multiple Compute Node and VMs

In some cases, extra route can be added which can have reachability through multiple Nova VMs. These VMs can be either connected on same compute node (or) different Compute Nodes. When VMs are in different compute nodes, DC-GW should learn all the route paths such that ECMP behavior can be applied for these multi path routes. When VMs are co-located in same compute node, DC-GW will not perform ECMP and compute node performs traffic splitting instead.

## ECMP forwarding for dispersed VMs

When configured extra route are reached through nova VMs which are connected with different compute node, then it is ODL responsibility to advertise these multiple route paths (but with same `MPLS label`) to Quagga BGP which in turn sends these routes into DC-GW. But DC-GW replaces the existing route with a new route received from the peer if the NLRI (prefix) is same in the two routes.

This is true even when multipath is enabled on the DC-GW and it is as per standard BGP RFC 4271, Section 9 UPDATE Message Handling. Hence the route is lost in DC-GW even before path computation for multipath is applied. This scenario is solved by adding multiple route distinguisher (RDs) for the vpn instance and let ODL uses the list of RDs to advertise the same prefix with different BGP NHs. Multiple RDs will be supported only for BGP VPNs.

## ECMP forwarding for co-located VMs

When extra routes on VM interfaces are connected with same compute node, LFIB/FIB and Terminating service table flow entries should be programmed so that traffic can be load balanced between local VMs. This can be done by creating load balancing next hop group for each vpn-to-extraroute (if nexthop-ip-list size is greater than 1) with buckets pointing to the actual VMs next hop group on source Compute Node. Even for the co-located VMs, VPN interface manager should assign separate RDs for each adjacency of same dest IP prefix and let route can be advertised again to Quagga BGP with same next hop (TEP IP address). This will enable DC-Gateway to realize ECMP behavior when an IP prefix can be reachable through multiple co located VMs on one Compute Node and an another VM connected on different Compute Node.

To create load balancing next hop group, the dest IP prefix is used as the key to generate group id. When any of next hop is removed, then adjust load balancing nexthop group so that traffic can be sent through active next hops.

## ECMP forwarding through two DC-Gateways

The current ITM implementation provides support for creating multiple GRE tunnels for the provided list of DC-GW IP addresses from compute node. This should help in creating corresponding load balancing group whenever Quagga BGP is advertising two routes on same IP prefix pointing to multiple DC GWs. The group id of this load balancing group can be derived from sorted order of DC GW TEP IP addresses with the following format `dc_gw_tep_ip_address_1: dc_gw_tep_ip_address_2`. This will be useful when multiple external IP prefixes share the same next hops. The load balancing next hop group buckets is programmed according to sorted remote end point DC-Gateway IP address. The support of action `move:NXM_NX_REG0(1)->MPLS label` is not supported in ODL openflowplugin. It has to be implemented. Since there are two DC gateways present for the data center, it is possible that multiple equal cost routes are supplied to ODL by Quagga BGP like Fig 2. The current Quagga BGP doesn't have multipath support and it will be done. When Quagga BGP announces route with multiple paths, then it is ODL responsibility to program Fib entries in all compute nodes where VPN instance blueprint is present, so that traffic can be load balanced between these two DC gateways. It requires changes in existing `odl-fib.yang` model (like below) to support multiple routes for same destination IP prefix.

BGPManager should be able to create vrf entry for the advertised IP prefix with multiple route paths. `VrfEntryListener` listens to DCN on these vrf entries and program Fib entries (21) based on number route paths available for given IP

prefix. For the given (external) destination IP prefix, if there is only one route path exists, use the existing approach to program FIB table flow entry matches on (vpnId, ipv4\_dst) and actions with push MPLS label and output to gre tunnel port. For the given (external) destination IP prefix, if there are two route paths exist, then retrieve next hop ip address from routes list in the same sorted order (i.e. using same logic which is used to create buckets for load balancing next hop group for DC- Gateway IP addresses), then program FIB table flow entry with an instruction like Fig 3. It should have two set field actions where first action sets MPLS label to NX\_REG0 for first sorted DC-GW IP address and second action sets MPLS label to NX\_REG1 for the second sorted DC-GW IP address. When more than two DC Gateways are used, then more number of NXM Registries have to be used to push appropriate MPLS label before sending it to next hop group. It needs operational DS container to have mapping between DC Gateway IP address and NXM\_REG. When one of the route is withdrawn for the IP prefix, then modify the FIB table flow entry with push MPLS label and output to the available gre tunnel port.

## ECMP for Intra-DC L3VPN communication

ECMP within data center is required to load balance the data traffic when extra route can be reached through multiple next hops (i.e. Nova VMs) when these are connected with different compute nodes. It mainly deals with how Compute Nodes can spray the traffic when dest IP prefix can be reached through two or more VMs (next hops) which are connected with multiple compute nodes.

When there are multiple RDs (if VPN is of type BGP VPN) assigned to VPN instance so that VPN engine can be advertise IP route with different RDs to achieve ECMP behavior in DC-GW as mentioned before. But for intra-DC, this doesn't make any more sense since it's all about programming remote FIB entries on compute nodes to achieve data traffic spray behavior.

Irrespective of RDs, when multiple next hops (which are from different Compute Nodes) are present for the extra-route adjacency, then FIB Manager has to create load balancing next hop group in remote compute node with buckets pointing with targeted Compute Node VxLAN tunnel ports.

To allocate group id for this load balancing next hop, the same destination IP prefix is used as the group key. The remote FIB table flow should point to this next hop group after writing prefix label into tunnel\_id. The bucket weight of remote next hop is adjusted according to number of VMs associated to given extra route and on which compute node the VMs are connected. For example, two compute node having one VM each, then bucket weight is 50 each. One compute node having two VMs and another compute node having one VM, then bucket weight is 66 and 34 each. The hop-count property in vrfEntry data store helps to decide what is the bucket weight for each bucket.

## ECMP Path decision based on Internal/External Tunnel Monitoring

ODL will use GRE-KA or BFD protocol to implement monitoring of GRE external tunnels. This implementation detail is out of scope in this document. Based on the tunnel state, GRE Load Balancing Group is adjusted accordingly as mentioned like below.

### GRE tunnel state handling

As soon as GRE tunnel interface is created in ODL, interface manager uses alivenessmonitor to monitor the GRE tunnels for its liveness using GRE Keep-alive protocol. When tunnel state changes, it has to be handled accordingly to adjust above load balancing group so that data traffic is sent to only active DC-GW tunnel. This can be done with listening to update StateTunnelList DCN.

When one GRE tunnel is operationally going down, then retrieve the corresponding bucket from the load balancing group and delete it. When GRE tunnel comes up again, then add bucket back into load balancing group and reprogram it.

When both GRE tunnels are going down, then just recreate load balancing group with empty. Withdraw the routes from that particular DC-GW. With the above implementation, there is no need of modifying Fib entries for GRE tunnel state changes.

But when BGP Quagga withdrawing one of the route for external IP prefix, then reprogram FIB flow entry (21) by directly pointing to output=<gre\_port> after pushing MPLS label.

### VxLAN tunnel state handling

Similarly, when VxLAN tunnel state changes, the Load Balancing Groups in Compute Nodes have to be updated accordingly so that traffic can flow through active VxLAN tunnels. It can be done by having config mapping between target data-path-id to next hop group Ids and vice versa.

For both GRE and VxLAN tunnel monitoring, L3VPN has to implement the following YANG model to update load balancing next hop group buckets according to tunnel status.

When one of the VxLAN/GRE tunnel is going down, then retrieve nexthop-key from dpid-l3vpn-lb-nexthops by providing tep-device-ids from src-info and dst-info of StateTunnelList while handling its update DCN.

After retrieving next hop key, fetch target-device-id list from l3vpn-lb-nexthops and reprogram VxLAN/GRE load balancing group in each remote Compute Node based on tunnel state between source and destination Compute Node. Similarly, when tunnel comes up, then logic have to be rerun to add its bucket back into Load balancing group.

### Assumptions

The support for action move:NXM\_NX\_REG0(1) -> MPLS label is already available in Compute Node.

### Reboot Scenarios

This feature support all the following Reboot Scenarios for EVPN:

- Entire Cluster Reboot
- Leader PL reboot
- Candidate PL reboot
- OVS Datapath reboots
- Multiple PL reboots
- Multiple Cluster reboots
- Multiple reboots of the same OVS Datapath.
- Openstack Controller reboots

### Clustering considerations

The feature should operate in ODL Clustered environment reliably.

### Other Infra considerations

N.A.

## Security considerations

N.A.

## Scale and Performance Impact

Not covered by this Design Document.

## Targeted Release

Carbon.

## Alternatives

Alternatives considered and why they were not selected.

## Usage

## Features to Install

This feature doesn't add any new karaf feature.

## REST API

## Implementation

## Assignee(s)

### Primary assignee(s):

- Manu B <manu.b@ericsson.com>
- Kency Kurian <kency.kurian@ericsson.com>
- Gobinath <gobinath@ericsson.com>
- P Govinda Rajulu <p.govinda.rajulu@ericsson.com>

### Other contributors:

- Periyasamy Palanisamy <periyasamy.palanisamy@ericsson.com>

## Work Items

The Trello cards have already been raised for this feature under l3vpn\_ecmp.

Link for the Trello Card: <https://trello.com/c/8E3LWIkq/121-ecmp-support-for-bgp-based-l3vpn-l3vpn-ecmp>



## Dependencies

Quagga BGP multipath support and APIs. This is needed to support when two DC-GW advertises routes for same external prefix with different route labels GRE tunnel monitoring. This is need to implement ECMP forwarding based on MPLSoGRE tunnel state. Support for action move:NXM\_NX\_REG0(1) -> MPLS label in ODL openflowplugin

## Testing

Capture details of testing that will need to be added.

## Unit Tests

Appropriate UTs will be added for the new code coming in once framework is in place.

## Integration Tests

There won't be any Integration tests provided for this feature.

## CSIT

CSIT will be enhanced to cover this feature by providing new CSIT tests.

## Documentation Impact

This will require changes to User Guide and Developer Guide.

## References

- <https://docs.google.com/document/d/1KRxrIGCLCBuz2D8f8IhU2I84VrM5EMa1Y7Scjb6qEKw>

### Table of Contents

- *Element Counters*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*

- \* *Targeted Release*
  - \* *Alternatives*
- *Usage*
  - \* *Features to Install*
  - \* *REST API*
  - \* *CLI*
- *Implementation*
  - \* *Assignee(s)*
  - \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.10 Element Counters

<https://git.opendaylight.org/gerrit/#/q/element-counters>

This feature depends on the Netvirt statistics feature.

This feature enables collecting statistics on filtered traffic passed from/to a network element. For example: traffic outgoing/incoming from a specific IP, tcp traffic, udp traffic, incoming/outgoing traffic only.

#### Problem description

Collecting statistics on filtered traffic sent to/from a VM is currently not possible.

#### Use Cases

- Tracking East/West communication between local VMs.
- Tracking East/West communication between VMs that are located in different compute nodes.
- Tracking communication between a local VM and an IP located in an external network.
- Tracking TCP/UDP traffic sent from/to a VM.
- Tracking dropped packets between 2 VMs.

## Proposed change

The Netvirt Statistics Plugin will receive requests regarding element filtered counters. A new service will be implemented (“CounterService”), and will be associated with the relevant interfaces (either ingress side, egress sides or both of them).

- Ingress traffic: The service will be the first one in the pipeline after the Ingress ACL service.
- Egress traffic: The service will be the last one after the Egress ACL service.
- The input for counters request regarding VM A, and incoming and outgoing traffic from VM B, will be VM A interface uuid and VM B IP.
- The input can also include other filters like TCP only traffic, UDP only traffic, incoming/outgoing traffic.
- In order to track dropped traffic between VM A and VM B, the feature should be activated on both VMS (either in the same compute node or in different compute nodes). service binding will be done on both VMs relevant interfaces.
- If the counters request involves an external IP, service binding will be done only on the VM interface.
- Adding/Removing the “CounterService” should be dynamic and triggered by requesting element counters.

The Statistics Plugin will use OpenFlow flow statistic requests for these new rules, allowing it to gather statistics regarding the traffic between the 2 elements. It will be responsible to validate and filter the counters results.

## Pipeline changes

Two new tables will be used: table 219 for outgoing traffic from the VM, and table 249 for incoming traffic from the VM. In both ingress and egress pipelines, the counter service will be just after the appropriate ACL service. The default rule will resubmit traffic to the appropriate dispatcher table.

Assuming we want statistics on VM A traffic, received or sent from VM B.

### VM A Outgoing Traffic (vm interface)

In table 219 traffic will be matched against dst-ip and lport tag.

```
Ingress dispatcher table (17): match:  lport-tag=vmA-interface, actions:  go to
table 219 =>
```

```
Ingress counters table (219): match:  dst-ip=vmB-ip, lport-tag=vmA-interface,
actions:  resubmit to table 17 =>
```

### VM A Incoming Traffic (vm interface)

In table 249 traffic will be matched against src-ip and lport tag.

```
Egress dispatcher table (220): match:  lport-tag=vmA-interface, actions:  go to
table 249 =>
```

```
Egress counters table (249): match:  lport-tag=vmA-interface, src-ip=vmB-ip,
actions:  resubmit to table 220 =>
```

Assuming we want statistics on VM A incoming TCP traffic.

### VM A Outgoing Traffic (vm interface)

Egress dispatcher table (220): match: lport-tag=vmA-interface, actions: go to table 249 =>

Egress counters table (249): match: lport-tag=vmA-interface, tcp, actions: resubmit to table 220 =>

Assuming we want statistics on VM A outgoing UDP traffic.

### VM A Incoming traffic (vm interface)

Ingress dispatcher table (17): match: lport-tag=vmA-interface, actions: go to table 219 =>

Ingress counters table (219): match: lport-tag=vmA-interface, udp, actions: resubmit to table 17 =>

Assuming we want statistics on all traffic sent to VM A port.

### VM A Incoming traffic (vm interface)

Ingress dispatcher table (17): match: lport-tag=vmA-interface, actions: go to table 219 =>

Ingress counters table (219): match: lport-tag=vmA-interface, actions: resubmit to table 17 =>

### Yang changes

Netvirt Statistics module will be enhanced with the following RPC:

```
grouping result {
  list counterResult {
    key id;
    leaf id {
      type string;
    }
    list groups {
      key name;
      leaf name {
        type string;
      }
      list counters {
        key name;
        leaf name {
          type string;
        }
        leaf value {
          type uint64;
        }
      }
    }
  }
}
```

```

grouping filters {
    leaf-list groupFilters {
        type string;
    }
    leaf-list counterFilter {
        type string;
    }
}

grouping elementRequestData {
    container filters {
        container tcpFilter {
            leaf on {
                type boolean;
            }
            leaf srcPort {
                type int32;
                default -1;
            }
            leaf dstPort {
                type int32;
                default -1;
            }
        }

        container udpFilter {
            leaf on {
                type boolean;
            }
            leaf dstPort {
                type int32;
                default -1;
            }
            leaf srcPort {
                type int32;
                default -1;
            }
        }

        container ipFilter {
            leaf ip {
                type string;
                default "";
            }
        }
    }
}

container elementCountersRequestConfig {
    list counterRequests {
        key "requestId";
        leaf requestId {
            type string;
        }
        leaf lportTag {
            type int32;
        }
        leaf dpn {

```

```
        type uint64;
    }
    leaf portId {
        type string;
    }
    leaf trafficDirection {
        type string;
    }
    uses elementRequestData;
}

}

rpc acquireElementCountersRequestHandler {
    input {
        leaf portId {
            type string;
        }
        container incomingTraffic {
            uses elementRequestData;
        }
        container outgoingTraffic {
            uses elementRequestData;
        }
        uses filters;
    }
    output {
        leaf incomingTrafficHandler {
            type string;
        }
        leaf outgoingTrafficHandler {
            type string;
        }
    }
}

rpc releaseElementCountersRequestHandler {
    input {
        leaf handler {
            type string;
        }
    }
    output {
    }
}

rpc getElementCountersByHandler {
    input {
        leaf handler {
            type string;
        }
    }
    output {
        uses result;
    }
}
```

## Configuration impact

The described above YANG model will be saved in the data store.

## Clustering considerations

None

## Other Infra considerations

None

## Security considerations

None

## Scale and Performance Impact

Since adding the new service is done by a request (as well as removing it), not all packets will be sent to the new tables described above.

## Targeted Release

Carbon

## Alternatives

None

## Usage

- Create router, network, 2 VMS, VXLAN tunnel.
- Connect to each one of the VMs and send ping to the other VM.
- Use REST to get the statistics.

Run the following to get interface ids:

```
http://10.0.77.135:8181/restconf/operational/ietf-interfaces:interfaces-state/
```

Choose VM B interface and use the following REST in order to get the statistics: Assuming VM A IP = 1.1.1.1, VM B IP = 2.2.2.2

Acquire counter request handler:

```
10.0.77.135:8181/restconf/operations/statistics-
↪plugin:acquireElementCountersRequestHandler, {"input":{"portId":"4073b4fe-a3d5-47c0-
↪b37d-4fb9db4be9b1", "incomingTraffic":{"filters":{"ipFilter":{"ip":"1.1.3.9"}}}},
↪headers={Authorization=Basic YWRtaW46YWRtaW4=, Cache-Control=no-cache, Content-
↪Type=application/json}}
```

Release handler:

```
10.0.77.135:8181/restconf/operations/statistics-  
→plugin:releaseElementCountersRequestHandler, input={"input":{"handler":"1"}},  
→headers={Authorization=Basic YWRtaW46YWRtaW4=, Cache-Control=no-cache, Content-  
→Type=application/json}]
```

Get counters:

```
10.0.77.135:8181/restconf/operations/statistics-plugin:getElementCountersByHandler,  
→input={"input":{"handler":"1"}}, headers={Authorization=Basic YWRtaW46YWRtaW4=,  
→Cache-Control=no-cache, Content-Type=application/json}]
```

Example counters output:

```
{  
  "output": {  
    "counterResult": [  
      {  
        "id": "SOME UNIQUE ID",  
        "groups": [  
          {  
            "name": "Duration",  
            "counters": [  
              {  
                "name": "durationNanoSecondCount",  
                "value": 298000000  
              },  
              {  
                "name": "durationSecondCount",  
                "value": 10369  
              }  
            ]  
          },  
          {  
            "name": "Bytes",  
            "counters": [  
              {  
                "name": "bytesTransmittedCount",  
                "value": 648  
              },  
              {  
                "name": "bytesReceivedCount",  
                "value": 0  
              }  
            ]  
          },  
          {  
            "name": "Packets",  
            "counters": [  
              {  
                "name": "packetsTransmittedCount",  
                "value": 8  
              },  
              {  
                "name": "packetsReceivedCount",  
                "value": 0  
              }  
            ]  
          }  
        ]  
      }  
    ]  
  }  
}
```



```
    }  
  ]  
}  
]  
}  
]  
}  
]
```

## Features to Install

odl-netvirt-openstack

## REST API

## CLI

## Implementation

## Assignee(s)

**Primary assignee:** Guy Regev <guy.regev@hpe.com>

**Other contributors:** TBD

## Work Items

<https://trello.com/c/88MnwGwb/129-element-to-element-counters>

- Add new service in Genius.
- Implement new rules installation.
- Update Netvirt Statistics module to support the new counters request.

## Dependencies

None

## Testing

## Unit Tests

## Integration Tests

## CSIT

## Documentation Impact

## References

Netvirt statistics feature: <https://git.opendaylight.org/gerrit/#/c/50164/8>

### Table of Contents

- *Hairpinning of floating IPs in flat/VLAN provider networks*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*
  - *Usage*
    - \* *Create external network with two subnets*
    - \* *Create internal networks with subnets*
    - \* *Create two router instances and connect each router to one internal subnet and one external subnet*
    - \* *Create router instance connected to both external subnets and the remaining internal subnets*
    - \* *Create floating ips from both subnets*
    - \* *Create 2 VM instance in each subnet and associate with floating ips*
    - \* *Connectivity tests*
    - \* *Features to Install*
    - \* *REST API*
    - \* *CLI*

- *Implementation*
  - \* *Assignee(s)*
  - \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.11 Hairpinning of floating IPs in flat/VLAN provider networks

<https://git.opendaylight.org/gerrit/#/q/topic:hairpinning>

This feature enables VM instances connected to the same router to communicate with each other using their floating ip addresses directly without traversing via the external gateway.

#### Problem description

Local and East/West communication between VMs using floating ips for flat/VLAN provider types is not handled internally by the pipeline currently. As a result, this type of traffic is mistakenly classified as North/South and routed to the external network gateway.

Today, SNATted traffic to flat/VLAN network is routed directly to the external gateway after traversing the SNAT/outbound NAPT pipeline using OF group per external network subnet. The group itself sets the destination mac as the mac address of the external gw associated with the floating ip/ router gw and output to the provider network port via the egress table. This workflow would be changed to align with the VxLAN provider type and direct SNATted traffic back to the FIB where the destination can then resolved to be floating ip on local or remote compute node.

#### Use Cases

- Local and East/West communication between VMs co-located on the same compute node using associated floating ip.
- Local and East/West communication between VMs located on different compute nodes using associated floating ip.

#### Proposed change

- The vpn-id used for classification of floating ips and router gateway external addresses in flat/VLAN provider networks is based on the external network id. It will be changed to reflect the subnet id associated with the floating ip/router gateway. This will allow traffic from the SNAT/outbound NAPT table to be resubmitted back to the FIB while preserving the subnet id.
- Each floating ip already has VRF entry in the fib table. The vpn-id of this entry will also be based on the subnet id of the floating ip instead of the external network id. If the VM associated with the floating ip is located on

remote compute node, the traffic will be routed to the remote compute based on the provider network of the subnet from which the floating ip was allocated e.g. if the private network is VxLAN and the external network is VLAN provider, traffic to floating ip on remote compute node will be routed to the provider port associated with the VLAN provider and not the tunnel associated with the VxLAN provider.

- In the FIB table of the egress node, the destination mac will be replaced with the mac address of the floating ip in case of routing to remote compute node. This will allow traffic from flat/VLAN provider enter the L3 pipeline for DNAT of the floating ip.
- Default flow will be added to the FIB table for each external subnet-id. If no floating ip match was found in the FIB table for the subnet id, the traffic will be sent to the group of the external subnet. Each group entry will perform the following: (a) replace the destination mac address to the external gateway mac address (b) send the traffic to the provider network via the egress table.
- Ingress traffic from flat/VLAN provider network is bounded to L3VPN service using vpn-id of the external network id. To allow traffic classification based on subnet id for floating ips and router gateway ips, the GW MAC table will replace the vpn-id of the external network with the vpn-id of the subnet id of the floating ip. For ingress traffic to router gateway mac, the vpn-id of the correct subnet will be determined at the FIB table based on the router gateway fixed ip.
- A new model will be introduced to contain the new vpn/subnet associations - `odl-nat:subnets-networks`. This model will be filled only for external flat/VLAN provider networks and will take precedence over `odl-nat:external-networks` model for selection of vpn-id. BGPVPN use cases won't be affected by these changes as this model will not be applicable for these scenarios.

## Pipeline changes

### Egress traffic from VM with floating IP to the internet

- For Pre SNAT, SNAT, FIB tables the vpn-id will be based on the subnet-id of the floating ip
- Packets from SNAT table resubmitted back to the FIB rather than straight to the external network subnet-id group. In the FIB table it should be matched against a new flow with lower priority than any other flow containing dst-ip match. Traffic will be redirected based on the vpn-id of the floating ip subnet to the external network subnet-id group.

Classifier table (0) =>

Dispatcher table (17) l3vpn service: set vpn-id=router-id=>

GW Mac table (19) match: vpn-id=router-id, dst-mac=router-interface-mac =>

FIB table (21) match: vpn-id=router-id=>

Pre SNAT table (26) match: vpn-id=router-id, src-ip=vm-ip set  
vpn-id=fip-subnet-id, src-ip=fip =>

SNAT table (28) match: vpn-id=fip-subnet-id, src-ip=fip set src-mac=fip-mac =>

FIB table (21) match: vpn-id=fip-subnet-id=>

Subnet-id group: set dst-mac=ext-subnet-gw-mac, reg6=provider-lport-tag =>

Egress table (220) output to provider network

### Ingress traffic from the internet to VM with floating IP

- For GW MAC, FIB table the vpn-id will be based on the subnet-id of the floating ip

```

Classifier table (0) =>
Dispatcher table (17) l3vpn service: set vpn-id=ext-net-id=>
GW Mac table (19) match: vpn-id=ext-net-id,dst-mac=floating-ip-mac set
vpn-id=fip-subnet-id=>
FIB table (21) match: vpn-id=fip-subnet-id,dst-ip=fip=>
Pre DNAT table (25) match: dst-ip=fip set vpn-id=router-id,dst-ip=vm-ip=>
DNAT table (27) match: vpn-id=router-id,dst-ip=vm-ip=>
FIB table (21) match: vpn-id=router-id,dst-ip=vm-ip=>
Local Next-Hop group: set dst-mac=vm-mac, reg6=vm-lport-tag=>
Egress table (220) output to VM port

```

### Egress traffic from VM with no associated floating IP to the internet - NAPT switch

- For Outbound NAPT, NAPT PFIB and FIB tables the vpn-id will be based on the subnet-id of the router gateway
- Packets from NAPT PFIB table resubmitted back to the FIB rather than straight to the external network subnet-id group. In the FIB table it should be matched against a new flow with lower priority than any other flow containing dst-ip match. Traffic will be redirected based on the vpn-id of the router gateway subnet to the external network subnet-id group.

```

Classifier table (0) =>
Dispatcher table (17) l3vpn service: set vpn-id=router-id=>
GW Mac table (19) match: vpn-id=router-id,dst-mac=router-interface-mac=>
FIB table (21) match: vpn-id=router-id=>
Pre SNAT table (26) match: vpn-id=router-id=>
Outbound NAPT table (46) match: src-ip=vm-ip,port=int-port set
src-ip=router-gw-ip,vpn-id=router-gw-subnet-id,port=ext-port=>
NAPT PFIB table (47) match: vpn-id=router-gw-subnet-id=>
FIB table (21) match: vpn-id=router-gw-subnet-id=>
Subnet-id group: set dst-mac=ext-subnet-gw-mac, reg6=provider-lport-tag=>
Egress table (220) output to provider network

```

### Ingress traffic from the internet to VM with no associated floating IP - NAPT switch

- For FIB table the vpn-id will be based on the subnet-id of the router gateway

```

Classifier table (0) =>
Dispatcher table (17) l3vpn service: set vpn-id=ext-net-id=>
GW Mac table (19) match: vpn-id=ext-net-id,dst-mac=router-gw mac=>
FIB table (21) match: vpn-id=ext-net-id,dst-ip=router-gw set
vpn-id=router-gw-subnet-id=>
Inbound NAPT table (44) match: dst-ip=router-gw,port=ext-port set
dst-ip=vm-ip,vpn-id=router-id,port=int-port=>
PFIB table (47) match: vpn-id=router-id=>
FIB table (21) match: vpn-id=router-id,dst-ip=vm-ip=>
Local Next-Hop group: set dst-mac=vm-mac, reg6=vm-lport-tag=>

```

Egress table (220) output to VM port

### Hairpinning - VM traffic to floating ip on the same compute node

- For Pre SNAT, SNAT, FIB tables the vpn-id will be based on the subnet-id of the floating ips

```
Classifier table (0) =>
Dispatcher table (17) l3vpn service: set vpn-id=router-id =>
GW Mac table (19) match: vpn-id=router-id, dst-mac=router-interface-mac =>
FIB table (21) match: vpn-id=router-id =>
Pre SNAT table (26) match: vpn-id=router-id, src-ip=src-vm-ip set
vpn-id=fip-subnet-id, src-ip=src-fip =>
SNAT table (28) match: vpn-id=fip-subnet-id, src-ip=src-fip set
src-mac=src-fip-mac =>
FIB table (21) match: vpn-id=fip-subnet-id, dst-ip=dst-fip =>
Pre DNAT table (25) match: dst-ip=dst-fip set
vpn-id=router-id, dst-ip=dst-vm-ip =>
DNAT table (27) match: vpn-id=router-id, dst-ip=dst-vm-ip =>
FIB table (21) match: vpn-id=router-id, dst-ip=dst-vm-ip =>
Local Next-Hop group: set dst-mac=dst-vm-mac, reg6=dst-vm-lport-tag =>
Egress table (220) output to VM port
```

### Hairpinning - VM traffic to floating ip on remote compute node

#### VM originating the traffic (Ingress DPN):

- For Pre SNAT, SNAT, FIB tables the vpn-id will be based on the subnet-id of the floating ip
- The destination mac is updated by the FIB table to be the floating ip mac. Traffic is sent to the egress DPN over the port of the flat/VLAN provider network.

```
Classifier table (0) =>
Dispatcher table (17) l3vpn service: set vpn-id=router-id =>
GW Mac table (19) match: vpn-id=router-id, dst-mac=router-interface-mac =>
FIB table (21) match: vpn-id=router-id =>
Pre SNAT table (26) match: vpn-id=router-id, src-ip=src-vm-ip set
vpn-id=fip-subnet-id, src-ip=src-fip =>
SNAT table (28) match: vpn-id=fip-subnet-id, src-ip=src-fip set
src-mac=src-fip-mac =>
FIB table (21) match: vpn-id=fip-subnet-id, dst-ip=dst-fip set
dst-mac=dst-fip-mac, reg6=provider-lport-tag =>
Egress table (220) output to provider network
```

**VM receiving the traffic (Egress DPN):**

- For GW MAC, FIB table the vpn-id will be based on the subnet-id of the floating ip

```

Classifier table (0) =>
Dispatcher table (17) l3vpn service:  set vpn-id=ext-net-id=>
GW Mac table (19) match:  vpn-id=ext-net-id,dst-mac=dst-fip-mac set
vpn-id=fip-subnet-id=>
FIB table (21) match:  vpn-id=fip-subnet-id,dst-ip=dst-fip=>
Pre DNAT table (25) match:  dst-ip=dst-fip set
vpn-id=router-id,dst-ip=dst-vm-ip=>
DNAT table (27) match:  vpn-id=router-id,dst-ip=dst-vm-ip=>
FIB table (21) match:  vpn-id=router-id,dst-ip=dst-vm-ip=>
Local Next-Hop group: set  dst-mac=dst-vm-mac,lport-tag=dst-vm-lport-tag=>
Egress table (220) output to VM port

```

**Hairpinning - traffic from VM with no associated floating IP to floating ip on remote compute node****VM originating the traffic (Ingress DPN) is non-NAPT switch:**

- No flow changes required. Traffic will be directed to NAPT switch and directed to the outbound NAPT table straight from the internal tunnel table

```

Classifier table (0) =>
Dispatcher table (17) l3vpn service:  set vpn-id=router-id=>
GW Mac table (19) match:  vpn-id=router-id,dst-mac=router-interface-mac=>
FIB table (21) match:  vpn-id=router-id=>
Pre SNAT table (26) match:  vpn-id=router-id=>
NAPT Group output to tunnel port of NAPT switch=>

```

**VM originating the traffic (Ingress DPN) is the NAPT switch:**

- For Outbound NAPT, NAPT PFIB, Pre DNAT, DNAT and FIB tables the vpn-id will be based on the common subnet-id of the router gateway and the floating-ip.
- Packets from NAPT PFIB table resubmitted back to the FIB where they will be matched against the destination floating ip.
- The destination mac is updated by the FIB table to be the floating ip mac. Traffic is sent to the egress DPN over the port of the flat/VLAN provider network.

```

Classifier table (0) =>
Dispatcher table (17) l3vpn service:  set vpn-id=router-id=>
GW Mac table (19) match:  vpn-id=router-id,dst-mac=router-interface-mac=>
FIB table (21) match:  vpn-id=router-id=>
Pre SNAT table (26) match:  vpn-id=router-id=>

```

Outbound NAPT table (46) match: src-ip=vm-ip,port=int-port set  
src-ip=router-gw-ip,vpn-id=router-gw-subnet-id,port=ext-port =>  
NAPT PFIB table (47) match: vpn-id=router-gw-subnet-id=>  
FIB table (21) match: vpn-id=router-gw-subnet-id dst-ip=dst-fip set  
dst-mac=dst-fip-mac, reg6=provider-lport-tag =>  
Egress table (220) output to provider network

### VM receiving the traffic (Egress DPN):

- For GW MAC, FIB table the vpn-id will be based on the subnet-id of the floating ip

Classifier table (0) =>  
Dispatcher table (17) l3vpn service: set vpn-id=ext-net-id=>  
GW Mac table (19) match: vpn-id=ext-net-id,dst-mac=dst-fip-mac set  
vpn-id=fip-subnet-id=>  
FIB table (21) match: vpn-id=fip-subnet-id,dst-ip=dst-fip=>  
Pre DNAT table (25) match: dst-ip=dst-fip set  
vpn-id=router-id,dst-ip=dst-vm-ip=>  
DNAT table (27) match: vpn-id=router-id,dst-ip=dst-vm-ip=>  
FIB table (21) match: vpn-id=router-id,dst-ip=dst-vm-ip=>  
Local Next-Hop group: set dst-mac=dst-vm-mac,lport-tag=dst-vm-lport-tag=>  
Egress table (220) output to VM port

### Yang changes

odl-nat module will be enhanced with the following container

```
container external-subnets {  
  list subnets {  
    key id;  
    leaf id {  
      type yang:uuid;  
    }  
    leaf vpnid {  
      type yang:uuid;  
    }  
    leaf-list router-ids {  
      type yang:uuid;  
    }  
    leaf external-network-id {  
      type yang:uuid;  
    }  
  }  
}
```

This model will be filled out only for flat/VLAN external network provider types. If this model is missing, vpn-id will be taken from odl-nat:external-networks model to maintain compatibility with BGPVPN models.

odl-nat:ext-routers container will be enhanced with the list of the external subnet-ids associated with the router.



```
container ext-routers {  
  list routers {  
    key router-name;  
    leaf router-name {  
      type string;  
    }  
    ...  
  
    leaf-list external-subnet-id {  
      type yang:uuid; }  
    }  
  }  
}
```

**Configuration impact**

None

**Clustering considerations**

None

**Other Infra considerations**

None

**Security considerations**

None

**Scale and Performance Impact**

None

**Targeted Release**

Carbon

**Alternatives**

None

## Usage

### Create external network with two subnets

```
neutron net-create public-net -- --router:external --is-default --provider:network_
↪type=flat
--provider:physical_network=physnet1
neutron subnet-create --ip_version 4 --gateway 10.64.0.1 --name public-subnet1
↪<public-net-uuid> 10.64.0.0/16
-- --enable_dhcp=False
neutron subnet-create --ip_version 4 --gateway 10.65.0.1 --name public-subnet2
↪<public-net-uuid> 10.65.0.0/16
-- --enable_dhcp=False
```

### Create internal networks with subnets

```
neutron net-create private-net1
neutron subnet-create --ip_version 4 --gateway 10.0.123.1 --name private-subnet1
↪<private-net1-uuid>
10.0.123.0/24
neutron net-create private-net2
neutron subnet-create --ip_version 4 --gateway 10.0.124.1 --name private-subnet2
↪<private-net2-uuid>
10.0.124.0/24
neutron net-create private-net3
neutron subnet-create --ip_version 4 --gateway 10.0.125.1 --name private-subnet3
↪<private-net3-uuid>
10.0.125.0/24
neutron net-create private-net4
neutron subnet-create --ip_version 4 --gateway 10.0.126.1 --name private-subnet4
↪<private-net4-uuid>
10.0.126.0/24
```

### Create two router instances and connect each router to one internal subnet and one external subnet

```
neutron router-create router1
neutron router-interface-add <router1-uuid> <private-subnet1-uuid>
neutron router-gateway-set --fixed-ip subnet_id=<public-subnet1-uuid> <router1-uuid>
↪<public-net-uuid>
neutron router-create router2
neutron router-interface-add <router2-uuid> <private-subnet2-uuid>
neutron router-gateway-set --fixed-ip subnet_id=<public-subnet2-uuid> <router2-uuid>
↪<public-net-uuid>
```

### Create router instance connected to both external subnets and the remaining internal subnets

```
neutron router-create router3
neutron router-interface-add <router3-uuid> <private-subnet3-uuid>
neutron router-interface-add <router3-uuid> <private-subnet4-uuid>
```

```
neutron router-gateway-set --fixed-ip subnet_id=<public-subnet1-uuid> --fixed-ip_
↪ subnet_id=<public-subnet2-uuid>
<router3-uuid> <public-net-uuid>
```

### Create floating ips from both subnets

```
neutron floatingip-create --subnet <public-subnet1-uuid> public-net
neutron floatingip-create --subnet <public-subnet1-uuid> public-net
neutron floatingip-create --subnet <public-subnet2-uuid> public-net
```

### Create 2 VM instance in each subnet and associate with floating ips

```
nova boot --image <image-id> --flavor <flavor-id> --nic net-id=<private-net1-uuid> VM1
nova floating-ip-associate VM1 <fip1-public-subnet1>
nova boot --image <image-id> --flavor <flavor-id> --nic net-id=<private-net1-uuid> VM2
nova floating-ip-associate VM2 <fip2-public-subnet1>
nova boot --image <image-id> --flavor <flavor-id> --nic net-id=<private-net2-uuid> VM3
nova floating-ip-associate VM3 <fip1-public-subnet2>
nova boot --image <image-id> --flavor <flavor-id> --nic net-id=<private-net2-uuid> VM4
nova boot --image <image-id> --flavor <flavor-id> --nic net-id=<private-net3-uuid> VM5
nova boot --image <image-id> --flavor <flavor-id> --nic net-id=<private-net4-uuid> VM6
```

### Connectivity tests

- Connect to the internet from all VMs. VM1 and VM2 will route traffic through external gateway 10.64.0.1 VM3 and VM4 route traffic through external gateway 10.65.0.1.
- Connect to the internet from VM5 and VM6. Each connection will be routed to different external gateway with the corresponding subnet router-gateway ip.
- Hairpinning when source VM is associated with floating ip - ping between VM1 and VM2 using their floating ips.
- Hairpinning when source VM is not associated with floating ip - ping from VM4 to VM3 using floating ip. Since VM4 has no associated floating ip a NAT entry will be allocated using the router-gateway ip.

### Features to Install

odl-netvirt-openstack

### REST API

N/A

### CLI

N/A

## Implementation

### Assignee(s)

**Primary assignee:** Yair Zinger <yair.zinger@hpe.com>

**Other contributors:** Tali Ben-Meir <tali@hpe.com>

### Work Items

<https://trello.com/c/uDcQw95v/104-pipeline-changes-fip-w-multiple-subnets-in-ext-net-hairpinning>

- Add external-subnets model
- Add vpn-instances for external flat/VLAN subnets
- Change pipeline to prefer vpn-id from external-subnets over vpn-id from external-networks
- Add write metadata to GW MAC table for floating ip/router gw mac addresses
- Add default subnet-id match in FIB table to external subnet group entry
- **Changes in remote next-hop flow for floating ip in FIB table**
  - Set destination mac to floating ip mac
  - Set egress actions to provider port of the network attached to the floating ip subnet
- Resubmit SNAT + Outbound NAT flows to FIB table

### Dependencies

None

### Testing

#### Unit Tests

#### Integration Tests

#### CSIT

- Hairpinning between VMs in the same subnet
- Hairpinning between VMs in different subnets connected to the same router
- Hairpinning with NAT - source VM is not associated with floating ip
- Traffic to external network with multiple subnets

### Documentation Impact

None

## References

[1] OpenDaylight Documentation Guide

### Table of Contents

- *IPv6 DC-Internet L3 North-South connectivity using L3VPN provider network types.*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Pipeline changes*
    - \* *Fib Manager changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*
  - *Usage*
    - \* *Features to Install*
    - \* *REST API*
    - \* *CLI*
  - *Implementation*
    - \* *Assignee(s)*
    - \* *Work Items*
  - *Dependencies*
  - *Testing*
    - \* *Unit Tests*
    - \* *Integration Tests*
    - \* *CSIT*
  - *Documentation Impact*
  - *References*

### 1.1.12 IPv6 DC-Internet L3 North-South connectivity using L3VPN provider network types.

<https://git.opendaylight.org/gerrit/#/q/topic:ipv6-l3vpn-internet>

In this specification we will be discussing the high level design of IPv6 Datacenter to Internet North-South connectivity support in OpenDaylight using L3VPN provider network type use-case.

#### Problem description

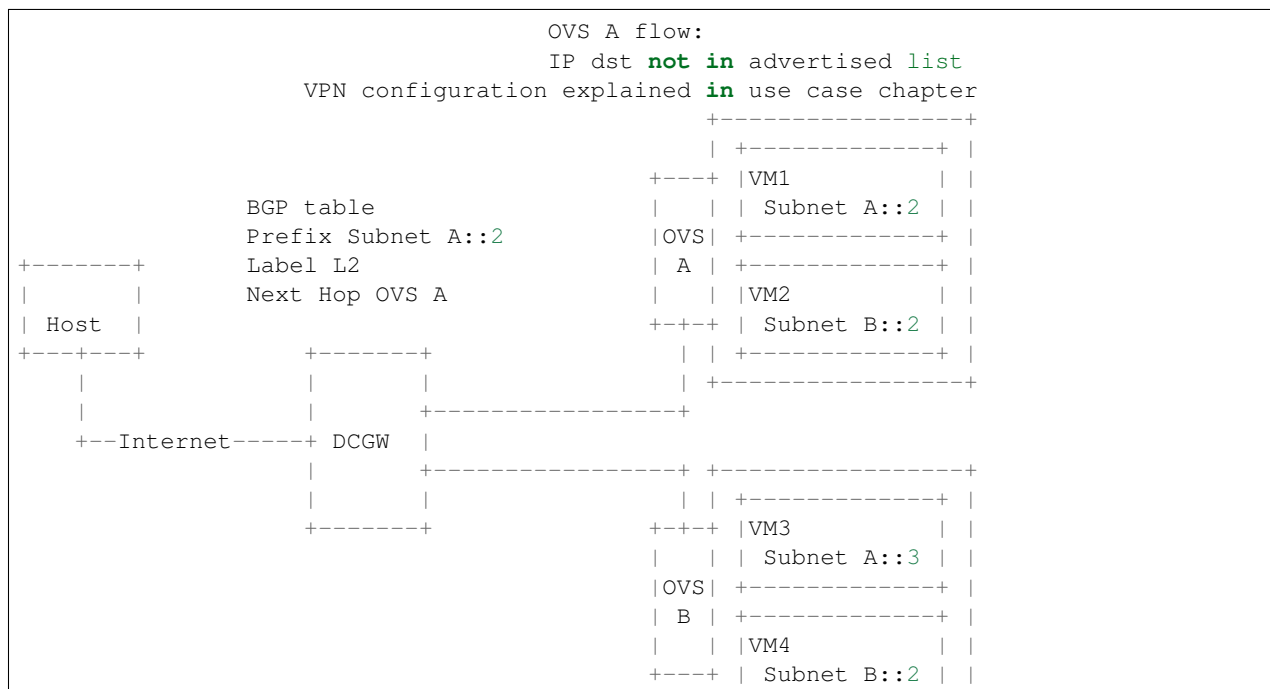
Provide IPv6 connectivity to virtual machines located in different subnets spread over multiple sites or Data center can be achieved through use of Globally Unique Addresses and capacity to update enough routing tables to forge a path between the two. Even if IPv6 is made to interconnect hosts without the help of any NAT mechanisms, routing with the best efficiency (shortest path) or policy (route weight, commercial relationships) must be configured using only few parameters, automatically updating routes for each VM spawned in new network.

Keep in mind that key aspects of L3VPN connectivity is Route Targets and VPN-IPv6 address family. Assuming an operator can configure data center gateways with a Route Distinguisher dedicated to Internet connectivity and a set of imported Route Targets, each time a virtual machine is spawned within a data center subnet associated with that Route Distinguisher, it will trigger the send of a BGP UPDATE message containing MP-BGP attributes required for reaching the VM outside the datacenter. In the same manner, adding extra-route or declaring subnetworks will trigger the same. Such behavior can be achieved by configuring a neutron router an internet public VPN address. For the following of the document, we focus to GUA/128 addresses that are advertised, when one VM start. Indeed, most of the requirements are dealing with VM access to internet.

Only IPv6 Globally Unique Address (eg /128) are advertised, this is not a scaling architecture since it implies as much routes to process as the number of spawned VMs, but with such BGP routing information base, DCGW can select the Compute Node to which a packet coming from the WAN should be forwarded to.

The following covers the case where a VM connects to a host located in the internet, and the destination ip address of packets is not part of the list of advertised prefixes (see spec [6]).

Following schema could help :



```

| +-----+ |
+-----+

```

## Use Cases

Datacenter IPv6 external connectivity to/from Internet for VMs spawned on tenant networks.

There are several techniques for VPNs to access the Internet. Those methods are described in [8], on section 11. Also a note describes in [8] the different techniques that could be applied to the DC-GW case. Note that not all solutions are compliant with the RFC. Also, we make the hypothesis of using GUA.

The method that will be described more in detail below is the option 2. Option 2 is external network connectivity option 2 from [8]). That method implies 2 VPNs. One VPN will be dedicated to Internet access, and will contain the Internet Routes, but also the VPNs routes. The Internet VPN can also contain default route to a gateway. Having a separated VPN brings some advantages: - the VPN that do not need to get Internet access get the private characteristic of VPNs.

- using a VPN internet, instead of default forwarding table is enabling flexibility, since it could permit creating more than one internet VPN. As consequence, it could permit applying different rules ( different gateway for example).

Having 2 VPNs implies the following for one packet going from VPN to the internet. The FIB table will be used for that. If the packet's destination address does not match any route in the first VPN, then it may be matched against the internet VPN forwarding table. Reversely, in order for traffic to flow natively in the opposite direction, some of the routes from the VPN will be exported to the internet VPN.

Configuration steps in a datacenter:

- Configure ODL and Devstack networking-odl for BGP VPN.
- Create a tenant network with IPv6 subnet using GUA prefix or an

admin-created-shared-ipv6-subnet-pool. - This tenant network is connected to an external network where the DCGW is

connected. Separation between both networks is done by DPN located on compute nodes. The subnet on this external network is using the same tenant as an IPv4 subnet used for MPLS over GRE tunnels endpoints between DCGW and DPN on Compute nodes. Configure one GRE tunnel between DPN on compute node and DCGW.

- Create a Neutron Router and connect its ports to all internal subnets
- Create a transport zone to declare that a tunneling method is planned to reach an external IP:

the IPv6 interface of the DC-GW

- The neutron router subnetworks will be associated to two L3 BGPVPN instance.

The step create the L3VPN instances and associate the instances to the router. Especially, two VPN instances will be created, one for the VPN, and one for the internetVPN.

```

operations:neutronvpn:createL3VPN ( "route-distinguisher" = "vpn1" "import-RT" =
["vpn1","internetvpn"] "export-RT" = ["vpn1","internetvpn"])

```

```

operations:neutronvpn:createL3VPN ( "route-distinguisher" = "internetvpn" "import-
RT" = "internetvpn" "export-RT" = "internetvpn")

```

- The DC-GW configuration will also include 2 BGP VPN instances. Below is a configuration from QBGW using vty command interface.

```
vrf rd "internetvpn" vrf rt both "internetvpn" vrf rd "vpn1" vrf rt both "vpn1" "internetvpn"
```

- Spawn VM and bind its network interface to a subnet, L3 connectivity between

VM in datacenter and a host on WAN must be successful. More precisely, a route belonging to VPN1 will be associated to VM GUA, and will be sent to remote DC-GW. DC-GW will import the entry to both "vpn1" and "internetvpn" so that the route will be known on both vpns. Reversely, because DC-GW knows internet routes in "internetvpn", those routes will be sent to QBGP. ODL will get those internet routes, only in the "internetvpn" vpn. For example, when a VM will try to reach a remote, a first lookup will be done in "vpn1" FIB table. If none is found, a second lookup will be found in the "internetvpn" FIB table. The second lookup should be successful, thus triggering the encapsulation of packet to the DC-GW.

**When the data centers is set up, there are 2 use cases:**

- Traffic from Local DPN to DC-Gateway
- Traffic from DC-Gateway to Local DPN

The use cases are slightly different from [6], on the Tx side.

## Proposed change

Similar as with [6], plus a specific processing on Tx side. An additionnal processing in DPN is required. When a packet is received by a neutron router associated with L3VPN, with destination mac address is the subnet gateway mac address, and the destination ip is not in the FIB (default gateway) of local DPN, then the packet should do a second lookup in the second VPN configured. So that the packet can enter the L3VPN netvirt pipeline. The MPLS label pushed on the IPv6 packet is the one configured to provide access to Internet at DCGW level.

## Pipeline changes

No pipeline changes, compared with [6]. However, FIB Manager will be modified so as to implement the fallback mechanism. The FIB tables of the import-RTs VPNs from the default VPN created will be parsed. In our case, a match will be found in the "internetVPN" FIB table. If not match is found, the drop rule will be applied.

Regarding the pipeline changes, we can use the same BGPVPNv4 pipeline (Tables Dispatcher (17), DMAC (19), LFIB (20), L3FIB (21), and NextHop Group tables) and enhance those tables to support IPv6 North-South communication through MPLS/GRE. For understanding, the pipeline is written below: l3vpn-id is the ID associated to the initial VPN, while l3vpn-internet-id is the ID associated to the internet VPN.

## Traffic from DC-Gateway to Local DPN (SYMMETRIC IRB)

When a packet is coming from DC-Gateway, the label will help finding out the associated VPN. The first one is l3vpn-id.

Classifier Table (0) =>

LFIB Table (20) match: tun-id=mpls\_label set vpn-id=l3vpn-id, pop\_mpls label, set output to nexthopgroup-dst-vm=>

NextHopGroup-dst-vm: set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag=>

Lport Egress Table (220) Output to dst vm port



## Traffic from Local DPN to DC-Gateway (SYMMETRIC IRB)

When a packet is going out from a dedicated VM, the l3vpn-id attached to that subnetwork will be used. Theoretically, in L3 FIB, there will be no match for dst IP with this l3vpn-id. However, because ODL know the relationship between both VPNs, then the dst IP will be attached with the first l3vpn-id.

However, since the gateway IP for inter-DC and external access is the same, the same MPLS label will be used for both VPNs.

Classifier Table (0) =>

Lport Dispatcher Table (17) “match: LportTag l3vpn service: set vpn-id=l3vpn-id” =>

DMAC Service Filter (19) match: dst-mac=router-internal-interface-mac l3vpn service: set vpn-id=internet-l3vpn-id =>

L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=<alternate-ip> set tun-id=mpls\_label output to MPLSoGRE tunnel port =>

L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=ext-ip-address set tun-id=mpls\_label output to MPLSoGRE tunnel port =>

## Fib Manager changes

### Ingress traffic from internet to VM

The FIB Manager is being configured with 2 entries for different RDs : l3vpn-id and internetvpn-id. The LFIB will be matched first. In our case, label NH and prefix are the same, whereas we have 2 VPN instances. So, proposed change is to prevent LFIB from adding entries if a label is already registered for that compute node.

### Egress traffic from VM to internet

The FIB Manager is being configured with the internet routes on one RD only : internetvpn-id. As packets that are emitted from the VM with vpn=l3vpn-id, the internet route will not be matched in l3vpn, if implementation remains as it is. In FIB Manager, solution is the following: - The internetvpn is not attached to any local subnetwork. so, any eligible VPNs are looked up in the list of VPN instances. for each VPN instance, for each RD, if an imported RT matches the internetvpnID, then a new rule will be appended.

## Yang changes

None

## Configuration impact

The configuration will require to create 2 VPN instances.

## Clustering considerations

None

## Other Infra considerations

None

## Security considerations

None

## Scale and Performance Impact

The number of entries will be duplicated, compared with [6]. This is the cost in order to keep some VPNs private, and others kind of public. Another impact is the double lookup that may result, when emitting a packet. This is due to the fact that the whole fib should be parsed to fallback to the next VPN, in order to make an other search, so that the packet can enter in the L3VPN flow.

## Targeted Release

Carbon

## Alternatives

None

## Usage

- Configure MPLS/GRE tunnel endpoint on DCGW connected to public-net network
- Configure neutron networking-odl plugin
- Configure BGP speaker in charge of retrieving prefixes for/from data center gateway in ODL through the set of `vpnservice.bgpspeaker.host.name` in `etc/custom.properties`. No REST API can configure that parameter. Use `config/ebgp:bgp` REST api to start BGP stack and configure VRF, address family and neighboring. In our case, as example, following values will be used:
  - `rd="100:2" # internet VPN - import-rt="100:2" - export-rt="100:2"`
  - `rd="100:1" # vpn1 - import-rt="100:1 100:2" - export-rt="100:1 100:2"`

```
POST config/ebgp:bgp
{
  "ebgp:as-id": {
    "ebgp:stalepath-time": "360",
    "ebgp:router-id": "<ip-bgp-stack>",
    "ebgp:announce-fbit": "true",
    "ebgp:local-as": "<as>"
  },
  "ebgp:neighbors": [
    {
      "ebgp:remote-as": "<as>",
      "ebgp:address-families": [
        {
          "ebgp:afi": "2",
```

```

        "ebgp:peer-ip": "<neighbor-ip-address>",
        "ebgp:safi": "128"
    },
    ],
    "ebgp:address": "<neighbor-ip-address>"
}
],
}

```

\* Configure BGP speaker on DCGW to exchange prefixes **with** ODL BGP stack. Since DCGW should be a vendor solution, the configuration of such equipment **is** out of the scope of this specification.

- Create an internal tenant network with an IPv6 (or dual-stack) subnet.

```

neutron net-create private-net
neutron subnet-create --name ipv6-int-subnet --ip-version 6
--ipv6-ra-mode slaac --ipv6-address-mode slaac private-net 2001:db8:0:2::/64

```

- Use neutronvpn:createL3VPN REST api to create L3VPN

POST /restconf/operations/neutronvpn:createL3VPN

```

{
  "input": {
    "l3vpn": [
      {
        "id": "vpnid_uuid_1",
        "name": "internetvpn",
        "route-distinguisher": [100:2],
        "export-RT": [100:2],
        "import-RT": [100:2],
        "tenant-id": "tenant_uuid"
      }
    ]
  }
}

```

POST /restconf/operations/neutronvpn:createL3VPN

```

{
  "input": {
    "l3vpn": [
      {
        "id": "vpnid_uuid_2",
        "name": "vpn1",
        "route-distinguisher": [100:1],
        "export-RT": [100:1, 100:2],
        "import-RT": [100:1, 100:2],
        "tenant-id": "tenant_uuid"
      }
    ]
  }
}

```

- Associate L3VPN To Network

```
POST /restconf/operations/neutronvpn:associateNetworks
```

```
{
  "input": {
    "vpn-id": "vpnid_uuid_1",
    "network-id": "network_uuid"
  }
}
```

- Spawn a VM in the tenant network

```
nova boot --image <image-id> --flavor <flavor-id> --nic net-id=<private-net> VM1
```

- Dump ODL BGP FIB

```
GET /restconf/config/odl-fib:fibEntries
```

```
{
  "fibEntries": {
    "vrfTables": [
      {
        "routeDistinguisher": <rd-uuid_1>
      },
      {
        "routeDistinguisher": <rd_vpn1>,
        "vrfEntry": [
          {
            "destPrefix": <IPv6_VM1/128>,
            "label": <label>,
            "nextHopAddressList": [
              <DPN_IPv4>
            ],
            "origin": "1"
          }
        ],
      },
    ]
  },
  {
    "routeDistinguisher": <rd-uuid_2>
  },
  {
    "routeDistinguisher": <rd_vpninternet>,
    "vrfEntry": [
      {
        "destPrefix": <IPv6_VM1/128>,
        "label": <label>,
        "nextHopAddressList": [
          <DPN_IPv4>
        ],
        "origin": "1"
      }
    ],
  },
]
}
```

## Features to Install

odl-netvirt-openstack

## REST API

## CLI

## Implementation

## Assignee(s)

**Primary assignee:** Julien Courtat <julien.courtat@6wind.com>

**Other contributors:** Noel de Prandieres <prandieres@6wind.com> Valentina Krasnobaeva <valentina.krasnobaeva@6wind.com> Philippe Guibert <philippe.guibert@6wind.com>

## Work Items

- Validate proposed setup so that each VM entry is duplicated in 2 VPN instances
- Implement FIB-Manager fallback mechanism for output packets

## Dependencies

[6]

## Testing

## Unit Tests

Unit tests related to fallback mechanism when setting up 2 VPN instances configured as above.

## Integration Tests

TBD

## CSIT

CSIT provided for the BGPVPNv6 versions will be enhanced to also support connectivity to Internet.

## Documentation Impact

Necessary documentation would be added on how to use this feature.

## References

- [1] OpenDaylight Documentation Guide
- [2] <https://specs.openstack.org/openstack/nova-specs/specs/kilo/template.html>
- [3] <http://docs.openstack.org/developer/networking-bgpvpn/overview.html>
- [4] IPv6 Distributed Router for Flat/VLAN based Provider Networks.
- [5] BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN
- [6] Spec to support IPv6 Inter DC L3VPN connectivity using BGPVPN.
- [7] Spec to support IPv6 North-South support for Flat/VLAN Provider Network.
- [8] External Network connectivity in IPv6 networks.
- [9] BGP/MPLS IP Virtual Private Networks (VPNs)

### Table of Contents

- *IPv6 Inter-DC L3 North-South connectivity using L3VPN provider network types.*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*
  - *Usage*
    - \* *Features to Install*
    - \* *REST API*
    - \* *CLI*
  - *Implementation*
    - \* *Assignee(s)*
    - \* *Work Items*
  - *Dependencies*
  - *Testing*
    - \* *Unit Tests*

- \* *Integration Tests*
- \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.13 IPv6 Inter-DC L3 North-South connectivity using L3VPN provider network types.

<https://git.opendaylight.org/gerrit/#/q/topic:ipv6-interdc-l3vpn>

In this specification we will be discussing the high level design of IPv6 Inter-Datacenter North-South connectivity support in OpenDaylight using L3VPN provider network type use-case.

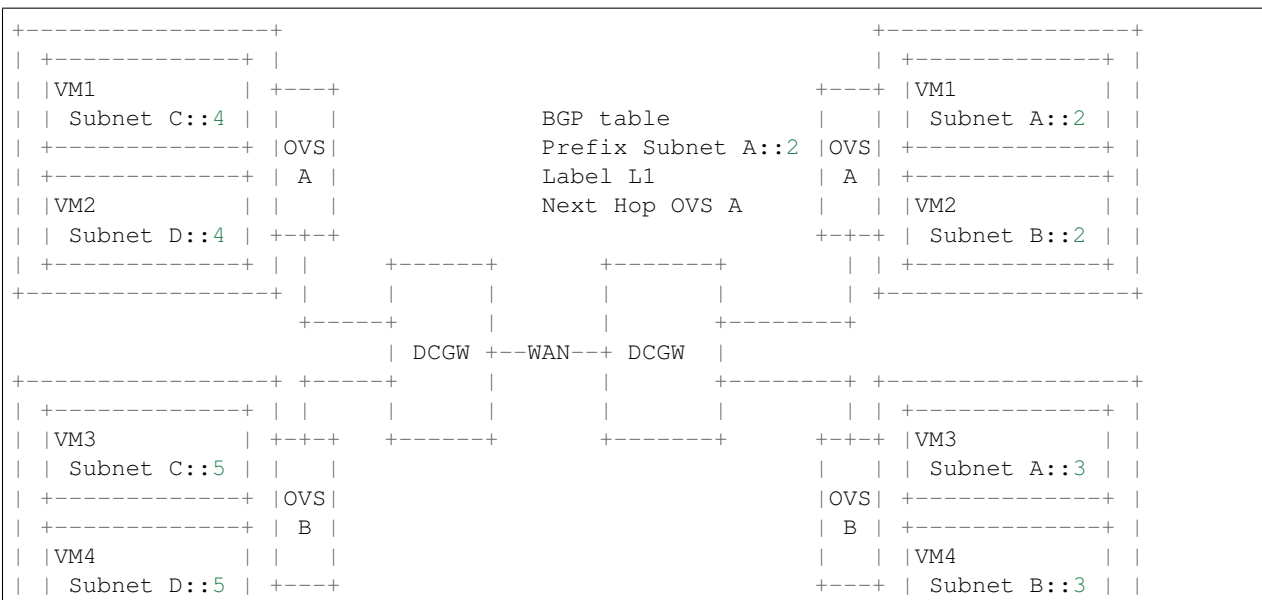
#### Problem description

Provide IPv6 connectivity to virtual machines located in different subnets spread over multiple sites or Data center can be achieved through use of Globally Unique Addresses and capacity to update enough routing tables to forge a path between the two. Even if IPv6 is made to interconnect hosts without the help of any NAT mechanisms, routing with the best efficiency (shortest path) or policy (route weight, commercial relationships) must be configured using only few parameters, automatically updating routes for each VM spawned in new network.

Keep in mind that key aspects of L3VPN connectivity is Route Targets and VPN-IPv6 address family. Assuming an operator can configure both data center gateways with same Route Distinguisher or set of imported Route Targets, each time a virtual machine is spawned within a new subnet, it will trigger the send of a BGP UPDATE message containing MP-BGP attributes required for reaching the VM. Such behavior can be achieved by configuring a neutron router a default gateway.

Only IPv6 Globally Unique Address (eg /128) are advertised, this is not a scaling architecture since it implies as much routes to process as the number of spawned VMs, but with such BGP routing information base, DCGW can select the Compute Node to which a packet coming from the WAN should be forwarded to.

Following schema could help :





BGP protocol and its MP-BGP extension would do the job as long as all BGP speakers are capable of processing UPDATE messages containing VPN-IPv6 address family, which AFI value is 2 and SAFI is 128. It is not required that BGP speakers peers using IPv6 LLA or GUA, IPv4 will be used to peer speakers together.

Opendaylight is already able to support the VPN-IPv4 address family (AFI=1, SAFI=128), and this blueprint focuses on specific requirements to VPN-IPv6.

One big question concerns the underlying transport IP version used with MPLS/GRE tunnels established between Data center Gateway (DCGW), and compute nodes (CNs). There is one MPLS/GRE tunnel setup from DCGW to each Compute Node involved in the L3VPN topology. Please note that this spec doesn't covers the case of VxLAN tunnels between DCGW and Compute Nodes.

According to RFC 4659 §3.2.1, the encoding of the nexthop attribute in MP-BGP UPDATE message differs if the tunneling transport version required is IPv4 or IPv6. In this blueprint spec, the assumption of transport IP version of IPv4 is preferred. This implies that any nexthop set for a prefix in FIB will be IPv4.

Within BGP RIB table, for each L3VPN entry, the nexthop and label are key elements for creating MPLS/GRE tunnel endpoints, and the prefix is used for programming netvirt pipeline. When a VM is spawned, the prefix advertised by BGP is 128 bits long and the nexthop carried along within UPDATE message is the ip address of the DPN interface used for DCGW connection. Since DCGW can be proprietary device, it may not support MPLS/GRE tunnel endpoint setup according to its internal BGP table. A static configuration of such tunnel endpoint may be required.

## Use Cases

Inter Datacenter IPv6 external connectivity for VMs spawned on tenant networks, routes exchanged between BGP speakers using same Route Distinguisher.

Steps in both data centers :

- Configure ODL and Devstack networking-odl for BGP VPN.
- Create a tenant network with IPv6 subnet using GUA prefix or an admin-created-shared-ipv6-subnet-pool.
- This tenant network is separated to an external network where the DCGW is connected. Separation between both networks is done by DPN located on compute nodes. The subnet on this external network is using the same tenant as an IPv4 subnet used for MPLS over GRE tunnels endpoints between DCGW and DPN on Compute nodes. Configure one GRE tunnel between DPN on compute node and DCGW.
- Create a Neutron Router and connect its ports to all internal subnets that will belong to the same L3 BGPVPN identified by a Route Distinguisher.
- Start BGP stack managed by ODL, possibly on same host as ODL.
- Create L3VPN instance.
- Associate the Router with the L3VPN instance.
- Spawn VM on the tenant network, L3 connectivity between VMs located on different datacenter sharing same Route Distinguisher must be successful.

When both data centers are set up, there are 2 use cases per data center:

- Traffic from DC-Gateway to Local DPN (VMS on compute node)
- Traffic from Local DPN to DC-Gateway



## Proposed change

ODL Controller would program the necessary pipeline flows to support IPv6 North South communication through MPLS/GRE tunnels out of compute node.

BGP manager would be updated to process BGP RIB when entries are IPv6 prefixes.

FIB manager would be updated to take into account IPv6 prefixes.

Thrift interface between ODL and BGP implementation (Quagga BGP) must be enhanced to support new AFI=2. Thrift interface will still carry IPv4 Nexthops, and it will be the Quagga duty to transform this IPv4 Nexthop address into an IPv4-mapped IPv6 address in every NLRI fields. Here is the new api proposed :

```
enum af_afi {
    AFI_IP = 1,
    AFI_IPV6 = 2,
}
i32 pushRoute(1:string prefix, 2:string nexthop, 3:string rd, 4:i32 label,
              5:af_afi afi)
i32 withdrawRoute(1:string prefix, 2:string rd, 3:af_afi afi)
oneway void onUpdatePushRoute(1:string rd, 2:string prefix,
                              3:i32 prefixlen, 4:string nexthop,
                              5:i32 label, 6:af_afi afi)
oneway void onUpdateWithdrawRoute(1:string rd, 2:string prefix,
                                  3:i32 prefixlen, 4:string nexthop,
                                  5:af_afi afi)
Routes getRoutes(1:i32 optype, 2:i32 winSize, 3:af_afi afi)
```

BGP implementation (Quagga BGP) announcing (AFI=2,SAFI=128) capability as well as processing UPDATE messages with such address family. Note that the required changes in Quagga is not part of the design task covered by this blueprint.

## Pipeline changes

Regarding the pipeline changes, we can use the same BGPVPNv4 pipeline (Tables Dispatcher (17), DMAC (19), LFIB (20), L3FIB (21), and NextHop Group tables) and enhance those tables to support IPv6 North-South communication through MPLS/GRE.

### Traffic from DC-Gateway to Local DPN (SYMMETRIC IRB)

Classifier Table (0) =>

LFIB Table (20) match: tun-id=mpls\_label set vpn-id=l3vpn-id, pop\_mpls label, set output to nexthopgroup-dst-vm =>

NextHopGroup-dst-vm: set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag =>

Lport Egress Table (220) Output to dst vm port

Please note that vpn-subnet-gateway-mac-address stands for MAC address of the neutron port of the internal subnet gateway router.

### Traffic from Local DPN to DC-Gateway (SYMMETRIC IRB)

Classifier Table (0) =>

Lport Dispatcher Table (17) match: LportTag l3vpn service: set vpn-id=l3vpn-id=>  
DMAC Service Filter (19) match: dst-mac=router-internal-interface-mac l3vpn service:  
set vpn-id=l3vpn-id=>  
L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=ext-ip-address set  
tun-id=mpls\_label output to MPLSoGRE tunnel port =>

Please note that router-internal-interface-mac stands for MAC address of the neutron port of the internal subnet gateway router.

## Yang changes

Changes will be needed in `ebgp.yang` to start supporting IPv6 networks advertisements.

A new leaf `afi` will be added to container `networks`

Listing 1.11: `ebgp.yang`

```
list networks {
  key "rd prefix-len";

  leaf rd {
    type string;
  }

  leaf prefix-len {
    type string;
  }

  leaf afi {
    type uint32;
    mandatory "false";
  }

  leaf nexthop {
    type inet:ipv4-address;
    mandatory "false";
  }

  leaf label {
    type uint32;
    mandatory "false";
  }
}
```

## Configuration impact

None

## Clustering considerations

None

## Other Infra considerations

None

## Security considerations

None

## Scale and Performance Impact

Impact on scaling inside datacenter essentially grow with the number of VM connected to subnets associated with the L3VPN. Since Globally Unique Address are used and there is no NAT involved in the datapath, it implies prefixes advertised are all /128. At the end, it means that every prefix advertised will have its entry in BGP RIB of all ODL controllers and DCGW involved in L3VPN (ie all bgp aware equipment will handle all prefixes advertised within a Route Distinguisher).

This may imply BGP table with very high number of entries. This also implies a high number of entries in ODL routing table and equivalent number of flows inserted in OVS, since prefix advertised add matching ip destination in OVS tables.

This fact also impact the scaling of the BGP speaker implementation (Quagga BGP) with many thousands of BG-PVPNv4 and BGPVPNv6 prefixes (as much as number of spawned VMs) with best path selection algorithm on route updates, graceful restart procedure, and multipath.

## Targeted Release

Carbon

## Alternatives

None

## Usage

- Configure MPLS/GRE tunnel endpoint on DCGW connected to public-net network
- Configure neutron networking-odl plugin
- Configure BGP speaker in charge of retrieving prefixes for/from data center gateway in ODL through the set of `vpnservice.bgpspeaker.host.name` in `etc/custom.properties`. No REST API can configure that parameter. Use `config/ebgp:bgp` REST api to start BGP stack and configure VRF, address family and neighboring

```
POST config/ebgp:bgp
{
  "ebgp:as-id": {
    "ebgp:stalepath-time": "360",
    "ebgp:router-id": "<ip-bgp-stack>",
    "ebgp:announce-fbit": "true",
    "ebgp:local-as": "<as>"
  },
  "ebgp:vrf": [
    {
```

```
    "ebgp:export-rt": [
      "<export-rt>"
    ],
    "ebgp:rd": "<RD>",
    "ebgp:import-rt": [
      "<import-rt>"
    ]
  },
],
"ebgp:neighbors": [
  {
    "ebgp:remote-as": "<as>",
    "ebgp:address-families": [
      {
        "ebgp:afi": "2",
        "ebgp:peer-ip": "<neighbor-ip-address>",
        "ebgp:safi": "128"
      }
    ],
    "ebgp:address": "<neighbor-ip-address>"
  }
],
]
```

- Configure BGP speaker on DCGW to exchange prefixes with ODL BGP stack. Since DCGW should be a vendor solution, the configuration of such equipment is out of the scope of this specification.
- Create an internal tenant network with an IPv6 (or dual-stack) subnet and connect ports.

```
neutron net-create private-net
neutron subnet-create private-net 2001:db8:0:2::/64 --name ipv6-int-subnet
--ip-version 6 --ipv6-ra-mode slaac --ipv6-address-mode slaac
neutron port-create private-net --name port1_private1
```

- Create a router and associate it to internal subnets.

```
neutron router-create router1
neutron router-interface-add router1 ipv6-int-subnet
```

- Use neutronvpn:createL3VPN REST api to create L3VPN

```
POST /restconf/operations/neutronvpn:createL3VPN
{
  "input": {
    "l3vpn": [
      {
        "id": "vpnid_uuid",
        "name": "vpn1",
        "route-distinguisher": [100:1],
        "export-RT": [100:1],
        "import-RT": [100:1],
        "tenant-id": "tenant_uuid"
      }
    ]
  }
}
```

- Associate L3VPN To Routers

```
POST /restconf/operations/neutronvpn:associateRouter
{
  "input":{
    "vpn-id":"vpn_id_uuid",
    "router-id":[ "router_uuid" ]
  }
}
```

- Create MPLSoGRE tunnel between DPN and DCGW

```
POST /restconf/operations/itm-rpc:add-external-tunnel-endpoint
{
  "itm-rpc:input": {
    "itm-rpc:destination-ip": "dcgw_ip",
    "itm-rpc:tunnel-type": "odl-interface:tunnel-type-mpls-over-gre"
  }
}
```

- Spawn a VM in the tenant network

```
nova boot --image <image-id> --flavor <flavor-id> \
  --nic net-id=port1_private1_uuid VM1
```

- Dump ODL BGP FIB

```
GET /restconf/config/odl-fib:fibEntries
{
  "fibEntries": {
    "vrfTables": [
      {
        "routeDistinguisher": <rd-uuid>
      },
      {
        "routeDistinguisher": <rd>,
        "vrfEntry": [
          {
            "destPrefix": <IPv6_VM1/128>,
            "label": <label>,
            "nextHopAddressList": [
              <DPN_IPv4>
            ],
            "origin": "1"
          },
          ]
        },
      ]
    }
  ]
}
```

## Features to Install

odl-netvirt-openstack

## REST API

## CLI

A new option `--afi` will be added to command `odl:bgp-network`:

```
opendaylight-user@root>
odl:bgp-network --prefix 2001:db8::1/128 --rd 100:1 --nexthop 192.168.0.2
                  --label 700 --afi 2 add/del
```

## Implementation

### Assignee(s)

**Primary assignee:** Julien Courtat <julien.courtat@6wind.com>

**Other contributors:** Noel de Prandieres <prandieres@6wind.com> Valentina Krasnobaeva  
<valentina.krasnobaeva@6wind.com> Philippe Guibert <philippe.guibert@6wind.com>

### Work Items

- Implement necessary APIs to allocate a transport over IPv6 requirement configuration for a given Route Target as the primary key.
- Support of BGPVPNV6 prefixes within MD-SAL. Enhance RIB-manager to support routes learned from other bgp speakers, [un]set static routes.
- BGP speaker implementation, Quagga BGP, to support BGPVPNV6 prefixes exchanges with other BGP speakers (interoperability), and thrift interface updates.
- Program necessary pipeline flows to support IPv6 to MPLS/GRE (IPv4) communication.

### Dependencies

Quagga from 6WIND is publicly available at the following url

- <https://github.com/6WIND/quagga>
- <https://github.com/6WIND/zrpcd>

### Testing

#### Unit Tests

Unit tests provided for the BGPVPNV4 versions will be enhanced to also support BGPVPNV6. No additional unit tests will be proposed.

#### Integration Tests

TBD

## CSIT

CSIT provided for the BGPVPNv4 versions will be enhanced to also support BGPVPNv6. No additional CSIT will be proposed.

## Documentation Impact

Necessary documentation would be added on how to use this feature.

## References

- [1] OpenDaylight Documentation Guide
- [2] <https://specs.openstack.org/openstack/nova-specs/specs/kilo/template.html>
- [3] <http://docs.openstack.org/developer/networking-bgpvpn/overview.html>
- [4] Spec to support IPv6 North-South support for Flat/VLAN Provider Network.
- [5] BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN

---

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

This work is licensed under a Creative Commons Attribution 3.0 Unported License. <http://creativecommons.org/licenses/by/3.0/legalcode>

---

### Table of Contents

- *IPv6 L3 North-South support for Flat/VLAN Provider Networks.*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*
  - *Usage*
    - \* *Features to Install*
    - \* *REST API*

- \* *CLI*
- *Implementation*
  - \* *Assignee(s)*
  - \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.14 IPv6 L3 North-South support for Flat/VLAN Provider Networks.

<https://git.opendaylight.org/gerrit/#/q/topic:ipv6-cvr-north-south>

In this specification we will be discussing the high level design of IPv6 North-South support in OpenDaylight for VLAN/FLAT provider network use-case.

#### Problem description

OpenDaylight currently supports IPv6 IPAM (IP Address Management) and a fully distributed east-west router. IPv6 external connectivity is not yet supported. This SPEC captures the implementation details of IPv6 external connectivity for VLAN/FLAT provider network use-cases.

We have a separate SPEC [3] that captures external connectivity for L3VPN use-case.

The expectation in OpenStack is that Tenant IPv6 subnets are created with Globally Unique Addresses (GUA) that are routable by the external physical IPv6 gateway in the datacenter for external connectivity. So, there is no concept of NAT or Floating-IPs for IPv6 addresses in Neutron. An IPv6 router is hence expected to do a plain forwarding.

Initially, we would like to pursue a Centralized IPv6 router (CVR) use-case and look into a fully distributed router via a future spec. One of the main reasons for pursuing the CVR over DVR is that OpenStack Neutron creates only a single router gateway port (i.e., port with device owner as network:router\_gateway) when the router is associated with the external network. When implementing a distributed router, we cannot use the same router gateway port MAC address from multiple Compute nodes as it could create issues in the underlying physical switches. In order to implement a fully distributed router, we would ideally require a router-gateway-port per compute node. We will be addressing the distributed router in a future spec taking into consideration both IPv4 and IPv6 use-cases.

#### Use Cases

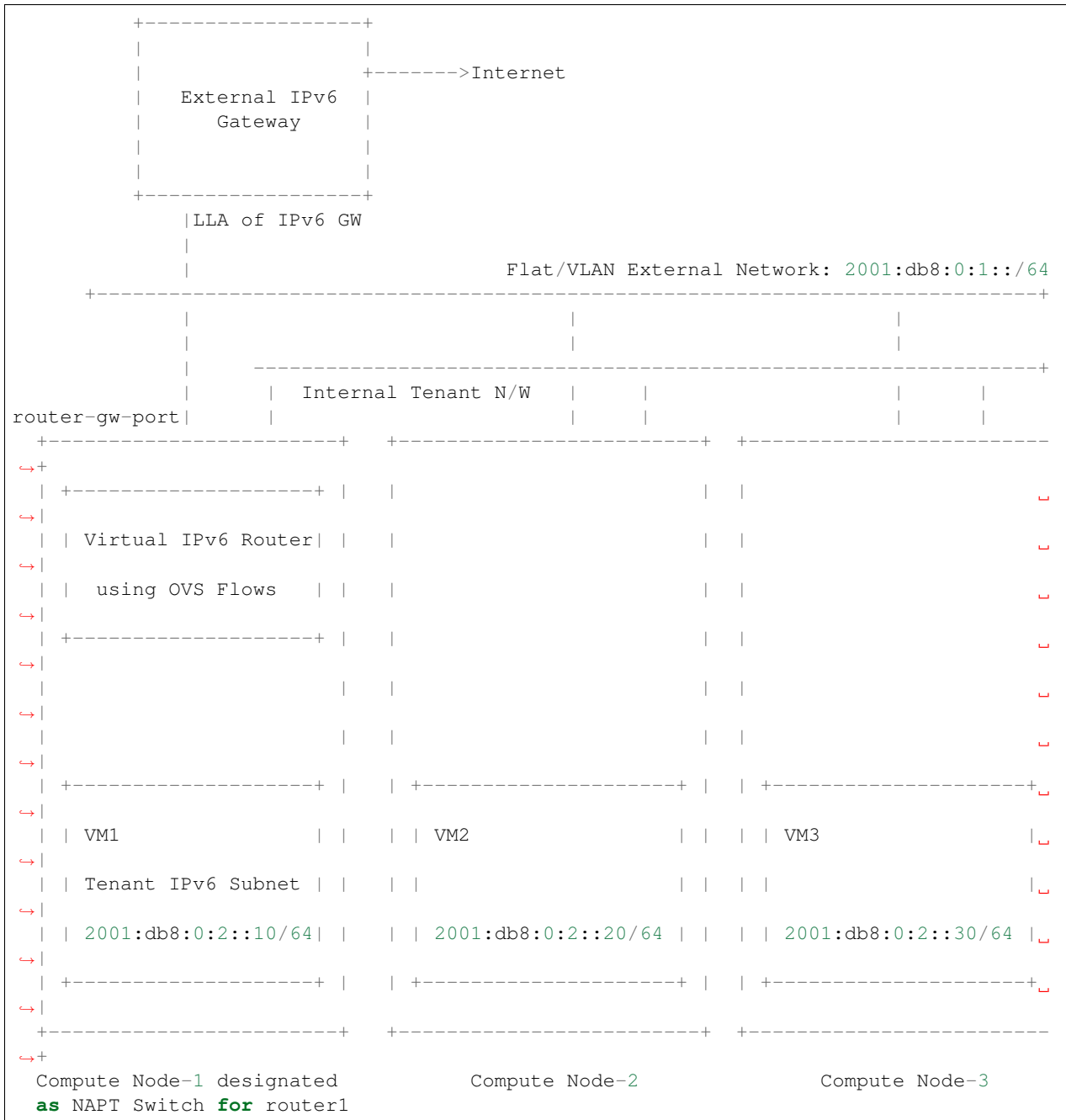
IPv6 external connectivity (north-south) for VMs spawned on tenant networks, when the external network is of type FLAT/VLAN based.

Steps:

- Create a tenant network with IPv6 subnet using GUA/ULA prefix or an admin-created-shared-ipv6-subnet-pool.



- Create an external network of type FLAT/VLAN with an IPv6 subnet where the gateway\_ip points to the Link Local Address (LLA) of external/physical IPv6 gateway.
- Create a Neutron Router and associate it with the internal subnets and external network.
- Spawn VMs on the tenant network.



## Proposed change

ODL Controller would implement the following.

- Program the necessary pipeline flows to support IPv6 forwarding

- Support Neighbor Discovery for Router Gateway port-ips on the external network. i.e., When the upstream/external IPv6 Gateway does a Neighbor Solicitation for the router-gateway-ip, ODL-Controller/ipv6service would respond with a Neighbor Advertisement providing the target link layer address.
- Enhance IPv6Service to learn the MAC-address of external-subnet-gateway-ip by framing the necessary Neighbor Solicitation messages and parsing the corresponding response. The APIs in IPv6Service would be triggered from Gateway MAC resolver code and the information obtained will be used while programming the Provider-NetworkGroup entries.

The implementation would be aligned with the existing IPv4 SNAT support we have in Netvirt. ODL controller would designate one of the compute nodes (also referred as NAPT Switch), one per router, to act as an IPv6/IPv4-SNAT router, from where the tenant traffic is routed to the external network. External traffic from VMs hosted on the NAPT switch is forwarded directly, whereas traffic from VMs hosted on other compute nodes would have to do an extra hop to NAPT switch before hitting the external network. If a router has both IPv4 and IPv6 subnets, the same NAPT Switch for the router will be used for IPv4-SNAT and IPV6 external-packet forwarding.

## Pipeline changes

### Flows on NAPT Switch for Egress traffic from VM to the internet

Classifier Table (0) =>

LPORT\_DISPATCHER\_TABLE (17) l3vpn service: set: vpn-id=router-id=>

L3\_GW\_MAC\_TABLE (19) priority=20, match: vpn-id=router-id,  
dst-mac=router-internal-interface-mac =>

L3\_FIB\_TABLE (21) priority=10, match: ipv6, vpn-id=router-id, default-route-flow  
=>

PSNAT\_TABLE (26) priority=5, match: ipv6, vpn-id=router-id, unknown-sip =>

OUTBOUND\_NAPT\_TABLE (46) priority=10, match: ipv6, vpn-id=router-id,  
ip-src=vm-ip set: src-mac=external-router-gateway-mac-address,  
vpn-id=external-net-id, =>

NAPT\_PFIB\_TABLE (47) priority=6, match: ipv6, vpn-id=external-net-id,  
src-ip=vm-ip =>

ProviderNetworkGroup: set dst-mac=ext-subnet-gw-mac, reg6=provider-lport-tag =>

EGRESS\_LPORT\_DISPATCHER\_TABLE (220) output to provider network

### Flows on NAPT Switch for Ingress traffic from internet to VM

Classifier Table (0) =>

LPORT\_DISPATCHER\_TABLE (17) l3vpn service: set: vpn-id=ext-net-id=>

L3\_GW\_MAC\_TABLE (19) priority=20, match: vpn-id=ext-net-id,  
dst-mac=router-gateway-mac =>

L3\_FIB\_TABLE (21) priority=138, match: ipv6, vpn-id=ext-net-id, dst-ip=vm-ip =>

INBOUND\_NAPT\_TABLE (44) priority=10, match: ipv6, vpn-id=ext-net-id,  
dst-ip=vm-ip set: vpn-id=router-id =>

NAPT\_PFIB\_TABLE (47) priority=5, match: ipv6, vpn-id=router-id set: in\_port=0  
=>

L3\_FIB\_TABLE (21) priority=138, match: ipv6, vpn-id=router-id, dst-ip=vm-ip =>

Local Next-Hop group: set: src-mac=router-intf-mac,  
dst-mac=vm-mac, reg6=vm-lport-tag =>

Egress table (220) output to VM port

### Flows for VMs hosted on Compute node that is not acting as an NAPT Switch

Same egress pipeline flows as above until L3\_FIB\_TABLE (21).

PSNAT\_TABLE (26) priority=5, match: ipv6, vpn-id=router-id set:  
tun\_id=<tunnel-id> =>

TunnelOutputGroup: output to tunnel-port =>

OnNAPTSwitch (for Egress Traffic from VM)

INTERNAL\_TUNNEL\_TABLE (36): priority=10, match: ipv6,  
tun\_id=<tunnel-id-set-on-compute-node> set: vpn-id=router-id,  
goto\_table:46

Rest of the flows are common.

OnNAPTSwitch (for Ingress Traffic from Internet to VM)

Same flows in ingress pipeline shown above until NAPT\_PFIB\_TABLE (47) =>

L3\_FIB\_TABLE (21) priority=138, match: ipv6, vpn-id=router-id, dst-ip=vm-ip  
set: tun\_id=<tunnel-id>, dst-mac=vm-mac, output: <tunnel-port> =>

### Yang changes

IPv6Service would implement the following YANG model.

```
module ipv6-ndutil {
  yang-version 1;
  namespace "urn:opendaylight:netvirt:ipv6service:ipv6util";
  prefix "ipv6-ndutil";

  import ietf-interfaces {
    prefix if;
  }

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

  import ietf-yang-types {
    prefix yang;
  }

  revision "2017-02-10" {
    description "IPv6 Neighbor Discovery Util module";
  }

  grouping interfaces {
    list interface-address {
      key interface;
      leaf interface {
        type leafref {
          path "/if:interfaces/if:interface/if:name";
        }
      }
      leaf src-ip-address {
```

```
        type inet:ipv6-address;
    }
    leaf src-mac-address {
        type yang:phys-address;
    }
}

rpc send-neighbor-solicitation {
    input {
        leaf target-ip-address {
            type inet:ipv6-address;
        }
        uses interfaces;
    }
}
}
```

neighbor-solicitation-packet container in neighbor-discovery.yang would be enhanced with Source Link Layer optional header.

```
container neighbor-solicitation-packet {
    uses ethernet-header;
    uses ipv6-header;
    uses icmp6-header;
    leaf reserved {
        type uint32;
    }
    leaf target-ip-address {
        type inet:ipv6-address;
    }
    leaf option-type {
        type uint8;
    }
    leaf source-addr-length {
        type uint8;
    }
    leaf source-ll-address {
        type yang:mac-address;
    }
}
```

## Configuration impact

None

## Clustering considerations

None

## Other Infra considerations

None

## Security considerations

## Scale and Performance Impact

- In the proposed implementation, we have to configure a static route on the external IPv6 Gateway with next-hop as the router-gateway-ip. In a future patch, we would enhance the implementation to use BGP for advertising the necessary routes.
- When the external IPv6 Gateway wants to contact the tenant VMs, it forwards all the traffic to the router-gateway-port on the designated NAPT Switch. To know the target-link-layer address of the router-gw-port, the external IPv6 Gateway would send out a Neighbor Solicitation for the router-gateway-port-ip. This request would be punted to the Controller and ipv6service would respond with the corresponding Neighbor Advertisement. In large deployments this can become a bottleneck. Note: Currently, OpenFlow does not have support to auto-respond to Neighbor Solicitation packets like IPv4 ARP. When the corresponding support is added in OpenFlow, we would program the necessary ovs flows to auto-respond to the Neighbor Solicitation requests for router-gateway-ports.

## Targeted Release

Carbon

## Alternatives

An alternate solution is to implement a fully distributed IPv6 router and would be pursued in a future SPEC.

## Usage

- Create an external FLAT/VLAN network with an IPv6 (or dual-stack) subnet.

```
neutron net-create public-net -- --router:external --is-default
--provider:network_type=flat --provider:physical_network=public

neutron subnet-create --ip_version 6 --name ipv6-public-subnet
--gateway <LLA-of-external-ipv6-gateway> <public-net-uuid> 2001:db8:0:1::/64
```

- Create an internal tenant network with an IPv6 (or dual-stack) subnet.

```
neutron net-create private-net
neutron subnet-create --name ipv6-int-subnet --ip-version 6
--ipv6-ra-mode slaac --ipv6-address-mode slaac private-net 2001:db8:0:2::/64
```

- Create a router and associate the external and internal subnets. Explicitly specify the fixed\_ip of router-gateway-port, as it would help us when manually configuring the downstream route on the external IPv6 Gateway.

```
neutron router-create router1
neutron router-gateway-set --fixed-ip subnet_id=<ipv6-public-subnet-id>, ip_
→address=2001:db8:0:10 router1 public-net
neutron router-interface-add router1 ipv6-int-subnet
```

- Manually configure a downstream route in the external IPv6 gateway for the IPv6 subnet “2001:db8:0:2::/64” with next hop address as the router-gateway-ip.

```
Example (on Linux host acting as an external IPv6 gateway):  
ip -6 route add 2001:db8:0:2::/64 via 2001:db8:0:10
```

- Spawn a VM in the tenant network

```
nova boot --image <image-id> --flavor <flavor-id> --nic net-id=<private-net> VM1
```

### Features to Install

odl-netvirt-openstack

### REST API

### CLI

### Implementation

### Assignee(s)

**Primary assignee:** Sridhar Gaddam <sgaddam@redhat.com>

**Other contributors:** TBD

### Work Items

<https://trello.com/c/cqjOFmow/147-ipv6-centralized-router-l3-north-south-support-for-flat-vlan-provider-networks>

- Program necessary pipeline flows to support IPv6 North-South communication.
- Enhance ipv6service to send out Neighbor Solicitation requests for the external/physical IPv6 gateway-ip and parse the response.
- Support controller based Neighbor Advertisement for router-gateway-ports on the external network.
- Implement Unit and Integration tests to validate the use-case.

### Dependencies

None

### Testing

#### Unit Tests

Necessary Unit tests would be added to validate the use-case.

#### Integration Tests

Necessary Integration tests would be added to validate the use-case.

## CSIT

We shall explore the possibility to validate this use-case in CSIT.

## Documentation Impact

Necessary documentation would be added on how to use this feature.

## References

[1] OpenDaylight Documentation Guide

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

[3] Spec to support IPv6 Inter DC L3VPN connectivity using BGPVPN

---

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

This work is licensed under a Creative Commons Attribution 3.0 Unported License. <http://creativecommons.org/licenses/by/3.0/legalcode>

---

### Table of Contents

- *Dual Stack VM support in OpenDaylight*
  - *Problem description*
  - *Setup Presentation*
  - *Known Limitations*
  - *Use Cases*
    - \* *Inter DC Access*
    - \* *External Internet Connectivity*
  - *Proposed changes*
  - *Pipeline changes*
    - \* *Traffic from DC-Gateway to Local DPN (SYMMETRIC IRB)*
    - \* *Traffic from Local DPN to DC-Gateway (SYMMETRIC IRB)*
  - *Configuration impact*
  - *ECMP impact*
  - *Clustering considerations*
  - *Other Infra considerations*
  - *Security considerations*
  - *Scale and Performance Impact*
  - *Targeted Release*
  - *Alternatives*

- *Usage*
- *Features to Install*
- *REST API*
- *CLI*
- *Implementation*
  - \* *Assignee(s)*
  - \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.15 Dual Stack VM support in OpenDaylight

<https://git.opendaylight.org/gerrit/#/q/topic:l3vpn-dual-stack-vm>

In this specification we will introduce a support of basic L3 forwarding for dualstack VMs connectivity over L3 in NetVirt. Dualstack VM is a virtual machine that has at least two IP addresses with different ethertypes: IPv4 address and IPv6 address.

In addition to this, the specification ensures initial support of dualstack VMs inside L3 BGPVPN. L3 forwarding for dualstack VMs connectivity inside L3 BGPVPN will be provided for the following variations of L3 BGPVPN:

1. L3 BGPVPN constructed purely using networks;
2. L3 BGPVPN constructed purely using a router;
3. L3 BGPVPN constructed using multiple networks and a router.

#### Problem description

As a dualstack VM, we assume a VM which has one Neutron Port, i.e. one VNIC, that inherits two IPs addresses with different ethertypes: one IPv4 address and one IPv6 address. We also will use in this document a term singlestack VM to describe a VM, which VNIC possesses either IPv4 or IPv6 address, but not both simultaneously.

So, dualstack VM has two IP addresses with different ethertypes. This could be achieved by two ways:

1. VM was initially created with one VNIC, i.e. one Neutron Port from network with IPv4 subnet. Second VNIC, corresponded to a Neutron Port from another network with IPv6 subnet, was added to this machine after its creation.
2. VM has one Neutron Port from a network, which contains 2 subnets: IPv4 subnet and IPv6 subnet.

OpenDaylight has already provided a support for the first way, so this use-case is not in the scope of the specification. For the second way the specification doesn't intend to cover a use-case when, Neutron Port will possess several IPv4 and several IPv6 addresses. More specifically this specification covers only the use-case, when Neutron Port has only one IPv4 and one IPv6 address.



Since there are more and more services that use IPv6 by default, support of dualstack VMs is important. Usage of IPv6 GUA addresses has increased during the last couple years. Administrators want to deploy services, which will be accessible from traditional IPv4 infrastructures and from new IPv6 networks as well.

Dualstack VM should be able to connect to other VMs, be they are of IPv4 (or) IPv6 ethertypes. So in this document we can handle following use cases:

- Intra DC, Inter-Subnet basic L3 Forwarding support for dualstack VMs;
- Intra DC, Inter-Subnet L3 Forwarding support for dualstack VMs within L3 BGPVPN.

Current L3 BGPVPN allocation scheme picks up only the first IP address of dualstack VM Neutron Port. That means that the L3 BGPVPN allocation scheme will not apply both IPv4 and IPv6 network configurations for a port. For example, if the first allocated IP address is IPv4 address, then L3 BGPVPN allocation scheme will only apply to IPv4 network configuration. The second IPv6 address will be ignored.

Separate VPN connectivity for singlestack VMs within IPv4 subnetworks and within IPv6 subnetworks is already achieved by using distinct L3 BGPVPN instances. What we want is to support a case, when the same L3 BGPVPN instance will handle both IPV4 and IPv6 VM connectivity.

Regarding the problem description above, we would propose to implement in OpenDaylight two following solutions, applying to two setups

#### 1. **two-router** setup solution

One router belongs to IPv4 subnetwork, another one belongs to IPv6 subnetwork. This setup brings flexibility to manage access to external networks. More specifically, by having two routers, where one is holding IPv4 subnet and another is holding IPv6 subnet, customer can tear-down access to external network for IPv4 subnet ONLY or for IPv6 subnet ONLY by doing a router-gateway-clear on a respective router.

Now this kind of orchestration step entail us to put a Single VPN Interface (representing the VNIC of DualStack VM) in two different Internal-VPNs, where each VPN represents one of the routers. To achieve this we will use L3 BGPVPN concept. We will extend existing L3 BGPVPN instance implementation to give it an ability to be associated with two routers. As consequence, IPv4 and IPv6 subnetworks, added as ports in associated routers and, hence, IPv4 and IPv6 FIB entries, would be gathered in one L3 BGPVPN instance.

L3 BGPVPN concept is the easiest solution to federate two routers in a single L3 BGPVPN entity. From the orchestration point of view and from the networking point of view, there is no any reason to provide IPv4 L3VPN and IPv6 L3VPN access separately for dualstack VMs. It makes sense to have the same L3 BGPVPN entity that can handle both IPv4 and IPv6 subnetworks.

The external network connectivity using L3 BGPVPN is not in scope of this specification. Please, find more details about this in [6]. Right now, this configuration will be useful for inter-subnet and intra-dc routing.

#### 2. **dualstack-router** setup solution

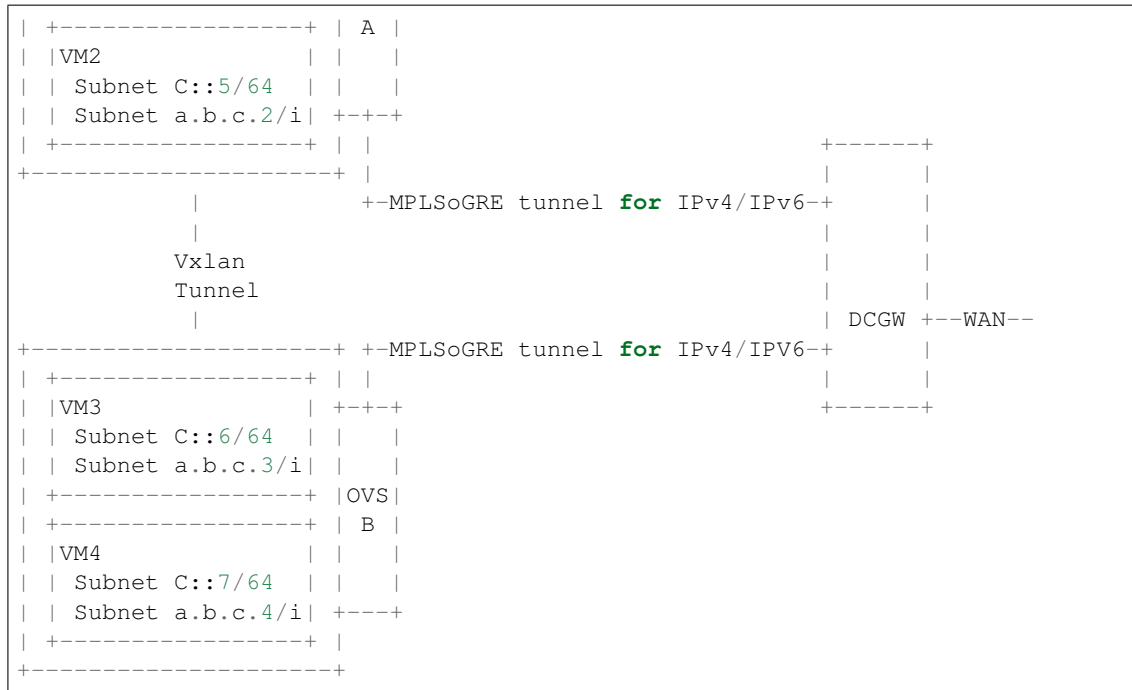
The router with 2 ports (one port for IPv4 subnet and another one for IPv6 subnet) is attached to a L3 BGPVPN instance.

The external network connectivity using L3 BGPVPN is not in the scope of this specification.

## Setup Presentation

Following drawing could help :



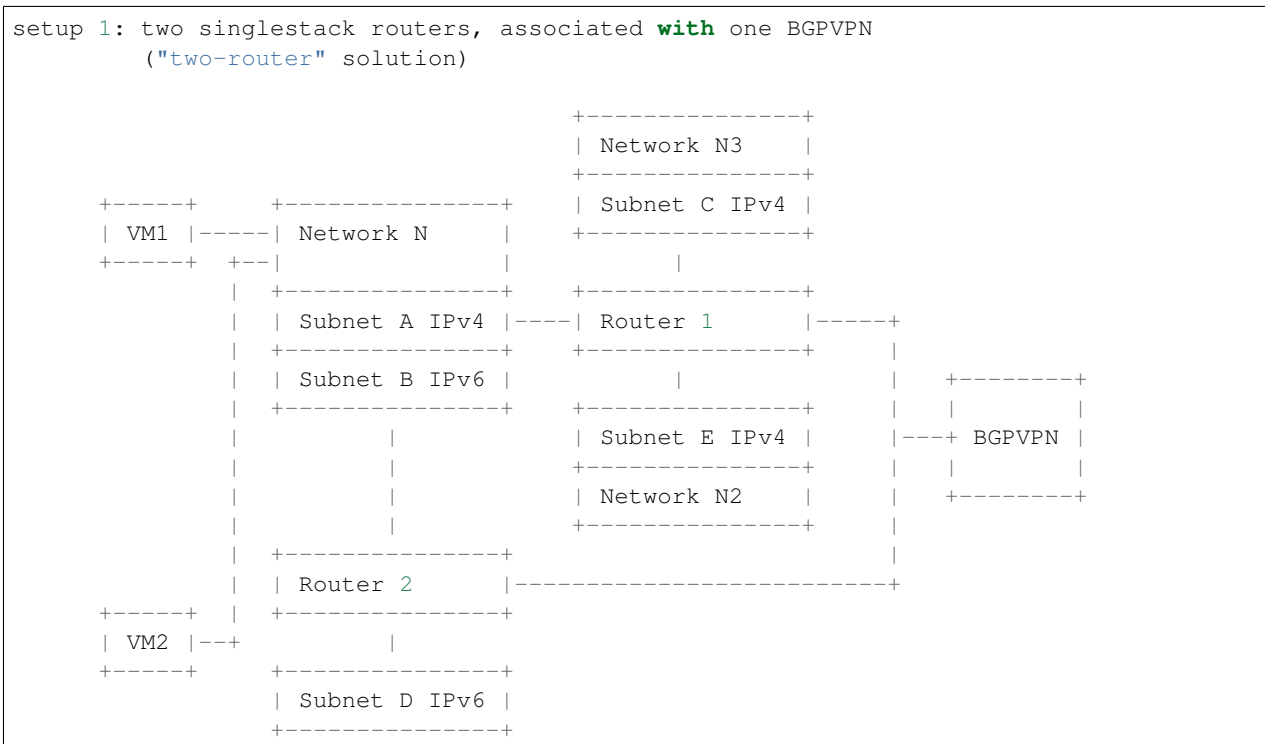


**We identify there 2 subnets:**

- IPv4 subnet: a.b.c.x/i
- IPv6 subnet: C::x/64

Each VM will receive IPs from these two defined subnets.

Following schemes stand for conceptual representation of used neutron configurations for each proposed solution.



```

| Network N1 |
+-----+

```

Network N gathers 2 subnetworks, subnet A IPv4 and subnet B IPv6. This makes possible to create Neutron Ports, which will have 2 IP addresses and whose attributes will inherit information (extraroutes, etc) from these 2 subnets A and B.

Router1 and Router2 are connected to Subnet A and Subnet B respectively and will be attached to a same L3 BGPVPN instance. Routers 1 and 2 can also have other ports, but they always should stay singlestack routers, otherwise this configuration will not be still supported. See the chapter “Configuration impact” for more details.

setup 2: one dualstack router associated **with** one BGPVPN  
 ("dualstack-router" solution)

```

+-----+      +-----+
| VM1 |-----| Network N |
+-----+  +---+ |         |
           | +-----+      +-----+      +-----+
           | | Subnet A IPv4 |-----+ | Router 1 |----+ BGPVPN |
           | +-----+      +-----+      +-----+
           | | Subnet B IPv6 |-----+ |         |
           | +-----+      +-----+      +-----+
+-----+ |
| VM2 |---+
+-----+

```

Network N gathers 2 subnetworks, subnet A IPv4 and subnet B IPv6. This makes possible to create Neutron Ports, which will have 2 IP addresses and whose attributes will inherit information (extraroutes, etc) from these 2 subnets A and B.

Router 1 is connected to Subnet A and Subnet B, and it will be attached to a L3 BGPVPN instance X. Other subnets can be added to Router 1, but this configurations will not be still supported. See the chapter “Configuration impact” for more details.

setup 3: networks associated **with** one BGPVPN

```

+-----+      +-----+      +-----+
| VM1 |-----| Network N1 |-----| BGPVPN |
+-----+  +---+ |         |         |
           | +-----+      +-----+      +-----+
           | | Subnet A IPv4 (1) |         |
+-----+ | +-----+      +-----+      +-----+
| VM2 |---+ | Subnet B IPv6 (2) |         |
+-----+ | +-----+      +-----+      +-----+
           |         |
           |         |
+-----+      +-----+      +-----+
| VM3 |-----+ Network N2 |-----+
+-----+      +-----+      +-----+
           | +-----+      +-----+
           | | Subnet C IPv4 (3) |
           | +-----+      +-----+
           | | Subnet D IPv6 (4) |
           | +-----+      +-----+

```

Network N1 gathers 2 subnets, subnet A with IPv4 ethertype and subnet B with IPv6 ethertype. When Neutron Port was created in the network N1, it has 1 IPv4 address and 1 IPv6 address. If user lately will add others subnets to the Network N1 and will create the second Neutron Port, anyway the second VPN port, constructed for a new Neutron

Port will keep only IP addresses from subnets (1) and (2). So valid network configuration in this case is a network with only 2 subnets: IPv4 and IPv6. See the chapter “Configuration impact” for more details. Second dualstack network N2 can be added to the same L3 BGPVPN instance.

It is valid for all schemes: in dependency of chosen ODL configuration, either ODL, or Neutron Dhcp Agent will provide IPv4 addresses for launched VMs. Please note, that currently DHCPv6 is supported only by Neutron Dhcp Agent. ODL provides only SLAAC GUA IPv6 address allocation for VMs launched in IPv6 private subnets attached to a Neutron router.

It is to be noted that today, setup 3 can not be executed for VPNv6 with the above allocation scheme previously illustrated. Indeed, only a neutron router is able to send router advertisements, which is the corner stone for DHCPv6 allocation. Either IPv6 fixed IPs will have to be used for this setup, or an extra enhancement for providing router advertisements for such a configuration will have to be done. The setup 3 will be revisited in future.

## Known Limitations

Currently, from Openstack-based Opendaylight Bgpvpn driver point-of-view, there is a check, where it does not allow more than one router to be associated to a single L3 BGPVPN. This was done in Openstack, because actually entire ODL modeling and enforcement supported only one router per L3 BGPVPN by design.

From Netvirt point of view, there are some limitations as well:

- We can not associate VPN port with both IPv4 and IPv6 Neutron Port addresses at the same time. Currently, any first Neutron Port IP address is using to create a VPN interface. If a Neutron Port possesses multiple IP Addresses, regardless of ethertype, this port might not work properly with ODL.
- It is not possible to associate a single L3 BGPVPN instance with two different routers.

## Use Cases

There is no change in the use cases described in [6] and [7], except that the single L3 BGPVPN instance serves both IPv4 and IPv6 subnets.

## Inter DC Access

### 1. **two-router** solution

IPv4 subnet Subnet A is added as a port in Router 1, IPv6 subnet Subnet B is added as a port in Router 2. The same L3 BGPVPN instance will be associated with both Router 1 and Router 2.

The L3 BGPVPN instance will distinguish ethertype of router ports and will create appropriate FIB entries associated to its own VPN entry, so IPv4 and IPv6 entries will be gathered in the same L3 BGPVPN.

### 2. **dualstack-router** solution

IPv4 subnet Subnet A is added as a port in Router 1, IPv6 subnet Subnet B is added as a port in Router 1 as well. L3 BGPVPN instance will be associated with Router 1.

The L3 BGPVPN instance will distinguish ethertype of routers ports and will create appropriate FIB entries associated to its own VPN entry as well. Appropriate BGP VRF context for IPv4 or IPv6 subnets will be also created.

## External Internet Connectivity

External Internet Connectivity is not in the scope of this specification.

## Proposed changes

All changes we can split in two main parts.

### 1. Distinguish IPv4 and IPv6 VRF tables with the same RD/iRT/eRT

#### 1.1 Changes in neutronvpn

To support a pair of IPv4 and IPv6 prefixes for each launched dualstack VM we need to obtain information about subnets, where dualstack VM was spawned and information about extraroutes, enabled for these subnets. Obtained information will be stored in `vmAdj` and `erAdjList` objects respectively. These objects are attributes of created for new dualstack VM VPN interface. Created VPN port instance will be stored as part of already existed L3 BGPVPN node instance in MDSAL DataStore.

When we update L3 BGPVPN instance node (associate/dissociated router or network), we need to provide information about ethertype of new attached/detached subnets, hence, Neutron Ports. New argument flags **ipv4On** and **ipv6On** will be introduced for that in **Neutron-vpnManager** function API, called to update current L3 BGPVPN instance (*updateVpnInstanceNode()* method). *UpdateVpnInstanceNode()* method is also called, when we create a new L3 BGPVPN instance. So, to provide appropriate values for **ipv4On**, **ipv6On** flags we need to parse subnets list. Then in dependency of these flags values we will set either **Ipv4Family** attribute for the new L3 BGPVPN instance or **Ipv6Family** attribute, or both attributes. **Ipv4Family**, **Ipv6Family** attributes allow to create ipv4 or/and ipv6 VRF context for underlayed vpnmanager and bgpmanager APIs.

#### 1.2. Changes in vpnmanager

When L3 BGPVPN instance is created or updated, VRF tables must be created for QBGp as well. What we want, is to introduce separate VRF tables, created according to **IPv4Family/IPv6Family** VPN attributes, i.e. we want to distinguish IPv4 and IPv6 VRF tables, because this will bring flexibility in QBGp. For example, if QBGp receives an entry IPv6 MPLSVpn on a router, which is expecting to receive only IPv4 entries, this entry will be ignored. The same for IPv4 MPLSVpn entries respectively.

So, for creating **VrfEntry** objects, we need to provide information about L3 BGPVPN instance ethertype (**Ipv4Family/Ipv6Family** attribute), route distinguishers list, route imports list and route exports lists (**RD/iRT/eRT**). **RD/iRT/eRT** lists will be simply obtained from subnetworks, attached to the chosen L3 BGPVPN. Presence of **IPv4Family**, **IPv6Family** in VPN will be translated in following `VpnInstanceListener` class attributes: **afiIpv4**, **afiIpv6**, **safiMplsVpn**, **safiEvpn**, which will be passed to *addVrf()* and *deleteVrf()* bgpmanager methods for creating/deleting either **IPv4 VrfEntry** or **IPv6 VrfEntry** objects.

**RD/iRT/eRT** lists will be the same for both **IPv4 VrfEntry** and **IPv6 VrfEntry** in case, when IPv4 and IPv6 subnetworks are attached to the same L3 BGPVPN instance.

#### 1.3 Changes in bgpmanager

In bgpmanager we need to change signatures of *addVrf()* and *deleteVrf()* methods, which will trigger signature changes of underlying API methods *addVrf()* and *delVrf()* from *Bgp-ConfigurationManager* class.

This allows *BgpConfigurationManager* class to create needed IPv4 VrfEntry and IPv6 VrfEntry objects with appropriate **AFI** and **SAFI** values and finally pass this appropriate **AFI** and **SAFI** values to *BgpRouter*.

*BgpRouter* represents client interface for thrift API and will create needed IPv4 and IPv6 VRF tables in QBGp.

#### 1.4 Changes in yang model

To support new attributes **AFI** and **SAFI** in bgpmanager classes, it should be added in ebgp.yang model:

```
list address-families {
  key "afi safi";
  leaf afi {
    type uint32;
    mandatory "true";
  }
  leaf safi {
    type uint32;
    mandatory "true";
  }
}
```

### 1.5 Changes in QBGp thrift interface

To support separate IPv4 and IPv6 VRF tables in QBGp we need to change signatures of underlying methods *addvrf()* and *delvrf()* in thrift API as well. They must include the address family and subsequent address families informations:

```
enum af_afi {
  AFI_IP = 1,
  AFI_IPV6 = 2,
}

i32 addVrf(1:layer_type l_type, 2:string rd, 3:list<string>_
↳irts, 4:list<string> erts,
          5:af_afi afi, 6:af_safi afi),
i32 delVrf(1:string rd, 2:af_afi afi, 3:af_safi safi)
```

## 2. Support of two routers, attached to the same L3 BGPVPN

### 2.1 Changes in neutronvpn

**two-router** solution assumes, that all methods, which are using to create, update, delete VPN interface or/and VPN instance must be adapted to a case, when we have a list of subnetworks and/or list of router IDs to attach. Due to this, appropriate changes need to be done in *nvpnManager* method APIs.

To support **two-router** solution properly, we also should check, that we do not try to associate to L2 BGPVPN a router, that was already associated to that VPN instance. Attached to L3 BGPVPN router list must contain maximum 2 router IDs. Routers, which IDs are in the list must be only singlestack routers. More information about supported router configurations is available below in chapter “Configuration Impact”.

For each created in dualstack network Neutron Port we take only the last received IPv4 address and the last received IPv6 address. So we also limit a length of subnets list, which could be attached to a L3 BGPVPN instance, to two elements. (More detailed information about supported network configurations is available below in chapter “Configuration Impact”.) Two corresponding **Subnetmap** objects will be created in *NeutronPortChangeListener* class for attached subnets. A list with created subnetmaps will be passed as argument, when *createVpnInterface* method will be called.

### 2.2 Changes in vpnmanager

*VpnMap* structure must be changed to support a list with router IDs. This change triggers modifications in all methods, which retry router ID from *VpnMap* object.

*VpnInterfaceManager* structure must be also changed, to support a list of VPN instance name. So all methods, which gives VPN router ID from *VpnInterfaceManager* should be modified as well.

As consequence, in operDS, a *VpnInterfaceOpDataEntry* structure is created, inherited from *VpnInterface* in configDS. While the latter structure has a list of VPN instance name, the former will be instantiated in operDS as many times as there are VPN instances. The services that were handling *VPNInterface* in operDS, will be changed to handle *VPNInterfaceOpDataEntry*. That structure will be indexed by InterfaceName and by VPNName. The services include natservice, fibmanager, vpnmanager, cloud service chain.

Also, an augment structure will be done for *VPNInterfaceOpDataEntry* to contain the list of operational adjacencies. As for *VpnInterfaceOpDataEntry*, the new *AdjacenciesOp* structure will replace Adjacencies that are in operDS. Similarly, the services will be modified for that.

Also, *VPNInterfaceOpDataEntry* will contain a *VPNInterfaceState* that stands for the state of the VPN Interface. Code change will be done to reflect the state of the interface. For instance, if VPNInstance is not ready, associated *VPNInterfaceOpDataEntry*s will have the state changed to INACTIVE. Reversely, the state will be changed to ACTIVE.

## 2.3 Changes in yang model

To provide change in *VpnMap* and in *VpnInterfaceManager* structures, described above, we need to modify following yang files.

### 2.3.1 neutronvpn.yang

- Currently, container *vpnMap* holds one router-id for each L3 BGPVPN instance ID. A change consists in replacing one router-id leaf by a leaf-list of router-ids. Obviously, no more than two router-ids will be used.
- Container *vpnMaps* is used internally for describing a L3 BGPVPN. Change router-id leaf by router-ids leaf-list in this container is also necessary.

```

--- a/vpnservice/neutronvpn/neutronvpn-api/src/main/yang/
↪neutronvpn.yang
+++ b/vpnservice/neutronvpn/neutronvpn-api/src/main/yang/
↪neutronvpn.yang
@@ -1,4 +1,3 @@
-
-
module neutronvpn {

namespace "urn:opendaylight:netvirt:neutronvpn";
@@ -120,7 +119,7 @@ module neutronvpn {
Format is ASN:nn or IP-address:nn.";
}

-      leaf router-id {
+      leaf-list router-ids {
          type yang:uuid;
          description "UUID router list";
      }
@@ -173,7 +172,7 @@ module neutronvpn {
description "The UUID of the tenant that will own the subnet.";
}

-      leaf router-id {
+      leaf-list router_ids {
          type yang:uuid;

```

```

        description "UUID router list";
    }

```

### 2.3.2 l3vpn.yang

- Currently, list vpn-interface holds a leaf vpn-instance-name, which is a container for VPN router ID. A change consists in replacing leaf vpn-instance-name by a leaf-list of VPN router IDs, because L3 BGPVPN instance can be associated with two routers. Obviously, no more than two VPN router-IDs will be stored in leaf-list vpn-instance-name.

```

--- a/vpnservice/vpnmanager/vpnmanager-api/src/main/yang/
↪l3vpn.yang
+++ b/vpnservice/vpnmanager/vpnmanager-api/src/main/yang/
↪l3vpn.yang
@@ -795,21 +795,21 @@

    list vpn-interface {
        key "name";
        max-elements "unbounded";
        min-elements "0";
        leaf name {
            type leafref {
                path "/if:interfaces/if:interface/if:name";
            }
        }
-    leaf vpn-instance-name {
+    leaf-list vpn-instance-name {
        type string {
            length "1..40";
        }
        leaf dpn-id {
            type uint64;
        }
        leaf scheduled-for-remove {
            type boolean;
        }
    }

```

### 2.3.3 odl-l3vpn.yang

```

augment "/odl-l3vpn:vpn-interface-op-data/odl-l3vpn:vpn-
↪interface-op-data-entry" {
    ext:augment-identifier "adjacencies-op";
    uses adjacency-list;
}

container vpn-interface-op-data {
    config false;
    list vpn-interface-op-data-entry {
        key "name vpn-instance-name";
        leaf name {
            type leafref {
                path "/if:interfaces/if:interface/if:name";
            }
        }
        leaf vpn-instance-name {
            type string {

```



```

        length "1..40";
    }
}
max-elements "unbounded";
min-elements "0";
leaf dpn-id {
    type uint64;
}
leaf scheduled-for-remove {
    type boolean;
}
leaf router-interface {
    type boolean;
}
leaf vpn-interface-state {
    description
        "This flag indicates the state of this interface_
↪in the VPN identified by vpn-name.
        ACTIVE state indicates that this vpn-interface_
↪is currently associated to vpn-name
        available as one of the keys.
        INACTIVE state indicates that this vpn-
↪interface has already been dis-associated
        from vpn-name available as one of the keys.";

    type enumeration {
        enum active {
            value "0";
            description
                "Active state";
        }
        enum inactive {
            value "1";
            description
                "Inactive state";
        }
    }
    default "active";
}
}
}

```

## Pipeline changes

There is no change in the pipeline, regarding the changes already done in [6] and [7].

## Traffic from DC-Gateway to Local DPN (SYMMETRIC IRB)

The DC-GW has the information, that permits to detect an underlay destination IP and MPLS label for a packet coming from the Internet or from another DC-GW.

Classifier Table (0) =>

LFIB Table (20) match: tun-id=mpls\_label set vpn-id=l3vpn-id, pop\_mpls label, set output to nexthopgroup-dst-vm =>

```
NextHopGroup-dst-vm: set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag=>
Lport Egress Table (220) Output to dst vm port
```

### Traffic from Local DPN to DC-Gateway (SYMMETRIC IRB)

Classifier Table (0) =>

```
Lport Dispatcher Table (17) match: LportTag l3vpn service: set vpn-id=l3vpn-id=>
```

```
DMAC Service Filter (19) match: dst-mac=router-internal-interface-mac l3vpn service:
set vpn-id=l3vpn-id=>
```

```
L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=ext-ipv4-address set
tun-id=mpls_label output to MPLSoGRE tunnel port=>
```

```
L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=ext-ipv6-address set
tun-id=mpls_label output to MPLSoGRE tunnel port=>
```

Please, note that `router-internal-interface-mac` stands for MAC address of the internal subnet gateway router port.

### Configuration impact

#### 1. Limitations for router configurations

- 1.1 Maximum number of singlestack routers that can be associated to a L3BGPVPN** is limited to 2. Maximum number of dualstack routers that can be associated with a BGPVPN is limited to 1.
- 1.2 If a L3 BGPVPN has already associated with a one singlestack router and we try to associate this VPN instance again with a dualstack router,** exception will not be raised. But this configuration will not be valid.
- 1.3 If a singlestack router is already associated to a L3 BGPVPN instance, and** it has more than one port and we try to add a port to this router with another ethertype, i.e. we try to make this router dualstack, exception will not be raised. But this configuration will not be valid and supported.
- 1.4 When a different ethertype port is added to a singlestack router, which already has** only one port and which is already associated to a L3 BGPVPN instance, singlestack router in this case becomes dualstack router with only two ports. This router configuration is allowed by current specification.

#### 2. Limitations for subnetworks configurations

- 2.1 Maximum numbers of different ethertype subnetworks associated to a one L3 BGPVPN** instance is limited to two. If a network contains more than two different ethertype subnetworks, exception won't be raised, but this configuration isn't supported.
- 2.2 When we associate a network with a L3 BGPVPN instance, we do not care if** subnetworks from this network are ports in some routers and these routers were associated with other VPNs. This configuration is not considered as supported as well.

#### 3. Limitations for number of IP addresses for a Neutron Port

The specification only targets dual-stack networks, that is to say with 1 IPv4 address and one IPv6 address only. For other cases, that is to say, adding subnetworks IPv4 or IPv6, will lead to undefined or untested use cases. The multiple subnets test case would be handled in a future spec.

## ECMP impact

ECMP - Equal Cost multiple path.

ECMP feature is currently provided for Neutron BGPVPN networks and described in the specification [10]. 3 cases have been cornered to use ECMP feature for BGPVPN usability.

- ECMP of traffic from DC-GW to OVS (inter-DC case)
- ECMP of traffic from OVS to DC-GW (inter-DC case)
- ECMP of traffic from OVS to OVS (intra-DC case)

In each case, traffic begins either at DC-GW or OVS node. Then it is sprayed to end either at OVS node or DC-GW.

ECMP feature for Neutron BGPVPN networks was successfully (OK) tested with IPv4 L3 BGPVPN and IPv6 L3 BGPVPN (OK). the dual stack VM connectivity should embrace ECMP

We've included this chapter to remind, that code changes for supporting dualstack VMs should be tested against ECMP scenario as well.

## Clustering considerations

None

## Other Infra considerations

None

## Security considerations

None

## Scale and Performance Impact

None

## Targeted Release

Carbon

## Alternatives

None

## Usage

Assume, that in the same provider network we have OpenStack installed with 1 controller and 2 compute nodes, DC-GW node and OpenDaylight node.

- create private tenant networks and subnetworks
  - create Network N;

- declare Subnet A IPv4 for Network N;
  - declare Subnet B IPv6 for Network N;
  - create two ports in Network N;
  - each port will inherit a dual IP configuration.
- create routers
  - **two-router** solution + create two routers A and B, each router will be respectively connected to IPv4 and IPv6 subnets;
    - \* add subnet A as a port to router A;
    - \* add subnet B as a port to router B.
  - **dualstack-router** solution + create router A; + add subnet A as a port to router A; + add subnet B as a port to router A.
- Create MPLSoGRE tunnel between DPN and DCGW

```
POST /restconf/operations/itm-rpc:add-external-tunnel-endpoint
{
  "itm-rpc:input": {
    "itm-rpc:destination-ip": "dcgw_ip",
    "itm-rpc:tunnel-type": "odl-interface:tunnel-type-mpls-over-gre"
  }
}
```

- create the DC-GW VPN settings
  - Create a L3 BGPVPN context. This context will have the same settings as in [7]. In dualstack case both IPv4 and IPv6 prefixes will be injected in the same L3 BGPVPN.
- create the ODL L3 BGPVPN settings
  - Create a BGP context. This step permits to start QBGW module depicted in [8] and [9]. ODL has an API, that permits interfacing with that external software. The BGP creation context handles the following:
    - \* start of BGP protocol;
    - \* declaration of remote BGP neighbor with the AFI/SAFI affinities. In our case, VPNv4 and VPNv6 address families will be used.
  - Create a L3 BGPVPN, this L3 BGPVPN will have a name and will contain VRF settings.
- associate created L3 BGPVPN to router
  - **two-router** solution: associate routers A and B with a created L3 BGPVPN;
  - **dualstack-router** solution: associate router A with a created L3 BGPVPN.
- Spawn a VM in a created tenant network:

The VM will possess IPv4 and IPv6 addresses from subnets A and B.
- Observation: dump ODL BGP FIB entries

At ODL node, we can dump ODL BGP FIB entries and we should see entries for both IPv4 and IPv6 subnets prefixes:

```
GET /restconf/config/odl-fib:fibEntries
{
  "fibEntries": {
```

```

    "vrfTables": [
      {
        "routeDistinguisher": <rd-uuid>
      },
      {
        "routeDistinguisher": <rd>,
        "vrfEntry": [
          {
            "destPrefix": <IPv6_VM1/128>,
            "label": <label>,
            "nextHopAddressList": [
              <DPN_IPv4>
            ],
            "origin": "I"
          },
          ...
        ]
      },
      ...
    ]
  }
}

```

## Features to Install

odl-netvirt-openstack

## REST API

## CLI

A new option `--afi` and `--safi` will be added to command `odl:bgp-vrf`:

```
odl:bgp-vrf --rd <> --import-rt <> --export-rt <> --afi <1|2> --safi <value> add|del
```

## Implementation

### Assignee(s)

**Primary assignee:** Philippe Guibert <philippe.guibert@6wind.com>

### Other contributors:

- Valentina Krasnobaeva <valentina.krasnobaeva@6wind.com>
- Noel de Prandieres <prandieres@6wind.com>

## Work Items

- QBGp Changes
- BGpManager changes
- VPNManager changes
- NeutronVpn changes

## Dependencies

Quagga from 6WIND is available at the following urls:

- <https://github.com/6WIND/quagga>
- <https://github.com/6WIND/zrpd>

## Testing

### Unit Tests

Some L3 BGPVPN testing may have been done. Complementary specification for other tests will be done.

### Integration Tests

TBD

### CSIT

Basically, IPv4 and IPv6 vpnservice functionality have to be validated by regression tests with a single BGPVRF.

CSIT specific testing will be done to check dualstack VMs connectivity with network configurations for **two-router** and **dualstack-router** solutions.

**Two-router** solution test suite:

1. Create 2 Neutron Networks NET\_1\_2RT and NET\_2\_2RT.
  - 1.1 Query ODL restconf API to check that both Neutron Network objects were** successfully created in ODL.
  - 1.2 Update NET\_1\_2RT with a new description attribute.
2. In each Neutron Network create one Subnet IPv4 and one Subnet IPv6: SUBNET\_V4\_1\_2RT, SUBNET\_V6\_1\_2RT, SUBNET\_V4\_2\_2RT, SUBNET\_V6\_2\_2RT, respectively.
  - 2.1 Query ODL restconf API to check that all Subnetwork objects were** successfully created in ODL.
  - 2.2 Update SUBNET\_V4\_2RT, SUBNET\_V6\_2RT with a new description attribute.
3. Create 2 Routers: ROUTER\_1 and ROUTER\_2.
  - 3.1 Query ODL restconf API to check that all Router objects were successfully** created in ODL.
4. Add SUBNET\_V4\_1\_2RT, SUBNET\_V4\_2\_2RT to ROUTER\_1 and SUBNET\_V6\_1\_2RT, SUBNET\_V6\_2\_2RT to ROUTER\_2.
5. Create 2 security-groups: SG6\_2RT and SG4\_2RT. Add appropriate rules to allow IPv6 and IPv4 traffic from/to created subnets, respectively.
6. In network NET\_1\_2RT create Neutron Ports: PORT\_11\_2RT, PORT\_12\_2RT, attached with security groups SG6\_2RT and SG4\_2RT; in network NET\_2\_2RT: PORT\_21\_2RT, PORT\_22\_2RT, attached with security groups SG6\_2RT and SG4\_2RT.
  - 6.1 Query ODL restconf API to check, that all Neutron Port objects were** successfully created in ODL.
  - 6.2 Update Name attribute of PORT\_11\_2RT.

7. Use each created Neutron Port to launch a VM with it, so we should have 4 VM instances: VM\_11\_2RT, VM\_12\_2RT, VM\_21\_2RT, VM\_22\_2RT.
  - 7.1 Connect to NET\_1\_2RT and NET\_2\_2RT dhcp-namespaces, check that subnet routes were successfully propagated.**
  - 7.2 Check that all VMs have: one IPv4 address and one IPv6 addresses.
8. Check IPv4 and IPv6 VMs connectivity within NET\_1\_2RT and NET\_2\_2RT.
9. Check IPv4 and IPv6 VMs connectivity across NET\_1\_2RT and NET\_2\_2RT with ROUTER\_1 and ROUTER\_2.
  - 9.1 Check that FIB entries were created for spawned Neutron Ports.
  - 9.2 Check that all needed tables (19, 17, 81, 21) are presented in OVS pipelines and VMs IPs, gateways MAC and IP addresses are taken in account.**
10. Connect to VM\_11\_2RT and VM\_21\_2RT and add extraroutes to other IPv4 and IPv6 subnets.
  - 10.1 Check other IPv4 and IPv6 subnets reachability from VM\_11\_2RT and VM\_21\_2RT.**
11. Delete created extraroutes.
12. Delete and recreate extraroutes and check its reachability again.
13. Create L3VPN and check with ODL REST API, that it was successfully created.
14. Associate ROUTER\_1 and ROUTER\_2 with created L3VPN and check the presence of router IDs in VPN instance with ODL REST API.
15. Check IPv4 and IPv6 connectivity accross NET\_1\_2RT and NET\_2\_2RT with associated to L3VPN routers.
  - 15.1 Check with ODL REST API, that VMs IP addresses are presented in VPN interfaces entries.**
  - 15.2 Verify OVS pipelines at compute nodes.
  - 15.3 Check the presence of VMs IP addresses in vrfTables objects with ODL REST API query.**
16. Dissociate L3VPN from ROUTER\_1 and ROUTER\_2.
17. Delete ROUTER\_1 and ROUTER\_2 and its interfaces from L3VPN.
18. Try to delete router with NonExistentRouter name.
19. Associate L3VPN to NET\_1\_2RT.
20. Dissociate L3VPN from NET\_1\_2RT.
21. Delete L3VPN.
22. Create multiple L3VPN.
23. Delete multiple L3VPN.

## Documentation Impact

Necessary documentation would be added if needed.

## References

- [1] [OpenDaylight Documentation Guide](#)
- [2] <https://specs.openstack.org/openstack/nova-specs/specs/kilo/template.html>
- [3] <http://docs.openstack.org/developer/networking-bgpvpn/overview.html>

- [4] Spec to support IPv6 North-South support for Flat/VLAN Provider Network.
- [5] BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN
- [6] Spec to support IPv6 DC to Internet L3VPN connectivity using BGPVPN
- [7] Spec to support IPv6 Inter DC L3VPN connectivity using BGPVPN
- [8] Zebra Remote Procedure Call
- [9] Quagga BGP protocol

### 1.1.16 Listener Dependency Helper

<https://git.opendaylight.org/gerrit/#/q/topic:ListenerDependencyHelper>

Listener Dependency Helper makes “Data Store Listeners” independent from dependency resolution.

#### Problem description

When a DataStore-Listener is fired with config add/update/delete event, as part of listener processing it may try to read the other data store objects, at times those datastore objects are not yet populated. In this scenario, listener event processing has to be delayed (or) discarded, as the required information is NOT entirely available. Later when the dependant data objects are available, this listener event will not be triggered again by DataStore.

This results in some events not getting processed resulting in possible data-path, bgp control and data plane failures.

**Example:** VpnInterface add() callback triggered by MD-SAL on vpnInterface add. While processing add() callback, the corresponding vpnInstance is expected to be present in MD-SAL operational DS; which means that vpnInstance creation is complete (updating the vpn-targets in Operational DS and BGP).

Information: vpnInstance Config-DS listener thread has to process vpnInstance creation and update vpnInstance in operational DS. vpnInstance creation listener callback is handled by different listener thread.

#### Use Cases

**Use Case 1:** VPNInterfaces may get triggered before VPNInstance Creation.

Current implementation: Delay based waits for handling VPNInterfaces that may get triggered before VPNInstance Creation(waitForVpnInstance()).

**Use Case 2:** VPNManager to handle successful deletion of VPN which has a large number of BGP Routes (internal/external);

Current implementation: Delay-based logic on VPNInstance delete in VPNManager (waitForOpRemoval()).

**Use Case 3:** VpnSubnetRouteHandler that may get triggered before VPNInstance Creation.

Current implementation: Delay based waits in VpnSubnetRouteHandler which may get triggered before VPNInstance Creation(waitForVpnInstance()).

**Use Case 4:** VPN Swaps (Internal to External and vice-versa)

Current implementation: Currently we support max of 100 VM's for swap (VpnInterfaceUpdateTimerTask, waitFibToRemoveVpnPrefix()).



## Proposed change

During Listener event call-back (AsyncDataTreeChangeListenerBase) from DataStore, check for pending events in “Listener-Dependent-Queue” with same InstanceIdentifier to avoid re-ordering.

### Generic Queue Event Format:

key : Instance Identifier eventType : Type of event (ADD/UPDATE/DELETE) oldData : Data before modification (for Update event); newData : Newly populated data queuedTime : at which the event is queued to LDH. lastProcessedTime : latest time at which dependency list verified expiryTime : beyond which processing for event is useless waitBetweenDependencyCheckTime : wait time between each dependency check dependentIIDs : list of dependent InstanceIdentifiers retryCount : max retries allowed. databroker : data broker. deferTimerBased : flag to choose between (timer/listener based).

For Use Case - 1: deferTimerBased shall be set to TRUE (as per the specification).

During processing of events (either directly from DataStore or from “Listener-Dependent-Queue”), if there any dependent objects are yet to be populated; queue them to “Listener-Dependent-Queue”.

Expectations from Listener: Listener will push the callable instance to “Listener-Dependent-Queue” if it cannot proceed with processing of the event due to dependent objects/InstanceIdentifier and list of dependent IID’s.

There are two approaches the Listener Dependency check can be verified.

**approach-1** Get the list of dependent-IID’s, query DataStore/Cache for

dependency resolution at regular intervals using “timer-task-pool”. Once all the dependent IID’s are resolved, call respective listener for processing.

LDH-task-pool : pool of threads which query for dependency resolution READ ONLY operation in DataStore. These threads are part of LDH common for all listeners.

hasDependencyResolved(<InstanceIdentifier iid, Boolean shouldDataExist, DataStoreType DType> List), this shall return either Null list (or) the list which has dependencies yet to be resolved. In case Listener has local-cache implemented for set of dependencies, it can look at cache and identify. This api will be called from LDH-task-pool of thread(s).

instanceIdentifier is the MD-SAL key value which need to be verified for existence/non-existence of data. Boolean shouldDataExist: shall be TRUE, if the Listener expects to have the information exists in MD-SAL; False otherwise.

**approach-2** Register Listener for wild-card path of IID’s.

When a Listener gets queued to “Listener-Dependent-Queue”, LDH shall register itself as Listener for the dependent IID’s (using wild-card-path/parent-node). Once the listener gets fired, identify the dependent listeners waiting for the Data. Once the dependent Listener is identified, if the dependent-IID list is NULL. Trigger listener for processing the event. LDH-task-pool shall unregister itself from wild-card-path/parent-node once there are no dependent listeners on child-nodes.

### Re-Ordering

The following scenario, when re-ordering can happen and avoidance of the same:

**Example: Key1 and Value1 are present in MD-SAL Data Store under Tree1, SubTree1** (for say). Update-Listener for Key1 is dependent on Dependency1.

Key1 received UPDATE event (UPDATE-1) with value=x, at the time of processing UPDATE-1, dependency is not available. So Listener Queued ‘UPDATE-1’ event to “UnProcessed-EventQueue”. same key1 received UPDATE event (UPDATE-2) with value=y, at the time of processing UPDATE-2, dependency is available (Dependency1 is resolved), so it goes and processes the event and updates value of Key1 to y.

After WaitTime, event Key1, UPDATE-1 is de-queued from “UnProcessed-EventQueue” and put for processing in Lister. Listener processes it and updates the Key1 value to x. (which is incorrect, happened due to re-ordering of events).

To avoid reordering of events within listener, every listener call back shall peek into “UnProcessed-EventQueue” to identify if there exists a pending event with same key value; if so, either suppress (or) queue the event. Below are event ordering expected from MD-SAL and respective actions:

**what to consider before processing the event to avoid re-ordering of events:**

Current Event	Queued Event	Action
ADD	ADD	NOT EXPECTED
ADD	REMOVE	QUEUE THE EVENT
ADD	UPDATE	NOT EXPECTED
UPDATE	ADD	QUEUE EVENT
UPDATE	UPDATE	QUEUE EVENT
UPDATE	REMOVE	NOT EXPECTED
REMOVE	ADD	SUPPRESS BOTH
REMOVE	UPDATE	EXECUTE REMOVE SUPPRESS UPDATE
REMOVE	REMOVE	NOT EXPECTED

## Pipeline changes

none

## Yang changes

none

## Configuration impact

none

## Clustering considerations

In the two approaches mentioned: 1 - Timer: polling MD-SAL for dependency resolution may incur in more number of reads.

2 - RegisterListener: RegisterListener may some impact at the time of registering listener after which a notification message to cluser nodes.

## Predined List of Listeners

perational/odl-l3vpn:vpn-instance-op-data/vpn-instance-op-data-entry/\*      operational/odl-l3vpn:vpn-instance-op-data/vpn-instance-op-data-entry/

vpn-id/vpn-to-dpn-list/\*

config/l3vpn:vpn-instances/\*

## Other Infra considerations

## Security considerations

none

## Scale and Performance Impact

this infra, shall improve scaling of application without having to wait for dependent data store gets populated. Performance shall remain intact.

## Targeted Release

## Alternatives

- use polling/wait mechanisms

## Features to Install

## REST API

## CLI

CLI will be added for debugging purpose.

## Implementation

## Assignee(s)

Primary assignee: Siva Kumar Perumalla ([sivakumar.perumalla@ericsson.com](mailto:sivakumar.perumalla@ericsson.com))

Other contributors: Suneelu Verma K.

Work Items

Dependencies

Testing

Unit Tests

Integration Tests

CSIT

Documentation Impact

References

Acronyms

IID: InstanceIdentifier

## Table of Contents

- *New SFC Classifier*
  - *Terminology*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Integration with Genius*
    - \* *Classifier and SFC Genius Services*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*
  - *Usage*
    - \* *Features to Install*
    - \* *REST API*

- \* *CLI*
- *Implementation*
  - \* *Assignee(s)*
  - \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.17 New SFC Classifier

<https://git.opendaylight.org/gerrit/#/q/topic:new-sfc-classifier>

The current SFC Netvirt classifier only exists in the old Netvirt. This blueprint explains how to migrate the old Netvirt classifier to a new Netvirt classifier.

#### Terminology

- NSH - Network Service Headers, used as Service Chaining encapsulation. NSH RFC Draft [1]
- NSI - Network Service Index, a field in the NSH header used to indicate the next hop
- NSP - Network Service Path, a field in the NSH header used to indicate the service chain
- RSP - Rendered Service Path, a service chain.
- SFC - Service Function Chaining. SFC RFC [2] ODL SFC Wiki [3].
- SF - Service Function
- SFF - Service Function Forwarder
- VXGPE - VXLAN GPE (Generic Protocol Encapsulation) Used as transport for NSH. VXGPE uses the same header format as traditional VXLAN, but adds a Next Protocol field to indicate NSH will be the next header. Traditional VXLAN implicitly expects the next header to be ethernet. VXGPE RFC Draft [4].

#### Problem description

In the Boron release, an SFC classifier was implemented, but in the old Netvirt. This blueprint intends to explain how to migrate the old Netvirt classifier to a new Netvirt classifier, which includes integrating the classifier and SFC with Genius.

The classifier is an integral part of Service Function Chaining (SFC). The classifier maps client/tenant traffic to a service chain by matching the packets using an ACL, and once matched, the classifier encapsulates the packets using some sort of Service Chaining encapsulation. Currently, the only supported Service Chaining encapsulation is NSH

using VXGPE as the transport. Very soon (possibly in the Carbon release) Vxlan will be added as another encapsulation/transport, in which case NSH is not used. The transport and encapsulation information to be used for the service chain is obtained by querying the Rendered Service Path (RSP) specified in the ACL action.

The transport and encapsulation used between the classifier and the SFF, and also between SFFs will be VXGPE+NSH. The transport and encapsulation used between the SFF and the SF will be Ethernet+NSH.

The following image details the packet headers used for Service Chaining encapsulation with VXGPE+NSH.

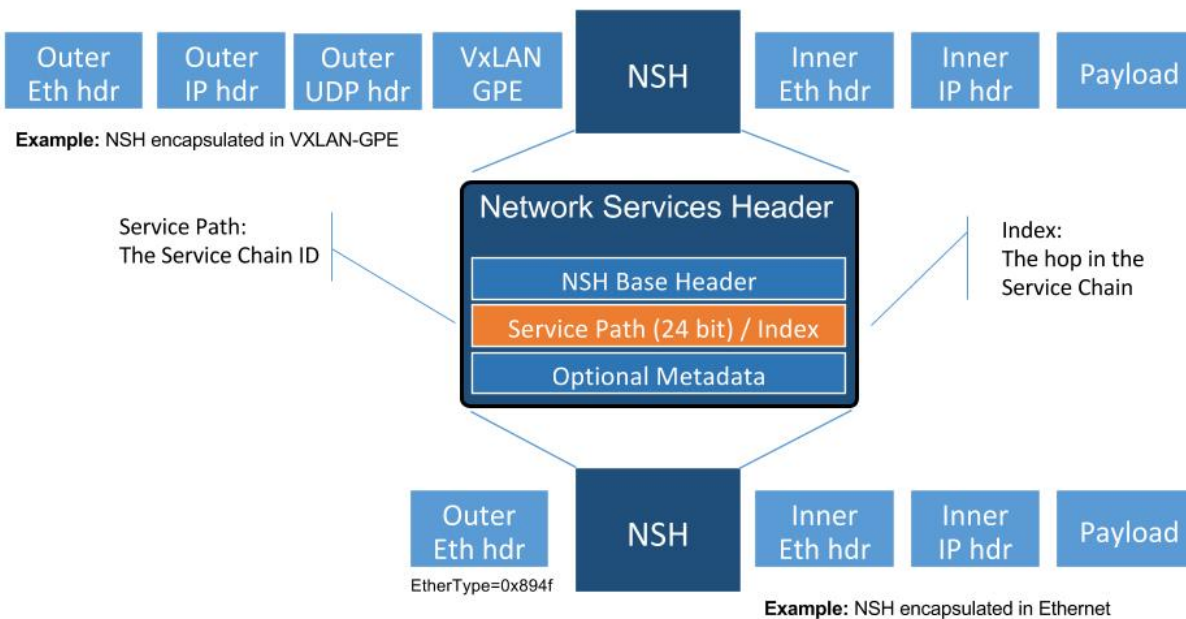


Diagram source [5].

The problem was originally discussed using the slides in this link [12] as a guideline. These slides are only intended for reference, and are not to be used for implementation.

## Use Cases

The main use case addressed by adding an SFC classifier to Netvirt is to integrate SFC with Netvirt, thus allowing for Service Chaining to be used in an OpenStack virtual deployment, such as the OPNFV SFC project [6].

SFC works with both OVS and VPP virtual switches, and its even possible to have a hybrid setup whereby Netvirt is hosted on OVS and SFC is hosted on VPP switches. This blueprint only addresses the use of SFC with NetVirt and OVS.

As mentioned previously, currently SFC works with VXGPE+NSH and Eth+NSH transport/encapsulation, and soon SFC will work with VXLAN as the transport and encapsulation. The first version of this implementation will focus on VXGPE+NSH and Eth+NSH. In the future, when VXLAN is implemented in SFC, VXLAN can be added to the Netvirt SFC classifier. Changes in the transport and encapsulation used for service chains will have no affect on the Netvirt ACL model, since the transport and encapsulation information is obtained via the RSP specified in the RSP.

## Proposed change

The existing old Netvirt SFC code can be found here:

- `netvirt/openstack/net-virt-sfc/{api,impl}`

Once the new Netvirt SFC classifier is implemented and working, the old Netvirt SFC classifier code will be left in place for at least one release cycle.

The new Netvirt SFC code base will be located here:

- `netvirt/vpnservice/sfc/classifier/{api,impl}`

The new Netvirt SFC classifier implementation will be new code. This implementation is not to be confused with the existing Netvirt aclservice, which is implemented for Security Groups. More details about the Genius integration can be found in the following section, but the Netvirt SFC classifier will be in a new Genius classifier service. The SFC implementation is already integrated with Genius and is managed via the Genius SFC service.

## Integration with Genius

Genius [7], [8] is an OpenDaylight project that provides generic infrastructure services to other OpenDaylight projects. New Netvirt makes use of Genius and the new Netvirt classifier will also make use of Genius services. Among these services, the interface manager, tunnel manager and service binding services are of special relevance for the new Netvirt classifier.

Genius interface manager handles an overlay of logical interfaces on top of the data plane physical ports. Based on these logical interfaces, different services/applications may be bound to them with certain priority ensuring that there is no interference between them. Avoiding interference between services/applications is called Application Coexistence in Genius terminology. Typically, the effect of an application binding to a logical interface is that downstream traffic from that interface will be handed off to that application pipeline. Each application is then responsible to either perform a termination action with the packet (i.e output or drop action) or to return the packet back to Genius so that another application can handle the packet. There is a predefined set of types of services that can bind, and Classifier is one of them.

For OpenStack environments, Netvirt registers Neutron ports as logical interfaces in the Genius interface manager. Classifying traffic for a client/tenant ultimately relies on classifying traffic downstream from their corresponding Neutron ports. As such, the Netvirt classifier will bind on these interfaces as a newly defined Genius Classifier service through the Genius interface manager. It was considered integrating the Netvirt classifier with the existing Netvirt security groups, but the idea was discarded due to the possible conflicts and other complications this could cause.

Netvirt also keeps track of the physical location of these Neutron ports in the data plane and updates the corresponding Genius logical interface with this information. Services integrated with Genius may consume this information to be aware of the physical location of a logical interface in the data plane and it's changes when a VM migrates from one location to another. New Netvirt classifier will install the classification rules based on the data plane location of the client/tenant Neutron ports whose traffic is to be classified. On VM migration, the classifier has to remove or modify the corresponding classification rules accounting for this location change, which can be a physical node change or a physical port change.

The classifier is responsible for forwarding packets to the first service function forwarder (SFF) in the chain. This SFF may or may not be on the same compute host as the classifier. If the classifier and SFF are located on the same compute host, then the encapsulated packet is sent to the SFF via the Genius Dispatcher and OpenFlow pipelines. The packets can be forwarded to the SFF locally via the ingress or egress classifier, and it will most likely be performed by the egress classifier, but this decision will be determined at implementation time.

In scenarios where the first SFF is on a different compute host than the client node, the encapsulated packet needs to be forwarded to that SFF through a tunnel port. Tunnels are handled by the Genius tunnel manager (ITM) with an entity called transport zone: all nodes in a transport zone will be connected through a tunnel mesh. Thus the netvirt classifier needs to ensure that the classifier and the SFF are included in a transport zone. The transport type is also specified at the transport zone level and for NSH it needs to be VXGPE. The classifier needs to make sure that this transport zone is handled for location changes of client VMs. Likewise, SFC needs to make sure the transport zone is handled for SF location changes.

The afore-mentioned Genius ITM is different than the tunnels currently used by Netvirt. SFC uses VXGPE tunnels, and requests they be created via the Genius ITM.

## **Classifier and SFC Genius Services**

There will be 2 new Genius services created in Netvirt for the new Netvirt SFC classifier, namely an “Ingress SFC Classifier” and an “Egress SFC Classifier”. There will also be a Genius service for the SFC SFF functionality that has already been created in the SFC project.

The priorities of the services will be as follows:

Ingress Dispatcher:

- SFC - P1
- IngressACL - P2
- Ingress SFC Classifier - P3
- IPv6, IPv4, L2 - P4...

Egress Dispatcher:

- EgressACL - P1
- Egress SFC Classifier - P2

The Ingress SFC classifier will bind on all the Neutron VM ports of the Neutron Network configured in the ACL. All packets received from these Neutron ports will be sent to the Ingress SFC classifier via the Genius Ingress Dispatcher, and will be subjected to ACL matching. If there is no match, then the packets will be returned to the Genius dispatcher so they can be sent down the rest of the Netvirt pipeline. If there is an ACL match, then the classifier will encapsulate NSH, set the NSP and NSI accordingly, initialize C1 and C2 to 0, and send the packet down the rest of the pipeline. Since the SFC service (SFF) will most likely not be bound to this same Neutron port, the packet won't be processed by the SFF on the ingress pipeline. If the classifier and first SFF are in the same node, when the packet is processed by the egress SFC classifier, it will be resubmitted back to the Ingress SFC service (SFC SFF) for SFC processing. If not, the packet will be sent to the first SFF.

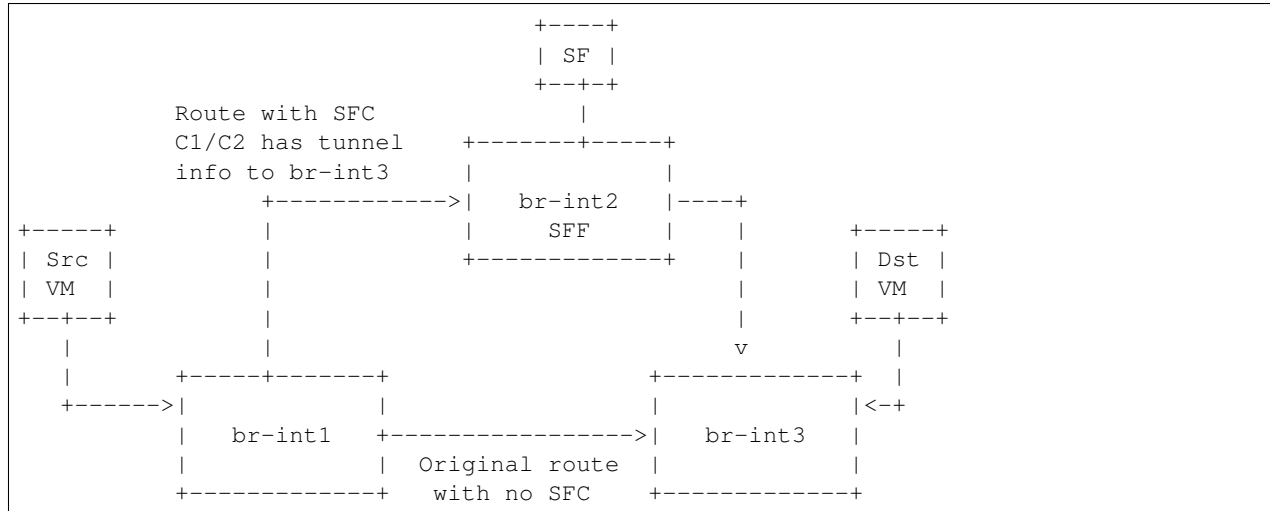
The Ingress SFC service (SFF) will bind on the Neutron ports for the Service Functions and on the VXGPE ports. The Ingress SFC service will receive packets from these Neutron and VXGPE ports, and also those that have been resubmitted from the Egress SFC Classifier. It may be possible that packets received from the SFs are not NSH encapsulated, so any packets received by the Ingress SFC service that are not NSH encapsulated will not be processed and will be sent back to the Ingress Dispatcher. For the NSH packets that are received, the Ingress SFC service will calculate the Next-Hop and modify either the VXGPE header if the next hop is a different SFF, or modify the Ethernet encapsulation header if the next hop is an SF on this same SFF. Once NSH packets are processed by the Ingress SFC service, they will be sent to the Egress Dispatcher.

The Egress SFC classifier service is the final phase of what the Ingress SFC classifier service started when an ACL match happens. The packet needed to go down the rest of the pipeline so the original packet destination can be calculated. The Egress SFC classifier will take the information prepared by the rest of the Netvirt pipeline and store the TunIPv4Dst and VNID of the destination compute host in C1 and C2 respectively. If the packet is not NSH encapsulated, then it will be sent back to the Egress Dispatcher. If the packet does have NSH encapsulation, then if C1/C2 is 0, then the fields will be populated as explained above. If the C1/C2 fields are already set, the packet will be sent out to either the Next Hop SF or SFF.

At the last hop SFF, when the packet egresses the Service Chain, the SFF will pop the NSH encapsulation and use the NSH C1 and C2 fields to tunnel the packet to its destination compute host. If the destination compute host is the same as the last hop SFF, then the packet will be sent down the rest of the Netvirt pipeline so it can be sent to its destination VM on this compute host. When the destination is local, then the inport will probably have to be adjusted.



An example of how the last hop SFF routing works, imagine the following diagram where packet from the Src VM would go from br-int1 to br-int3 to reach the Dst VM when there is no service chaining employed. When the packets from the Src VM are subjected to service chaining, the pipeline in br-int1 need to calculate the the final destination is br-int3, and the appropriate information needs to be set in the NSH C1/C2 fields. Then the SFC SFF on br-int2, upon chain egress will use C1/C2 to send the packets to br-int3 so they can ultimately reach the Dst VM.



## Pipeline changes

The existing Netvirt pipeline will not change as a result of adding the new classifier, other than the fact that the Ingress SFC classifier and Egress SFC classifier Genius Services will be added, which will change the Genius Service priorities as explained previously. The Genius pipelines can be found here [10].

### Ingress Classifier Flows:

The following flows are an approximation of what the Ingress Classifier service pipeline will look like. Notice there are 2 tables defined as follows:

- **table 100: Ingress Classifier Filter table.**
  - Only allows Non-NSH packets to proceed in the classifier
- **table 101: Ingress Classifier ACL table.**
  - Performs the ACL classification, and sends packets to Ingress Dispatcher

The final table numbers may change depending on how they are assigned by Genius.

```

// Pkt has NSH, send back to Ingress Dispatcher
cookie=0xf005ball00000101 table=100, n_packets=11, n_bytes=918,
priority=550,nsp=42 actions=resubmit(,GENIUS_INGRESS_DISPATCHER_TABLE)

// Pkt does NOT have NSH, send to GENIUS_INGRESS_DISPATCHER_TABLE
cookie=0xf005ball00000102 table=100, n_packets=11, n_bytes=918,
priority=5 actions=goto_table:GENIUS_INGRESS_DISPATCHER_TABLE

// ACL match: if TCP port=80
// Action: encapsulate NSH and set NSH NSP, NSI, C1, C2, first SFF
// IP in Reg0, and send back to Ingress Dispatcher to be sent down
// the Netvirt pipeline. The in_port in the match is derived from
// the Neutron Network specified in the ACL match and identifies
// the tenant/Neutron Network the packet originates from

```

```
cookie=0xf005ball00000103, table=101, n_packets=11, n_bytes=918,
tcp,tp_dst=80, in_port=10
actions=push_nsh,
    load:0x1->NXM_NX_NSH_MDTYPE[],
    load:0x0->NXM_NX_NSH_C1[],
    load:0x0->NXM_NX_NSH_C2[],
    load:0x2a->NXM_NX_NSP[0..23],
    load:0xff->NXM_NX_NSI[],
    load:0x0a00010b->NXM_NX_REG0[],
    resubmit(,GENIUS_INGRESS_DISPATCHER_TABLE)
```

### Egress Classifier Flows:

The following flows are an approximation of what the Egress Classifier service pipeline will look like. Notice there are 3 tables defined as follows:

- **table 221: Egress Classifier Filter table.**
  - Only allows NSH packets to proceed in the egress classifier
- **table 222: Egress Classifier NextHop table.**
  - Set C1/C2 accordingly
- **table 223: Egress Classifier TransportEgress table.**
  - Final egress processing and egress packets
  - Determines if the packet should go to a local or remote SFF

The final table numbers may change depending on how they are assigned by Genius.

```
// If pkt has NSH, goto table 222 for more processing
cookie=0x14 table=221, n_packets=11, n_bytes=918,
priority=260,md_type=1
actions=goto_table:222

// Pkt does not have NSH, send back to Egress Dispatcher
cookie=0x14 table=110, n_packets=0, n_bytes=0,
priority=250
actions=resubmit(,GENIUS_EGRESS_DISPATCHER_TABLE)

// Pkt has NSH, if NSH C1/C2 = 0, Set C1/C2 and overwrite TunIpv4Dst
// with SFF IP (Reg0) and send to table 223 for egress
cookie=0x14 table=222, n_packets=11, n_bytes=918,
priority=260,nshc1=0,nshc2=0
actions=load:NXM_NX_TUN_IPV4_DST[]->NXM_NX_NSH_C1[],
    load:NXM_NX_TUN_ID[]->NXM_NX_NSH_C2[],
    load:NXM_NX_REG0[]->NXM_NX_TUN_IPV4_DST[]
goto_table:223

// Pkt has NSH, but NSH C1/C2 already set,
// send to table 223 for egress
cookie=0x14 table=222, n_packets=11, n_bytes=918,
priority=250
actions=goto_table:223

// Checks if the first SFF (IP stored in reg0) is on this node,
// if so resubmit to SFC SFF service
```

```

cookie=0x14 table=223, n_packets=0, n_bytes=0,
  priority=260,nsp=42,reg0=0x0a00010b
  actions=resubmit(, SFF_TRANSPORT_INGRESS_TABLE)

cookie=0x14 table=223, n_packets=0, n_bytes=0,
  priority=250,nsp=42
  actions=outport:6

```

### Ingress SFC Service (SFF) Flows:

The following flows are an approximation of what the Ingress SFC service (SFF) pipeline will look like. Notice there are 3 tables defined as follows:

- **table 83: SFF TransportIngress table.**
  - Only allows NSH packets to proceed into the SFF
- tables 84 and 85 are not used for NSH
- **table 86: SFF NextHop table.**
  - Set the destination of the next SF
- **table 87: SFF TransportEgress table.**
  - Prepare the packet for egress

The final table numbers may change depending on how they are assigned by Genius.

```

// Pkt has NSH, send to table 86 for further processing
cookie=0x14 table=83, n_packets=11, n_bytes=918,
  priority=250,nsp=42
  actions=goto_table:86
// Pkt does NOT have NSH, send back to Ingress Dispatcher
cookie=0x14 table=83, n_packets=0, n_bytes=0,
  priority=5
  actions=resubmit(,GENIUS_INGRESS_DISPATCHER_TABLE)

// Table not used for NSH, shown for completeness
cookie=0x14 table=84, n_packets=0, n_bytes=0,
  priority=250
  actions=goto_table:86

// Table not used for NSH, shown for completeness
cookie=0x14 table=85, n_packets=0, n_bytes=0,
  priority=250
  actions=goto_table:86

// Match on specific NSH NSI/NSP, Encapsulate outer Ethernet
// transport. Send to table 87 for further processing.
cookie=0x14 table=86, n_packets=11, n_bytes=918,
  priority=550,nsi=255,nsp=42
  actions=load:0xb00000c->NXM_NX_TUN_IPV4_DST[],
  goto_table:87
// The rest of the packets are sent to
// table 87 for further processing
cookie=0x14 table=86, n_packets=8, n_bytes=836,
  priority=5
  actions=goto_table:87

// Match on specific NSH NSI/NSP, C1/C2 set

```

```
// prepare pkt for egress, send to Egress Dispatcher
cookie=0xba5eb1100000101 table=87, n_packets=11, n_bytes=918,
priority=650, nsi=255, nsp=42
actions=move:NXM_NX_NSH_MDTYPE[]->NXM_NX_NSH_MDTYPE[],
          move:NXM_NX_NSH_NP[]->NXM_NX_NSH_NP[],
          move:NXM_NX_TUN_ID[0..31]->NXM_NX_TUN_ID[0..31],
          load:0x4->NXM_NX_TUN_GPE_NP[],
          resubmit(, GENIUS_EGRESS_DISPATCHER_TABLE)
```

### Yang changes

The api YANGs used for the classifier build on the ietf acl models from the mdsal models.

Multiple options can be taken, depending on the desired functionality. Depending on the option chosen, YANG changes *might be* required.

Assuming no YANG changes, SFC classification will be performed on all VMs in the same neutron-network - this attribute is already present in the YANG model. **This is the proposed route**, since it hits a sweet-spot in the trade-off between functionality and risk.

If classifying the traffic from specific interfaces is desired, then the YANG model would need to be updated, possibly by adding a list of interfaces on which to classify.

### Configuration impact

None

### Clustering considerations

None

### Other Infra considerations

Since SFC uses NSH, and the new Netvirt Classifier will need to add NSH encapsulation, a version of OVS that supports NSH must be used. NSH has not been officially accepted into the OVS project, so a branched version of OVS is used. Details about the branched version of OVS can be found here [9].

### Security considerations

None

### Scale and Performance Impact

None

### Targeted Release

This change is targeted for the ODL Carbon release.

## Alternatives

None

## Usage

The new Netvirt Classifier will be configured via the REST JSON configuration mentioned in the REST API section below.

## Features to Install

The existing old Netvirt SFC classifier is implemented in the following Karaf feature:

odl-ovsdb-sfc

When the new Netvirt SFC classifier is implemented, the previous Karaf feature will no longer be needed, and the following will be used:

odl-netvirt-sfc

## REST API

The classifier REST API wont change from the old to the new Netvirt. The following example is how the old Netvirt classifier is configured.

Defined in netvirt/openstack/net-virt-sfc/api/src/main/yang/netvirt-acl.yang

An ACL is created which specifies the matching criteria and the action, which is to send the packets to an SFC RSP. Notice the “network-uuid” is set. This is for binding the Netvirt classifier service to a logical port. The procedure will be to query Genius for all the logical ports in that network uuid, and bind the Netvirt classifier service to each of them.

If the RSP has not been created yet, then the classification can not be created, since there wont be any information available about the RSP. In this case, the ACL information will be buffered, and there will be a separate listener for RSPs. When the referenced RSP is created, then the classifier processing will continue.

```
URL: /restconf/config/ietf-access-control-list:access-lists/

{
  "access-lists": {
    "acl": [
      {
        "acl-name": "ACL1",
        "acl-type": "ietf-access-control-list:ipv4-acl",
        "access-list-entries": {
          "ace": [
            {
              "rule-name": "ACE1",
              "actions": {
                "netvirt-sfc-acl:rsp-name": "RSP1"
              },
              "matches": {
                "network-uuid": "eccb57ae-5a2e-467f-823e-45d7bb2a6a9a",
                "source-ipv4-network": "192.168.2.0/24",
                "protocol": "6",
                "source-port-range": {
                  "lower-port": 0
                }
              }
            }
          ]
        }
      }
    ]
  }
}
```

```
        },
        "destination-port-range": {
            "lower-port": 80
        }
    }
}
]
}
}
]]}}
```

## CLI

None.

## Implementation

### Assignee(s)

Primary assignee:

- <brady.allen.johnson@ericsson.com>

Other contributors:

- <brady.allen.johnson@ericsson.com>
- <david.suarez.fuentes@ericsson.com>
- <jaime.camaano.ruiz@ericsson.com>
- <miguel.duarte.de.mora.barroso@ericsson.com>

## Work Items

### Simple scenario:

- Augment the provisioned ACL with the ‘neutron-network’ augmentation - [11]
- From the neutron-network, get a list of neutron-ports - the interfaces connecting the VMs to that particular neutron-network. For each interface, do as follows:
  - Extract the DPN-ID of the node hosting the VM having that neutron-port
  - Extract the DPN-ID of the node hosting the first SF of the RSP
  - The forwarding logic to implement depends on the co-location of the client’s VM with the first SF in the chain.
    - \* When the VMs are co-located (i.e. located in the same host), the output actions are to forward the packet to the first table of the SFC pipeline.
    - \* When the VMs are **not** co-located (i.e. hosted on different nodes) it is necessary to:
      - Use genius RPCs to get the interface connecting 2 DPN-IDs. This will return the tunnel endpoint connecting the compute nodes.
      - Use genius RPCs to get the list of actions to reach the tunnel endpoint.

### Enabling VM mobility:

1. Handle first SF mobility

Listen to RSP updates, where the only relevant migration is when the first SF moves to another node (different DPN-IDs). In this scenario, we delete the flows from the *old* node, and install the newly calculated flows in the new one. This happens for **each** node having an interface to classify attached to the provisioned neutron-network.

2. Handle client VM mobility

Listen to client's InterfaceState changes, re-evaluating the Forwarding logic, since the tunnel interface used to reach the target DPN-ID is different. This means the action list to implement it, will also be different. The interfaces to listen to will be ones attached to the provisioned neutron-network.

3. **Must** keep all the nodes having interfaces to classify (i.e. nodes having neutron-ports attached to the neutron-network) and the first SF host node within the same transport zone. By listening to InterfaceState changes of clients within the neutron-network & the first SF neutron ports, the transport zone rendering can be redone.

**TODO:** *is there a better way to identify when the transport zone needs to be updated?*

## Dependencies

No dependency changes will be introduced by this change.

## Testing

### Unit Tests

Unit tests for the new Netvirt classifier will be modeled on the existing old Netvirt classifier unit tests, and tests will be removed and/or added appropriately.

### Integration Tests

The existing old Netvirt Classifier Integration tests will need to be migrated to use the new Netvirt classifier.

### CSIT

The existing Netvirt CSIT tests for the old classifier will need to be migrated to use the new Netvirt classifier.

## Documentation Impact

User Guide documentation will be added by one of the following contributors:

- <brady.allen.johnson@ericsson.com>
- <david.suarez.fuentes@ericsson.com>
- <jaime.camaano.ruiz@ericsson.com>
- <miguel.duarte.de.mora.barroso@ericsson.com>

## References

- [1] <https://datatracker.ietf.org/doc/draft-ietf-sfc-nsh/>
- [2] <https://datatracker.ietf.org/doc/rfc7665/>
- [3] [https://wiki.opendaylight.org/view/Service\\_Function\\_Chaining:Main](https://wiki.opendaylight.org/view/Service_Function_Chaining:Main)
- [4] <https://datatracker.ietf.org/doc/draft-ietf-nvo3-vxlan-gpe/>
- [5] <https://docs.google.com/presentation/d/1kBY5PKPETEtRA4KRQ-GvVUSLbJoojPsmJlvpKyfZ5dU/edit?usp=sharing>
- [6] <https://wiki.opnfv.org/display/sfc/Service+Function+Chaining+Home>
- [7] <http://docs.opendaylight.org/en/stable-boron/user-guide/genius-user-guide.html>
- [8] [https://wiki.opendaylight.org/view/Genius:Design\\_doc](https://wiki.opendaylight.org/view/Genius:Design_doc)
- [9] [https://wiki.opendaylight.org/view/Service\\_Function\\_Chaining:Main#Building\\_Open\\_vSwitch\\_with\\_VxLAN-GPE\\_and\\_NSH\\_support](https://wiki.opendaylight.org/view/Service_Function_Chaining:Main#Building_Open_vSwitch_with_VxLAN-GPE_and_NSH_support)
- [10] <http://docs.opendaylight.org/en/latest/submodules/genius/docs/pipeline.html>
- [11] <https://github.com/openaylight/netvirt/blob/master/openstack/net-virt-sfc/api/src/main/yang/netvirt-acl.yang>
- [12] <https://docs.google.com/presentation/d/1gN8GnpVGwku4mp1on7EBZiE41RI7lZ-FFmFS2QlUTKk/edit?usp=sharing>

### Table of Contents

- *Netvirt Statistics*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*
  - *Usage*
    - \* *Features to Install*
    - \* *REST API*
    - \* *CLI*
  - *Implementation*



- \* *Assignee(s)*
- \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.18 Netvirt Statistics

<https://git.opendaylight.org/gerrit/#/q/topic:netvirt-counters>

The feature enables getting statistics on ports and switches.

#### Problem description

Being able to ask for statistics, given as input Netvirt identifiers. It will enable filtering the results and having aggregated result. In a later stage, it will be also used to get element to element counters. Examples for possible filters: RX only, TX only, port + VLAN counters...

#### Use Cases

- Getting port counters, given its interface id (ietf interface name).
- Getting node counters, given its node id.

Port counters can be useful also to get statistics on traffic going into tunnels when requesting it from the tunnel endpoint port. In addition, there will also be support in aggregated results. For example: Getting the total number of transmitted packets from a given switch.

#### Proposed change

Adding a new bundle named “statistics-plugin” to Netvirt. This bundle will be responsible for converting the Netvirt unique identifiers into OpenFlow ones, and will get the relevant statistics by using OpenFlowPlugin capabilities. It will also be responsible of validating and filtering the results. It will be able to provide a wide range of aggregated results in the future.

Work flow description: Once a port statistics request is received, it is translated to a port statistics request from openflow plugin. Once the transaction is received, the data is validated and translated to a user friendly data. The user will be notified if a timeout occurs. In case of a request for aggregated counters, the user will receive a single counter result divided to groups (such as “bits”, “packets”...). The counters in each group will be the sum of all of the matching counters for all ports. Neither one of the counter request nor the counter response will not be stored in the configuration database. Moreover, requests are not periodic and they are on demand only.

## Pipeline changes

None

## Yang changes

The new plugin introduced will have the following models:

```
grouping result {
  list counterResult {
    key id;
    leaf id {
      type string;
    }
    list groups {
      key name;
      leaf name {
        type string;
      }
      list counters {
        key name;
        leaf name {
          type string;
        }
        leaf value {
          type uint64;
        }
      }
    }
  }
}

grouping filters {
  leaf-list groupFilters {
    type string;
  }
  leaf-list counterFilter {
    type string;
  }
}

rpc getNodeConnectorCounters {
  input {
    leaf portId {
      type string;
    }
  }
  uses filters;
  output {
    uses result;
  }
}

rpc getNodeCounters {
  input {
    leaf nodeId {
      type uint64;
    }
  }
}
```

```
    }  
  }  
  output {  
    uses result;  
  }  
}  
  
rpc getNodeAggregatedCounters {  
  input {  
    leaf nodeId {  
      type uint64;  
    }  
    uses filters;  
  }  
  output {  
    uses result;  
  }  
}
```

### Configuration impact

None

### Clustering considerations

None

### Other Infra considerations

None

### Security considerations

None

### Scale and Performance Impact

None

### Targeted Release

Carbon

### Alternatives

Getting the statistics from OpenFlow flows: it would be possible to target the appropriate rules in ingress/egress tables, and count the hits on these flows. The reason we decided to work with ports instead is because we don't want to be dependent on flow structure changes.

## Usage

- Create router, network, VMS, VXLAN tunnel.
- Connect to one of the VMs, send ping ping to the other VM.
- Use REST to get the statistics.

Port statistics:

```
http://10.0.77.135:8181/restconf/operational/ietf-interfaces:interfaces-state/
```

Choose a port id and use the following REST in order to get the statistics:

```
10.0.77.135:8181/restconf/operations/statistics-plugin:getNodeConnectorCounters,
↪input={"input":{"portId":"b99a7352-1847-4185-ba24-9ecb4c1793d9"}}, headers=
↪{Authorization=Basic YWRtaW46YWRtaW4=, Cache-Control=no-cache, Content-
↪Type=application/json}}
```

Node statistics:

```
http://10.0.77.135:8181/restconf/config/odl-interface-meta:bridge-interface-info/
```

Choose a node dpId and use the following REST in order to get the statistics:

```
10.0.77.135:8181/restconf/operations/statistics-plugin:getNodeCounters, input=
  {"input": { "portId": "b99a7352-1847-4185-ba24-9ecb4c1793d9", "groups": [{ "name
↪": "byte*",
                                "counters": [{
                                                "name": "rec*",
                                                }, {
                                                "name": "transmitted*",
                                                }]
                                }
  }},
headers={Authorization=Basic YWRtaW46YWRtaW4=, Cache-Control=no-cache, Content-
↪Type=application/json}}
```

```
10.0.77.135:8181/restconf/operations/statistics-plugin:getNodeAggregatedCounters,
↪input=
  {"input": { "portId": "b99a7352-1847-4185-ba24-9ecb4c1793d9", "groups": [{ "name
↪": "byte*",
                                "counters": [{
                                                "name": "rec*",
                                                }, {
                                                "name": "transmitted*",
                                                }]
                                }
  }},
headers={Authorization=Basic YWRtaW46YWRtaW4=, Cache-Control=no-cache, Content-
↪Type=application/json}}
```

Example for a filtered request:

```
10.0.77.135:8181/restconf/operations/statistics-plugin:getPortCounters, input={"input
↪": {"portId":"b99a7352-1847-4185-ba24-9ecb4c1793d9"} }, headers=
↪{Authorization=Basic YWRtaW46YWRtaW4=, Cache-Control=no-cache, Content-
↪Type=application/json}}
```

An example for node connector counters result:

```
{
  "output": {
    "counterResult": [
      {
        "id": "openflow:194097926788804:5",
        "groups": [
          {
            "name": "Duration",
            "counters": [
              {
                "name": "durationNanoSecondCount",
                "value": 471000000
              },
              {
                "name": "durationSecondCount",
                "value": 693554
              }
            ]
          },
          {
            "name": "Bytes",
            "counters": [
              {
                "name": "bytesReceivedCount",
                "value": 1455
              },
              {
                "name": "bytesTransmittedCount",
                "value": 14151299
              }
            ]
          },
          {
            "name": "Packets",
            "counters": [
              {
                "name": "packetsReceivedCount",
                "value": 9
              },
              {
                "name": "packetsTransmittedCount",
                "value": 9
              }
            ]
          }
        ]
      }
    ]
  }
}
```

An example for node counters result:

```
{
  "output": {
    "counterResult": [
      {
```

```
"id": "openflow:194097926788804:3",
"groups": [
  {
    "name": "Duration",
    "counters": [
      {
        "name": "durationNanoSecondCount",
        "value": 43000000
      },
      {
        "name": "durationSecondCount",
        "value": 694674
      }
    ]
  },
  {
    "name": "Bytes",
    "counters": [
      {
        "name": "bytesReceivedCount",
        "value": 0
      },
      {
        "name": "bytesTransmittedCount",
        "value": 648
      }
    ]
  },
  {
    "name": "Packets",
    "counters": [
      {
        "name": "packetsReceivedCount",
        "value": 0
      },
      {
        "name": "packetsTransmittedCount",
        "value": 0
      }
    ]
  }
],
{
  "id": "openflow:194097926788804:2",
  "groups": [
    {
      "name": "Duration",
      "counters": [
        {
          "name": "durationNanoSecondCount",
          "value": 882000000
        },
        {
          "name": "durationSecondCount",
          "value": 698578
        }
      ]
    }
  ]
}
```

```

    },
    {
      "name": "Bytes",
      "counters": [
        {
          "name": "bytesReceivedCount",
          "value": 0
        },
        {
          "name": "bytesTransmittedCount",
          "value": 648
        }
      ]
    },
    {
      "name": "Packets",
      "counters": [
        {
          "name": "packetsReceivedCount",
          "value": 0
        },
        {
          "name": "packetsTransmittedCount",
          "value": 0
        }
      ]
    }
  ],
  {
    "id": "openflow:194097926788804:1",
    "groups": [
      {
        "name": "Duration",
        "counters": [
          {
            "name": "durationNanoSecondCount",
            "value": 978000000
          },
          {
            "name": "durationSecondCount",
            "value": 698627
          }
        ]
      },
      {
        "name": "Bytes",
        "counters": [
          {
            "name": "bytesReceivedCount",
            "value": 6896336558
          },
          {
            "name": "bytesTransmittedCount",
            "value": 161078765
          }
        ]
      }
    ]
  },

```

```
{
  "name": "Packets",
  "counters": [
    {
      "name": "packetsReceivedCount",
      "value": 35644913
    },
    {
      "name": "packetsTransmittedCount",
      "value": 35644913
    }
  ]
},
{
  "id": "openflow:194097926788804:LOCAL",
  "groups": [
    {
      "name": "Duration",
      "counters": [
        {
          "name": "durationNanoSecondCount",
          "value": 339000000
        },
        {
          "name": "durationSecondCount",
          "value": 698628
        }
      ]
    },
    {
      "name": "Bytes",
      "counters": [
        {
          "name": "bytesReceivedCount",
          "value": 0
        },
        {
          "name": "bytesTransmittedCount",
          "value": 0
        }
      ]
    },
    {
      "name": "Packets",
      "counters": [
        {
          "name": "packetsReceivedCount",
          "value": 0
        },
        {
          "name": "packetsTransmittedCount",
          "value": 0
        }
      ]
    }
  ]
}
```



```

    },
    {
      "id": "openflow:194097926788804:5",
      "groups": [
        {
          "name": "Duration",
          "counters": [
            {
              "name": "durationNanoSecondCount",
              "value": 787000000
            },
            {
              "name": "durationSecondCount",
              "value": 693545
            }
          ]
        },
        {
          "name": "Bytes",
          "counters": [
            {
              "name": "bytesReceivedCount",
              "value": 1455
            },
            {
              "name": "bytesTransmittedCount",
              "value": 14151073
            }
          ]
        },
        {
          "name": "Packets",
          "counters": [
            {
              "name": "packetsReceivedCount",
              "value": 9
            },
            {
              "name": "packetsTransmittedCount",
              "value": 9
            }
          ]
        }
      ]
    }
  ]
}

```

## Features to Install

odl-netvirt-openflowplugin-genius-openstack

## REST API

## CLI

## Implementation

## Assignee(s)

**Primary assignee:** Guy Regev <[guy.regev@hpe.com](mailto:guy.regev@hpe.com)>

**Other contributors:** TBD

## Work Items

<https://trello.com/c/ZdoLQWoV/126-netvirt-statistics>

- Support port counters.
- Support node counters.
- Support aggregated results.
- Support filters on results.

## Dependencies

- Genius
- OpenFlow Plugin
- Infrautils

## Testing

Capture details of testing that will need to be added.

## Unit Tests

## Integration Tests

## CSIT

## Documentation Impact

## References

---

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

This work is licensed under a Creative Commons Attribution 3.0 Unported License. <http://creativecommons.org/licenses/by/3.0/legalcode>

---

**Table of Contents**

- *Policy based path selection for multiple VxLAN tunnels*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*
  - *Usage*
    - \* *Features to Install*
    - \* *REST API*
    - \* *CLI*
  - *Implementation*
    - \* *Assignee(s)*
    - \* *Work Items*
  - *Dependencies*
  - *Testing*
    - \* *Unit Tests*
    - \* *Integration Tests*
    - \* *CSIT*
  - *Documentation Impact*
  - *References*

**1.1.19 Policy based path selection for multiple VxLAN tunnels**

<https://git.opendaylight.org/gerrit/#/q/topic:policy-based-path-selection>

The purpose of this feature is to allow selection of primary and backup VxLAN tunnels for different types of VxLAN encapsulated traffic between a pair of OVS nodes based on some predefined policy.

Egress traffic can be classified using different characteristics e.g. 5-tuple, ingress port+VLAN, service-name to determine the best available path when multiple VxLAN endpoints are configured for the same destination.

## Problem description

Today, netvirt is not able to classify traffic and route it over different tunnel endpoints based on a set of predefined characteristics. This is an essential infrastructure for applications on top of netvirt offering premium and personalized services.

## Use Cases

- Forwarding of VxLAN traffic between hypervisors with multiple physical/logical ports.

## Proposed change

The current implementation of transport-zone creation generates vtep elements based on the `local_ip` definition in the `other-config` column of the `Open_vSwitch` schema where the `local_ip` value represents the tunnel interface ip. This feature will introduce a new `other-config` property `local_ips`. `local_ips` will express the association between multiple tunnel ip addresses and multiple underlay networks using the following format:

```
local_ips=<tun1-ip>:<underlay1-net>,<tun2-ip>:<underlay2-net>,...,<tunN-ip>:<underlayN-  
↪net>
```

Upon transport-zone creation, if the `local_ips` configuration is present, full tunnel mesh will be created between all TEP ips in the same underlay network considering the existing transport-zone optimizations i.e. tunnels will be created only between compute nodes with at least one spawned VM in the same VxLAN network or between networks connected to the same router if at least one of the networks is VxLAN-based.

*Note that configuration of multiple tunnel IPs for the same DPN in the same underlay network is not a supported part of this feature and requires further enhancements in both ITM and the transport-zone model.*

The underlay networks are logical entities that will be used to distinguish between multiple uplinks for routing of egress VxLAN traffic. They have no relation to Openstack and neutron networks definition. A new yang module is introduced to model the association between different types of OVS egress VxLAN traffic and the selected underlay network paths to output the traffic.

Policy-based path selection will be defined as a new egress tunnel service and depends on tunnel service binding functionality detailed in [3].

The policy service will be bounded only for tunnels of type logical tunnel group defined in [2].

The service will classify different types of traffic based on a predefined set of policy rules to find the best available path to route each type of traffic. The policy model will be agnostic to the specific topology details including DPN ids, tunnel interface and logical interface names. The only reference from the policy model to the list of preferred paths is made using underlay network-ids described earlier in this document.

Each policy references an ordered set of `policy-routes`. Each `policy-route` can be a `basic-route` referencing single underlay-network or `route-group` composed of multiple underlay networks. This set will get translated in each DPN to OF *fast-failover* group. The content of the buckets in each DPN depends on the existing underlay networks configured as part of the `local_ips` in the specific DPN.

The order of the buckets in the *fast-failover* group depends on the order of the underlay networks in the `policy-routes` model. `policy-routes` with similar set of routes in different order will be translated to different groups.

Each bucket in the *fast-failover* group can either reference a single tunnel or an additional OF *select* group depending on the type of policy route as detailed in the following table:

Policy route type	Bucket actions	OF Watch type
Basic route	load reg6(tun-lport) resubmit(220)	watch_port(tun-port)
Route group	goto_group(select-grp)	watch_group(select-grp)

This OF *select* group does not have the same content as the select groups defined in [2] and the content of its' buckets is based on the defined *route-group* elements and weights.

Logical tunnel will be bounded to the policy service if and only if there is at least one *policy-route* referencing one or more of the underlay networks in the logical group.

This service will take precedence over the default weighted LB service defined in [2] for logical tunnel group interfaces.

Policy-based path selection and weighted LB service pipeline example:

```

cookie=0x6900000, duration=0.802s, table=220, n_packets=0, n_bytes=0, priority=6,
  ↳ reg6=0x500
actions=load:0xe000500->NXM_NX_REG6[], write_metadata:0xe000500000000000/
  ↳ 0xffffffff00000000, goto_table:230
cookie=0x6900000, duration=0.802s, table=220, n_packets=0, n_bytes=0, priority=6,
  ↳ reg6=0xe000500
actions=load:0xf000500->NXM_NX_REG6[], write_metadata:0xf000500000000000/
  ↳ 0xffffffff00000000, group:800002
cookie=0x8000007, duration=0.546s, table=220, n_packets=0, n_bytes=0, priority=7,
  ↳ reg6=0x600 actions=output:3
cookie=0x8000007, duration=0.546s, table=220, n_packets=0, n_bytes=0, priority=7,
  ↳ reg6=0x700 actions=output:4
cookie=0x8000007, duration=0.546s, table=220, n_packets=0, n_bytes=0, priority=7,
  ↳ reg6=0x800 actions=output:5
cookie=0x9000007, duration=0.546s, table=230, n_packets=0, n_bytes=0, priority=7, ip,
metadata=0x222e0/0xffffffffe, nw_dst=10.0.123.2, tp_dst=8080 actions=write_
  ↳ metadata:0x200/0xffffffffe, goto_table:231
cookie=0x9000008, duration=0.546s, table=230, n_packets=0, n_bytes=0, priority=0,
  ↳ resubmit(, 220)
cookie=0x7000007, duration=0.546s, table=231, n_packets=0, n_bytes=0, priority=7,
  ↳ metadata=0x500000000200/0xfffff00ffffffffffe,
actions=group:800000
cookie=0x9000008, duration=0.546s, table=231, n_packets=0, n_bytes=0, priority=0,
  ↳ resubmit(, 220)
group_id=800000, type=ff,
bucket=weight:0, watch_group=800001, actions=group=800001,
bucket=weight:0, watch_port=5, actions=load:0x800->NXM_NX_REG6[], resubmit(, 220)
group_id=800001, type=select,
bucket=weight:50, watch_port=3, actions=load:0x600->NXM_NX_REG6[], resubmit(, 220),
bucket=weight:50, watch_port=4, actions=load:0x700->NXM_NX_REG6[], resubmit(, 220),
group_id=800002, type=select,
bucket=weight:50, watch_port=3, actions=load:0x600->NXM_NX_REG6[], resubmit(, 220),
bucket=weight:25, watch_port=4, actions=load:0x700->NXM_NX_REG6[], resubmit(, 220),
bucket=weight:25, watch_port=5, actions=load:0x800->NXM_NX_REG6[], resubmit(, 220)

```

Each bucket in the *fast-failover* group will set the *watch\_port* or *watch\_group* property to monitor the liveness of the OF port in case of *basic-route* and underlay group in case of *route-group*. This will allow the OVS to route egress traffic only to the first live bucket in each *fast-failover* group.

The policy model rules will be based on IETF ACL data model [4]. The following enhancements are proposed for this model to support policy-based path selection:

	Name	At-tributes	Description	OF implementation
<b>ACE matches</b>	ingress-interface	name	Policy match based on the ingress port and optionally the VLAN id	Match lport-tag metadata bits
		vlan-id		
	service	service-type	Policy match based on the service-name of L2VPN/L3VPN e.g. ELAN name/VPN instance name	Match service/vrf-id metadata bits depending on the service-type
		service-name		
<b>ACE actions</b>	set policy-classifier	policy-classifier	Set ingress/egress classifier that can be later used for policy routing etc. Only the egress classifier will be used in this feature	Set policy classifier in the metadata service bits
		direction		

To enable matching on previous services in the pipeline e.g. L2/L3VPN, the egress service binding for tunnel interfaces will be changed to preserve the metadata of preceding services rather than override it as done in the current implementation.

Each `policy-classifier` will be associated with `policy-route`. The same route can be shared by multiple classifiers.

The policy service will also maintain counters on number of policy rules assigned to underlay network per dpn in the operational DS.

## Pipeline changes

- The following new tables will be added to support the policy-based path selection service:

Table Name	Matches	Actions
Policy classifier table (230)	ACE matches	ACE policy actions: set policy-classifier
Policy routing table (231)	match policy-classifier	set FF group-id

- Each Access List Entry (ACE) composed of standard and/or policy matches and policy actions will be translated to a flow in the policy classifier table.

Each policy-classifier name will be allocated with id from a new pool - `POLICY_SERVICE_POOL`. Once a policy classifier has been determined for a given ACE match, the classifier-id will be set in the `service` bits of the metadata.

- Classified traffic will be sent from the policy classifier table to the policy routing table where the classifier-id will be matched to select the preferred tunnel using OF *fast-failover* group. Multiple classifiers can point to a single group.
- The default flow in the policy tables will resubmit traffic with no predefined policy/set of routes back to the egress dispatcher table in order to continue processing in the next bounded egress service.
- For all the examples below it is assumed that a logical tunnel group was configured for both ingress and egress DPNs. The logical tunnel group is composed of { `tun1`, `tun2`, `tun3` } and bound to a policy service.

## Traffic between VMs on the same DPN

No pipeline changes required

## L3 traffic between VMs on different DPNs

### VM originating the traffic (Ingress DPN):

- Remote next hop group in the FIB table references the logical tunnel group.
- Policy service on the logical group selects the egress interface by classifying the traffic e.g. based on destination ip and port.

```
Classifier table (0) =>
Dispatcher table (17) l3vpn service:  set vpn-id=router-id =>
GW Mac table (19) match:  vpn-id=router-id, dst-mac=router-interface-mac =>
FIB table (21) match:  vpn-id=router-id, dst-ip=vm2-ip set dst-mac=vm2-mac
tun-id=vm2-label reg6=logical-tun-lport-tag =>
Egress table (220) match:  reg6=logical-tun-lport-tag =>
Policy classifier table (230) match:
vpn-id=router-id, dst-ip=vm2-ip, dst-tcp-port=8080 set
egress-classifier=clf1 =>
Egress policy indirection table (231) match:
reg6=logical-tun-lport-tag, egress-classifier=clf1 =>
Logical tunnel tun1 FF group set reg6=tun1-lport-tag =>
Egress table (220) match:  reg6=tun1-lport-tag output to tun1
```

### VM receiving the traffic (Ingress DPN):

- No pipeline changes required

```
Classifier table (0) =>
Internal tunnel Table (36) match: tun-id=vm2-label =>
Local Next-Hop group: set dst-mac=vm2-mac, reg6=vm2-lport-tag =>
Egress table (220) match:  reg6=vm2-lport-tag output to VM 2
```

## SNAT traffic from non-NAPT switch

### VM originating the traffic is non-NAPT switch:

- NAPT group references the logical tunnel group.
- Policy service on the logical group selects the egress interface by classifying the traffic based on the L3VPN service id.

```
Classifier table (0) =>
Dispatcher table (17) l3vpn service:  set vpn-id=router-id =>
GW Mac table (19) match:  vpn-id=router-id, dst-mac=router-interface-mac =>
FIB table (21) match:  vpn-id=router-id =>
Pre SNAT table (26) match:  vpn-id=router-id =>
```

```
NAPT Group set tun-id=router-id reg6=logical-tun-lport-tag =>
Egress table (220) match: reg6=logical-tun-lport-tag =>
Policy classifier table (230) match: vpn-id=router-id set egress-classifier=clf2 =>
Policy routing table (231) match:
reg6=logical-tun-lport-tag, egress-classifier=clf2 =>
Logical tunnel tun2 FF group set reg6=tun2-lport-tag =>
Egress table (220) match: reg6=tun2-lport-tag output to tun2
```

### Traffic from NAPT switch punted to controller:

- No explicit pipeline changes required

```
Classifier table (0) =>
Internal tunnel Table (36) match:tun-id=router-id =>
Outbound NAPT table (46) set vpn-id=router-id, punt-to-controller
```

### L2 unicast traffic between VMs in different DPNs

#### VM originating the traffic (Ingress DPN):

- ELAN DMAC table references the logical tunnel group
- Policy service on the logical group selects the egress interface by classifying the traffic based on the ingress port.

```
Classifier table (0) =>
Dispatcher table (17) l3vpn service: set vpn-id=router-id =>
GW Mac table (19) =>
Dispatcher table (17) l2vpn service: set elan-tag=vxlan-net-tag =>
ELAN base table (48) =>
ELAN SMAC table (50) match: elan-tag=vxlan-net-tag, src-mac=vm1-mac =>
ELAN DMAC table (51) match: elan-tag=vxlan-net-tag, dst-mac=vm2-mac set
tun-id=vm2-lport-tag reg6=logical-tun-lport-tag =>
Egress table (220) match: reg6=logical-tun-lport-tag =>
Policy classifier table (230) match: lport-tag=vm1-lport-tag set
egress-classifier=clf3 =>
Policy routing table (231) match:
reg6=logical-tun-lport-tag, egress-classifier=clf3 =>
Logical tunnel tun1 FF group set reg6=tun1-lport-tag =>
Egress table (220) match: reg6=tun1-lport-tag output to tun1
```

#### VM receiving the traffic (Ingress DPN):

- No explicit pipeline changes required



Classifier table (0) =>  
 Internal tunnel Table (36) match:tun-id=vm2-lport-tag set reg6=vm2-lport-tag =>  
 Egress table (220) match: reg6=vm2-lport-tag output to VM 2

## L2 multicast traffic between VMs in different DPNs with undefined policy

### VM originating the traffic (Ingress DPN):

- ELAN broadcast group references the logical tunnel group.
- Policy service on the logical group has no classification for this type of traffic. Fallback to the default logical tunnel service - weighted LB [2].

Classifier table (0) =>  
 Dispatcher table (17) l3vpn service: set vpn-id=router-id =>  
 GW Mac table (19) =>  
 Dispatcher table (17) l2vpn service: set elan-tag=vxlan-net-tag =>  
 ELAN base table (48) =>  
 ELAN SMAC table (50) match: elan-tag=vxlan-net-tag,src-mac=vm1-mac =>  
 ELAN DMAC table (51) =>  
 ELAN DMAC table (52) match: elan-tag=vxlan-net-tag =>  
 ELAN BC group goto\_group=elan-local-group, set tun-id=vxlan-net-tag  
 reg6=logical-tun-lport-tag =>  
 Egress table (220) match: reg6=logical-tun-lport-tag set  
 reg6=default-egress-service&logical-tun-lport-tag =>  
 Policy classifier table (230) =>  
 Egress table (220) match: reg6=default-egress-service&logical-tun-lport-tag =>  
 Logical tunnel LB select group set reg6=tun2-lport-tag =>  
 Egress table (220) match: reg6=tun2-lport-tag output to tun2

### VM receiving the traffic (Ingress DPN):

- No explicit pipeline changes required

Classifier table (0) =>  
 Internal tunnel Table (36) match:tun-id=vxlan-net-tag =>  
 ELAN local BC group set tun-id=vm2-lport-tag =>  
 ELAN filter equal table (55) match: tun-id=vm2-lport-tag set reg6=vm2-lport-tag =>  
 Egress table (220) match: reg6=vm2-lport-tag output to VM 2

## Yang changes

The following yang modules will be added to support policy-based routing:

## Policy Service Yang

policy-service.yang define policy profiles and add augmentations on top of ietf-access-control-list:access-lists to apply policy classifications on access control entries.

```
module policy-service {
  yang-version 1;
  namespace "urn:opendaylight:netvirt:policy";
  prefix "policy";

  import ietf-interfaces { prefix if; }

  import ietf-access-control-list { prefix ietf-acl; }

  import aclservice { prefix acl; }

  import yang-ext { prefix ext; }

  import opendaylight-l2-types { prefix ethertype; revision-date "2013-08-27"; }

  description
    "Policy Service module";

  revision "2017-02-07" {
    description
      "Initial revision";
  }

  identity policy-acl {
    base ietf-acl:acl-base;
  }

  augment "/ietf-acl:access-lists/ietf-acl:acl/"
  + "ietf-acl:access-list-entries/ietf-acl:ace/ietf-acl:matches" {
    ext:augment-identifier "ingress-interface";
    leaf name {
      type if:interface-ref;
    }

    leaf vlan-id {
      type ethertype:vlan-id;
    }
  }

  augment "/ietf-acl:access-lists/ietf-acl:acl/"
  + "ietf-acl:access-list-entries/ietf-acl:ace/ietf-acl:matches" {
    ext:augment-identifier "service";
    leaf service-type {
      type identityref {
        base service-type-base;
      }
    }

    leaf service-name {
      type string;
    }
  }
}
```

```

augment "/ietf-acl:access-lists/ietf-acl:acl/"
+ "ietf-acl:access-list-entries/ietf-acl:ace/ietf-acl:actions" {
    ext:augment-identifier "set-policy-classifier";
    leaf policy-classifier {
        type leafref {
            path "/policy-profiles/policy-profile/policy-classifier";
        }
    }

    leaf direction {
        type identityref {
            base acl:direction-base;
        }
    }
}

container underlay-networks {
    list underlay-network {
        key "network-name";
        leaf network-name {
            type string;
        }

        leaf network-access-type {
            type identityref {
                base access-network-base;
            }
        }

        leaf bandwidth {
            type uint64;
            description "Maximum bandwidth. Units in byte per second";
        }

        list dpn-to-interface {
            config false;
            key "dp-id";
            leaf dp-id {
                type uint64;
            }

            list tunnel-interface {
                key "interface-name";
                leaf interface-name {
                    type string;
                }
            }
        }

        list policy-profile {
            config false;
            key "policy-classifier";
            leaf policy-classifier {
                type string;
            }
        }
    }
}

```

```

container underlay-network-groups {
    list underlay-network-group {
        key "group-name";
        leaf group-name {
            type string;
        }

        list underlay-network {
            key "network-name";
            leaf network-name {
                type leafref {
                    path "/underlay-networks/underlay-network/network-name";
                }
            }

            leaf weight {
                type uint16;
                default 1;
            }
        }

        leaf bandwidth {
            type uint64;
            description "Maximum bandwidth of the group. Units in byte per second";
        }
    }
}

container policy-profiles {
    list policy-profile {
        key "policy-classifier";
        leaf policy-classifier {
            type string;
        }

        list policy-route {
            key "route-name";
            leaf route-name {
                type string;
            }

            choice route {
                case basic-route {
                    leaf network-name {
                        type leafref {
                            path "/underlay-networks/underlay-network/network-name
↪";
                        }
                    }
                }

                case route-group {
                    leaf group-name {
                        type leafref {
                            path "/underlay-network-groups/underlay-network-group/
↪group-name";
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
  }
}

list policy-acl-rule {
  config false;
  key "acl-name";
  leaf acl-name {
    type leafref {
      path "/ietf-acl:access-lists/ietf-acl:acl/ietf-acl:acl-name";
    }
  }

  list ace-rule {
    key "rule-name";
    leaf rule-name {
      type leafref {
        path "/ietf-acl:access-lists/ietf-acl:acl/ietf-acl:access-
↪list-entries/ietf-acl:ace/ietf-acl:rule-name";
      }
    }
  }
}

}

}

container policy-route-counters {
  config false;

  list underlay-network-counters {
    key "network-name";
    leaf network-name {
      type leafref {
        path "/underlay-networks/underlay-network/network-name";
      }
    }
  }

  list dpn-counters {
    key "dp-id";
    leaf dp-id {
      type uint64;
    }

    leaf counter {
      type uint32;
    }
  }

  list path-counters {
    key "source-dp-id destination-dp-id";
    leaf source-dp-id {
      type uint64;
    }

    leaf destination-dp-id {
      type uint64;
    }
  }
}

```

```
        leaf counter {
            type uint32;
        }
    }
}

identity service-type-base {
    description "Base identity for service type";
}

identity l3vpn-service-type {
    base service-type-base;
}

identity l2vpn-service-type {
    base service-type-base;
}

identity access-network-base {
    description "Base identity for access network type";
}

identity mpls-access-network {
    base access-network-base;
}

identity docsis-access-network {
    base access-network-base;
}

identity pon-access-network {
    base access-network-base;
}

identity dsl-access-network {
    base access-network-base;
}

identity umts-access-network {
    base access-network-base;
}

identity lte-access-network {
    base access-network-base;
}
}
```

### Policy service tree view

```
module: policy-service
+--rw underlay-networks
|   +--rw underlay-network* [network-name]
|       +--rw network-name      string
|       +--rw network-access-type? identityref
```

```

|      +--rw bandwidth?          uint64
|      +--ro dpn-to-interface* [dp-id]
|      |      +--ro dp-id          uint64
|      |      +--ro tunnel-interface*
|      |      |      +--ro interface-name?  string
|      +--ro policy-profile* [policy-classifier]
|      |      +--ro policy-classifier  string
+--rw underlay-network-groups
| +--rw underlay-network-group* [group-name]
| | +--rw group-name          string
| | +--rw underlay-network* [network-name]
| | | +--rw network-name      -> /underlay-networks/underlay-network/network-name
| | | +--rw weight?          uint16
| | +--rw bandwidth?        uint64
+--rw policy-profiles
| +--rw policy-profile* [policy-classifier]
| | +--rw policy-classifier  string
| | +--rw policy-route* [route-name]
| | | +--rw route-name      string
| | | +--rw (route)?
| | | +--:(basic-route)
| | | | +--rw network-name?  -> /underlay-networks/underlay-network/
↪network-name
| | | +--:(route-group)
| | | +--rw group-name?      -> /underlay-network-groups/underlay-network-
↪group/group-name
| | +--ro policy-acl-rule* [acl-name]
| | | +--ro acl-name        -> /ietf-acl:access-lists/acl/acl-name
| | | +--ro ace-rule* [rule-name]
| | | +--ro rule-name       -> /ietf-acl:access-lists/acl/access-list-entries/
↪ace/rule-name
| +--ro policy-route-counters
| | +--ro underlay-network-counters* [network-name]
| | | +--ro network-name      -> /underlay-networks/underlay-network/network-name
| | | +--ro dpn-counters* [dp-id]
| | | | +--ro dp-id          uint64
| | | | +--ro counter?       uint32
| | +--ro path-counters* [source-dp-id destination-dp-id]
| | | +--ro source-dp-id      uint64
| | | +--ro destination-dp-id uint64
| | | +--ro counter?         uint32
augment /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:access-list-entries/ietf-
↪acl:ace/ietf-acl:matches:
| +--rw name?          if:interface-ref
| +--rw vlan-id?       ethernet:vlan-id
augment /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:access-list-entries/ietf-
↪acl:ace/ietf-acl:matches:
| +--rw service-type?  identityref
| +--rw service-name?  string
augment /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:access-list-entries/ietf-
↪acl:ace/ietf-acl:actions:
| +--rw policy-classifier? -> /policy-profiles/policy-profile/policy-classifier
| +--rw direction?        identityref

```

**Configuration impact**

This feature introduces a new `other_config` parameter `local_ips` to support multiple ip:network associations as detailed above. Compatibility with the current `local_ip` parameter will be maintained but if both are present, `local_ips` would take presedence over `local_ip`.

**Clustering considerations**

None

**Other Infra considerations**

None

**Security considerations**

None

**Scale and Performance Impact**

None

**Targeted Release**

Carbon

**Alternatives**

None

**Usage****Features to Install**

odl-netvirt-openstack

**REST API****Sample JSON data**



## Create policy rule

**URL:** restconf/config/ietf-access-control-list:access-lists

The following REST will create rule to classify all http traffic to ports 8080-8181 from specific vpn-id

```
{
  "access-lists": {
    "acl": [
      {
        "acl-type": "policy-service:policy-acl",
        "acl-name": "http-policy",
        "access-list-entries": {
          "ace": [
            {
              "rule-name": "http-ports",
              "matches": {
                "protocol": 6,
                "destination-port-range": {
                  "lower-port": 8080,
                  "upper-port": 8181
                },
                "policy-service:service-type": "l3vpn",
                "policy-service:service-name": "71f7eb47-59bc-4760-8150-
↪e5e408d2ba10"
              },
              "actions": {
                "policy-service:policy-classifier" : "classifier1",
                "policy-service:direction" : "egress"
              }
            }
          ]
        }
      }
    ]
  }
}
```

## Create underlay networks

**URL:** restconf/config/policy-service:underlay-networks

The following REST will create multiple underlay networks with different access types

```
{
  "underlay-networks": {
    "underlay-network": [
      {
        "network-name": "MPLS",
        "network-access-type": "policy-service:mpls-access-network"
      },
      {
        "network-name": "DSL1",
        "network-access-type": "policy-service:dsl-access-network"
      },
      {

```

```
        "network-name": "DSL2",
        "network-access-type": "policy-service:dsl-access-network"
    }
  ]
}
```

## Create underlay group

**URL:** restconf/config/policy-service:underlay-network-groups

The following REST will create group for the DSL underlay networks

```
{
  "underlay-network-groups": {
    "underlay-network-group": [
      {
        "group-name": "DSL",
        "underlay-network": [
          {
            "network-name": "DSL1",
            "weight": 75
          },
          {
            "network-name": "DSL2",
            "weight": 25
          }
        ]
      }
    ]
  }
}
```

## Create policy profile

**URL:** restconf/config/policy-service:policy-profiles

The following REST will create profile for classifier1 with multiple policy-routes

```
{
  "policy-profiles": {
    "policy-profile": [
      {
        "policy-classifier": "classifier1",
        "policy-route": [
          {
            "route-name": "primary",
            "network-name": "MPLS"
          },
          {
            "route-name": "backup",
            "group-name": "DSL"
          }
        ]
      }
    ]
  }
}
```

```

    ]
  }
}
```

## CLI

None

## Implementation

### Assignee(s)

**Primary assignee:** Tali Ben-Meir <tali@hpe.com>

**Other contributors:** Yair Zinger <yair.zinger@hpe.com>

## Work Items

Trello card: <https://trello.com/c/Uk3yrjUG/25-multiple-vxlan-endpoints-for-compute>

- Transport-zone creation for multiple tunnels based on underlay network definitions
- Extract ACL flow programming to common location so it can be used by the policy service
- Create policy OF groups based on underlay network/group definitions
- Create policy classifier table based on ACL rules
- Create policy routing table
- Bind policy service to logical tunnels
- Maintain policy-route-counters per dpn/dpn-path

## Dependencies

None

## Testing

### Unit Tests

### Integration Tests

The test plan defined for CSIT below could be reused for integration tests.

## CSIT

Adding multiple ports to the CSIT setups is challenging due to rackspace limitations. As a result, the test plan defined for this feature uses white-box methodology and not verifying actual traffic was sent over the tunnels.

### Policy routing with single tunnel per access network type

- Set `local_ips` to contain top ips for networks `underlay1` and `underlay2`
- Each underlay network will be defined with different `access-network-type`
- Create the following policy profiles
  - Profile1: `policy-classifier=clf1, policy-routes=underlay1, underlay2`
  - Profile2: `policy-classifier=clf2, policy-routes=underlay2, underlay1`
- Create the following policy rules
  - Policy rule 1: `dst_ip=vm2_ip, dst_port=8080 set_policy_classifier=clf1`
  - Policy rule 2: `src_ip=vm1_ip set_policy_classifier=clf2`
  - Policy rule 3: `service-type=l2vpn service-name=elan-name set_policy_classifier=clf1`
  - Policy rule 4: `service-type=l3vpn service-name=router-name set_policy_classifier=clf2`
  - Policy rule 5: `ingress-port=vm3_port set_policy_classifier=clf1`
  - Policy rule 6: `ingress-port=vm4_port vlan=vlan-id set_policy_classifier=clf2`
- Verify policy service flows/groups for all policy rules
- Verify flows/groups removal after the profiles were deleted

### Policy routing with multiple tunnels per access network type

- Set `local_ips` to contain top ips for networks `underlay1..“underlay4”`
- `underlay1, underlay2 and underlay3, underlay4` are from the same `access-network-type`
- Create the following policy profiles where each route can be either group or basic route
  - Profile1: `policy-classifier=clf1, policy-routes={underlay1, underlay2}, {underlay3, underlay4}`
  - Profile2: `policy-classifier=clf2, policy-routes={underlay3, underlay4}, {underlay1, underlay2}`
  - Profile3: `policy-classifier=clf3, policy-routes=underlay1, {underlay3, underlay4}`
  - Profile4: `policy-classifier=clf4, policy-routes={underlay1, underlay2}, underlay3`
  - Profile5: `policy-classifier=clf5, policy-routes={underlay1, underlay2}`
  - Profile6: `policy-classifier=clf6, policy-routes=underlay4`
- Create the following policy rules
  - Policy rule 1: `dst_ip=vm2_ip, dst_port=8080 set_policy_classifier=clf1`
  - Policy rule 2: `src_ip=vm1_ip set_policy_classifier=clf2`
  - Policy rule 3: `service-type=l2vpn service-name=elan-name set_policy_classifier=clf3`

- Policy rule 4: `service-type=l3vpn service-name=router-name set_policy_classifier=clf4`
- Policy rule 5: `ingress-port=vm3_port set_policy_classifier=clf5`
- Policy rule 6: `ingress-port=vm4_port vlan=vlan-id set_policy_classifier=clf6`
- Verify policy service flows/groups for all policy rules
- Verify flows/groups removal after the profiles were deleted

## Documentation Impact

Netvirt documentation needs to be updated with description and examples of policy service configuration

## References

- [1] [OpenDaylight Documentation Guide](#)
- [2] [Load balancing and high availability of multiple VxLAN tunnels](#)
- [3] [Service Binding On Tunnels](#)
- [4] [Network Access Control List \(ACL\) YANG Data Model](#)

### Table of Contents

- *Support for QoS Alert*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Log file format*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*
  - *Usage*
    - \* *Features to Install*
    - \* *REST API*
    - \* *CLI*
  - *Implementation*

- \* *Assignee(s)*
- \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.20 Support for QoS Alert

<https://git.opendaylight.org/gerrit/#/q/topic:qos-alert>

This feature adds support to monitor the per port packet drop counts when QoS rate limit rule is applied.

#### Problem description

If QoS bandwidth policy is applied on a neutron port, all packets exceeding the rate limit are dropped by the switch. This spec proposes a new service to monitor the packet drop ratio and log the alert message if packet drop ratio is greater than the configured threshold value.

#### Use Cases

Periodically monitor the port statistics of neutron ports having bandwidth limit rule and log an alert message in a log file if packet drop ratio cross the threshold value. Log file can be analyzed offline later to check the health/diagnostics of the network.

#### Proposed change

Proposed new service will use the RPC `/operations/.opendaylight-direct-statistics:get-node-connector-statistics` provided by openflowplugin to retrieve port statistics directly from switch by polling at regular interval. Polling interval is configurable with default value of 2 minutes.

Port packet drop ratio is calculated using delta of two port statistics counters `rx_dropped` and `rx_received` between the sample interval.

```
packet drop ratio = 100 * (rx_dropped / (rx_received + rx_dropped))
```

An message is logged if packet drop ratio is greater than the configured threshold value.

Existing logging framework `log4j` shall be used to log the alert messages in the log file. A new appender `qosalertmsg` shall be added in `org.ops4j.pax.logging.cfg` to define the logging properties.

## Log file format

```

2017-01-17 01:17:49,550 Packet drop threshold hit for qos policy qospolicy1 with qos-
→id qos-2dbf02f6-dcd1-4c13-90ee-6f727e21fe8d for port port-3afde68d-1103-4b8a-a38d-
→9cae631f7d67 on network network-563f9610-dd91-4524-ae23-8ec3c32f328e rx_received_
→4831 rx_dropped 4969
2017-01-17 01:17:49,550 Packet drop threshold hit for qos policy qospolicy2 with qos-
→id qos-cb7e5f67-2552-4d49-b534-0ce90ebc8d97 for port port-09d3a437-f4a4-43eb-8655-
→85df8bbe4793 on network network-389532a1-2b48-4ba9-9bcd-c1705d9e28f9 rx_received_
→3021 rx_dropped 4768
2017-01-17 01:19:49,339 Packet drop threshold hit for qos policy qospolicy1 with qos-
→id qos-2dbf02f6-dcd1-4c13-90ee-6f727e21fe8d for port port-3afde68d-1103-4b8a-a38d-
→9cae631f7d67 on network network-563f9610-dd91-4524-ae23-8ec3c32f328e rx_received_
→3837 rx_dropped 3961
2017-01-17 01:19:49,339 Packet drop threshold hit for qos policy qospolicy2 with qos-
→id qos-cb7e5f67-2552-4d49-b534-0ce90ebc8d97 for port port-09d3a437-f4a4-43eb-8655-
→85df8bbe4793 on network network-389532a1-2b48-4ba9-9bcd-c1705d9e28f9 rx_received_
→2424 rx_dropped 2766

```

## Pipeline changes

None.

## Yang changes

A new yang file shall be created for qos-alert configuration as specified below:

Listing 1.12: qos-alert-config.yang

```

module qosalert-config {

  yang-version 1;
  namespace "urn:opendaylight:params:xml:ns:yang:netvirt:qosalert:config";
  prefix "qosalert";

  revision "2017-01-03" {
    description "Initial revision of qosalert model";
  }

  description "This YANG module defines QoS alert configuration.";

  container qosalert-config {

    config true;

    leaf qos-alert-enabled {
      description "QoS alert enable-disable config knob";
      type boolean;
      default false;
    }

    leaf qos-drop-packet-threshold {
      description "QoS Packet drop threshold config. Specified as % of rx packets";
      type uint8 {
        range "1..100";
      }
    }
  }
}

```

```
    }
    default 5;
  }

  leaf qos-alert-poll-interval {
    description "Polling interval in minutes";
    type uint16 {
      range "1..3600";
    }
    default 2;
  }
}
}
```

## Configuration impact

Following new parameters shall be made available as configuration. Initial or default configuration is specified in `netvirt-qosservice-config.xml`

Sl No.	configuration	Description
1.	<code>qos-alert-enabled</code>	configuration parameter to enable/disable the alerts
2.	<code>qos-drop-packet-threshold</code>	Drop percentage threshold configuration.
3.	<code>qos-alert-poll-interval</code>	Polling interval in minutes

Logging properties like log file name, location, size and maximum number of backup files are configured in file `org.ops4j.pax.logging.cfg`

## Clustering considerations

In cluster setup, only one instance of qosalert service shall poll for port statistics. Entity owner service (EOS) shall be used to determine the owner of service.

## Other Infra considerations

N.A.

## Security considerations

None.

## Scale and Performance Impact

QoS Alert Service minimizes scale and performance impact by following:



- Proposed service uses the direct-statistics RPC instead of OpenflowPlugin statistics-manager. This is lightweight because only node-connector statistics are queried instead of all statistics.
- Polling frequency is quite slow. Default polling interval is **two minutes** and minimum allowed value is 1 minute.

## Targeted Release

Carbon.

## Alternatives

N.A.

## Usage

### Features to Install

This feature can be used by installing `odl-netvirt-openstack`. This feature doesn't add any new karaf feature.

## REST API

### Put Qos Alert Config

Following API puts Qos Alert Config.

**Method:** POST

**URI:** /config/qosalert-config:qosalert-config

**Parameters:**

Parameter	Type	Value range	Comments
qos-alert-enabled	Boolean	true/false	Optional (default false)
qos-drop-packet-threshold	Uint16	1..100	Optional (default 5)
qos-alert-poll-interval	Uint16	1..65535	Optional time interval in minute(s) (default 2)

**Example:** .. code-block:: json

```
{
  "input": {
    "qos-alert-enabled": true,
    "qos-drop-packet-threshold": 35,
    "qos-alert-poll-interval": 5
  }
}
```

## CLI

Following new karaf CLIs are added

```
qos:enable-qos-alert <true|false>
qos:drop-packet-threshold <threshold value in %>
qos:alert-poll-interval <polling interval in minutes>
```

## Implementation

### Assignee(s)

#### Primary assignee:

- Arun Sharma ([arun.e.sharma@ericsson.com](mailto:arun.e.sharma@ericsson.com))

#### Other contributors:

- Ravi Sundareswaran ([ravi.sundareswaran@ericsson.com](mailto:ravi.sundareswaran@ericsson.com))
- Mukta Rani ([mukta.rani@tcs.com](mailto:mukta.rani@tcs.com))

## Work Items

Trello Link <<https://trello.com/c/780v28Yw/148-netvirt-qos-alert>>

1. Adding new yang file and listener.
2. Adding new `log4j` appender in `odlparent org.ops4j.pax.logging.cfg` file.
3. Retrieval of port statistics data using the openflowplugin RPC.
4. Logging alert message into the log file.
5. UT and CSIT

## Dependencies

This doesn't add any new dependencies.

## Testing

Capture details of testing that will need to be added.

## Unit Tests

Standard UTs will be added.

## Integration Tests

N.A.

## CSIT

Following new CSIT tests shall be added

1. Verify that alerts are generated if drop packets percentage is more than the configured threshold value.
2. Verify that alerts are not generated if drop packets percentage is less than threshold value.
3. Verify that alerts are not generated when `qos-alert-enabled` if false irrespective of drop packet percentage.

## Documentation Impact

This will require changes to User Guide.

User Guide will need to add information on how qosalert service can be used.

## References

[1] Neutron QoS

[2] Spec for NetVirt QoS

[3] Openflowplugin port statistics

### Table of Contents

- *Neutron Quality of Service API Enhancements for NetVirt*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*
  - *Usage*
    - \* *Features to Install*
    - \* *REST API*
    - \* *CLI*
  - *Implementation*

- \* *Assignee(s)*
- \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.21 Neutron Quality of Service API Enhancements for NetVirt

QoS patches: <https://git.opendaylight.org/gerrit/#/q/topic:qos>

The Carbon release will enhance the initial implementation of Neutron QoS API<sup>1</sup> support for NetVirt which was released in Boron. The Boron release added support for Neutron QoS policies and the Egress bandwidth rate limiting rule. The Carbon release will update the QoS feature set of NetVirt by providing support for the DSCP Marking rule and QoS Rule capability reporting.

#### Problem description

It is important to be able to configure QoS attributes of workloads on virtual networks. The Neutron QoS API provides a method for defining QoS policies and associated rules which can be applied to Neutron Ports and Networks. These rules include:

- Egress Bandwidth Rate Limiting
- DSCP Marking

(Note that for the Neutron API, the direction of traffic flow (ingress, egress) is from the perspective of the OpenStack instance.)

As a Neutron provider for ODL, NetVirt will provide the ability to report back to Neutron its QoS rule capabilities and provide the ability to configure and manage the supported QoS rules on supported backends (e.g. OVS, ...). The key changes in the Carbon release will be the addition of support for the DSCP Marking rule.

#### Use Cases

Neutron QoS API support, including:

- Egress rate limiting - Drop traffic that exceeds the specified rate parameters for a Neutron Port or Network.
- DSCP Marking - Set the DSCP field for IP packets arriving from Neutron Ports or Networks.
- Reporting of QoS capabilities - Report to Neutron which QoS Rules are supported.

---

<sup>1</sup> Neutron QoS [http://docs.openstack.org/developer/neutron/devref/quality\\_of\\_service.html](http://docs.openstack.org/developer/neutron/devref/quality_of_service.html)

## Proposed change

To handle DSCP marking, listener support will be added to the *neutronvpn* service to respond to changes in DSCP Marking Rules in QoS Policies in the Neutron Northbound QoS models<sup>2,3</sup>.

To implement DSCP marking support, a new ingress (from vswitch perspective) QoS Service is defined in Genius. When DSCP Marking rule changes are detected, a rule in a new OpenFlow table for QoS DSCP marking rules will be updated.

The QoS service will be bound to an interface when a DSCP Marking rule is added and removed when the DSCP Marking rule is deleted. The QoS service follows the DHCP service and precedes the IPV6 service in the sequence of Genius ingress services.

Some use cases for DSCP marking require that the DSCP mark set on the inner packet be replicated to the DSCP marking in the outer packet. Therefore, for packets egressing out of OVS through vxlan/gre tunnels the option to copy the DSCP bits from the inner IP header to the outer IP header is needed. Marking of the inner header is done via OpenFlow rules configured on the corresponding Neutron port as described above. For cases where the outer tunnel header should have a copy of the inner header DSCP marking, the `tos` option on the tunnel interface in OVSDB must be configured to the value `inherit`. The setting of the `tos` option is done with a configurable parameter defined in the ITM module. By default the `tos` option is set to `0` as specified in the OVSDB specification<sup>4</sup>.

On the creation of new tunnels, the `tos` field will be set to either the user provided value or to the default value, which may be controlled via configuration. This will result in the `tunnel-options` field in the IFM (Interface Manager) to be set which will in turn cause the `options` field for the tunnel interface on the OVSDB node to be configured.

To implement QoS rule capability reporting back towards Neutron, code will be added to the *neutronvpn* service to populate the operational `qos-rule-types` list in the Neutron Northbound QoS model<sup>3</sup> with a list of the supported QoS rules - which will be the bandwidth limit rule and DSCP marking rule for the Carbon release.

## Pipeline changes

A new QoS DSCP table is added to support the new QoS Service:

Table	Match	Action
QoS DSCP [90]	Ethtype == IPv4 or IPv6 AND LPort tag	Mark packet with DSCP value

## Yang changes

A new leaf `option-tunnel-tos` is added to `tunnel-end-points` in *itm-state.yang* and to `vsteps` in *itm.yang*.

Listing 1.13: itm-state.yang

```
list tunnel-end-points {
  ordered-by user;
  key "portname VLAN-ID ip-address tunnel-type";

  leaf portname {
    type string;
  }
  leaf VLAN-ID {
    type uint16;
  }
}
```

<sup>2</sup> Neutron Northbound QoS Model Extensions <https://github.com/opendaylight/neutron/blob/master/model/src/main/yang/neutron-qos-ext.yang>

<sup>3</sup> Neutron Northbound QoS Model <https://github.com/opendaylight/neutron/blob/master/model/src/main/yang/neutron-qos.yang>

<sup>4</sup> OVSDB Schema <http://openvswitch.org/ovs-vswitchd.conf.db.5.pdf>

```
leaf ip-address {
    type inet:ip-address;
}
leaf subnet-mask {
    type inet:ip-prefix;
}
leaf gw-ip-address {
    type inet:ip-address;
}
list tz-membership {
    key "zone-name";
    leaf zone-name {
        type string;
    }
}
leaf interface-name {
    type string;
}
leaf tunnel-type {
    type identityref {
        base odlif:tunnel-type-base;
    }
}
leaf option-of-tunnel {
    description "Use flow based tunnels for remote-ip";
    type boolean;
    default false;
}
leaf option-tunnel-tos {
    description "Value of ToS bits to be set on the encapsulating
        packet. The value of 'inherit' will copy the DSCP value
        from inner IPv4 or IPv6 packets. When ToS is given as
        and numeric value, the least significant two bits will
        be ignored. ";
    type string;
}
}
```

Listing 1.14: itm.yang

```
list vsteps {
    key "dpn-id portname";
    leaf dpn-id {
        type uint64;
    }
    leaf portname {
        type string;
    }
    leaf ip-address {
        type inet:ip-address;
    }
    leaf option-of-tunnel {
        description "Use flow based tunnels for remote-ip";
        type boolean;
        default false;
    }
    leaf option-tunnel-tos {
        description "Value of ToS bits to be set on the encapsulating
```

```

        packet. The value of 'inherit' will copy the DSCP value
        from inner IPv4 or IPv6 packets. When ToS is given as
        and numeric value, the least significant two bits will
        be ignored. ";
        type string;
    }
}

```

A configurable parameter `default-tunnel-tos` is added to *itm-config.yang* which defines the default ToS value to be applied to tunnel ports.

Listing 1.15: itm-config.yang

```

container itm-config {
    config true;

    leaf default-tunnel-tos {
        description "Default value of ToS bits to be set on the encapsulating
            packet. The value of 'inherit' will copy the DSCP value
            from inner IPv4 or IPv6 packets. When ToS is given as
            and numeric value, the least significant two bits will
            be ignored. ";
        type string;
        default 0;
    }
}

```

## Configuration impact

A configurable parameter `default-tunnel-tos` is added to *genius-itm-config.xml* which specifies the default ToS to use on a tunnel if it is not specified by the user when a tunnel is created. This value may be set to `inherit` for some DSCP Marking use cases.

Listing 1.16: genius-itm-config.xml

```

<itm-config xmlns="urn:opendaylight:genius:itm:config">
    <default-tunnel-tos>0</default-tunnel-tos>
</itm-config>

```

## Clustering considerations

None.

## Other Infra considerations

None.

## Security considerations

None.

## Scale and Performance Impact

Additional OpenFlow packets will be generated to configure DSCP marking rules in response to QoS Policy changes coming from Neutron.

## Targeted Release

Carbon

## Alternatives

Use of OpenFlow meters was desired, but the OpenvSwitch datapath implementation does not support meters (although the OpenvSwitch OpenFlow protocol implementation does support meters).

## Usage

The user will use the QoS support by enabling and configuring the QoS extension driver for networking-odl. This will allow QoS Policies and Rules to be configured for Neutron Ports and Networks using Neutron.

Perform the following configuration steps:

- In *neutron.conf* enable the QoS service by appending `qos` to the `service_plugins` configuration:

Listing 1.17: `/etc/neutron/neutron.conf`

```
service_plugins = odl-router, qos
```

- Add the QoS notification driver to the *neutron.conf* file as follows:

Listing 1.18: `/etc/neutron/neutron.conf`

```
[qos]
notification_drivers = odl-qos
```

- Enable the QoS extension driver for the core ML2 plugin. In file *ml2.conf.ini* append `qos` to `extension_drivers`

Listing 1.19: `/etc/neutron/plugins/ml2/ml2.conf.ini`

```
[ml2]
extensions_drivers = port_security,qos
```

## Features to Install

Install the ODL Karaf feature for NetVirt (no change):

- `odl-netvirt-openstack`

## REST API

None.



## CLI

Refer to the Neutron CLI Reference<sup>5</sup> for the Neutron CLI command syntax for managing QoS policies and rules for Neutron networks and ports.

## Implementation

### Assignee(s)

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

Primary assignee:

- Poovizhi Pugazh <poovizhi.p@ericsson.com>

Other contributors:

- Ravindra Nath Thakur <ravindra.nath.thakur@ericsson.com>
- Eric Multanen <eric.w.multanen@intel.com>
- Praveen Mala <praveen.mala@intel.com> (including CSIT)

### Work Items

Task list in Carbon Trello: <https://trello.com/c/bLE2n2B1/14-qos>

### Dependencies

Genius project - Code<sup>6</sup> to support QoS Service needs to be added.

Neutron Northbound - provides the Neutron QoS models for policies and rules (already done).

**Following projects currently depend on NetVirt:** Unimgr

### Testing

Capture details of testing that will need to be added.

### Unit Tests

### Integration Tests

### CSIT

### Documentation Impact

Documentation to describe use of Neutron QoS support with NetVirt will be added.

OpenFlow pipeline documentation updated to show QoS service table.

---

<sup>5</sup> Neutron CLI Reference <http://docs.openstack.org/cli-reference/neutron.html#neutron-qos-available-rule-types>

<sup>6</sup> Genius code supporting QoS service <https://git.opendaylight.org/gerrit/#/c/49084/>

## References

<http://specs.openstack.org/openstack/neutron-specs/specs/newton/ml2-qos-with-dscp.html>

ODL gerrit adding QoS models to Neutron Northbound: <https://git.opendaylight.org/gerrit/#/c/37165/>

### Table of Contents

- *Setup Source-MAC-Address for routed packets destined to virtual endpoints*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*
  - *Usage*
    - \* *Features to Install*
    - \* *REST API*
    - \* *CLI*
  - *Implementation*
    - \* *Assignee(s)*
    - \* *Work Items*
  - *Dependencies*
  - *Testing*
    - \* *Unit Tests*
    - \* *Integration Tests*
    - \* *CSIT*
  - *Documentation Impact*
  - *References*

## 1.1.22 Setup Source-MAC-Address for routed packets destined to virtual endpoints

[https://git.opendaylight.org/gerrit/#/q/topic:SMAC\\_virt\\_endpoints](https://git.opendaylight.org/gerrit/#/q/topic:SMAC_virt_endpoints)

All L3 Routed packets destined to virtual endpoints in the datacenter managed by ODL do not carry a proper source-mac address in such frames put out to virtual endpoints.

This spec makes sure a proper source-mac is updated in the packet at the point where the packet is delivered to the VM, regardless of the tenant network type. On the actual datapath, there will be no change in the source mac-addresses and packets continue to use the same mechanism that is used today.

Addressing the datapath requires unique MAC allocation per OVS Datapath, so that it can be used as the source MAC for all distributively routed packets of an ODL enabled cloud. It would be handled in some future spec.

## Problem description

Today all L3 Routed packets destined to virtual endpoints in the datacenter either

- Incorrectly carry the source mac-address of the originator (regardless of which network the originator is in)
- Incorrectly carry sometimes the reserved source mac address of 00:00:00:00:00:00

This spec is intended to setup a source-mac-address in the frame of L3 Routed packets just before such frames are directed into the virtual endpoints themselves. This enables use-cases where certain virtual endpoints which are VNFs in the datacenter that are source-mac conscious (or mandate that src-mac in frames be valid) can become functional on their instantiation in an OpenDaylight enabled cloud.

## Use Cases

- Intra-Datacenter L3 forwarded packets within a hypervisor.
- Intra-Datacenter L3 forwarded packets over Internal VXLAN Tunnels between two hypervisors in the datacenter.
- Inter-Datacenter L3 forwarded packets :
  - Destined to VMs associated floating IP over External VLAN Provider Networks.
  - Destined to VMs associated floating IP over External MPLSOverGRE Tunnels.
  - SNAT traffic from VMs over External MPLSOverGRE Tunnels.
  - SNAT traffic from VMS over External VLAN Provider Networks.

## Proposed change

All the L3 Forwarded traffic today reaches the VM via a LocalNextHopGroup managed by the VPN Engine (including FIBManager).

Currently the LocalNextHopGroup sets-up the destination MAC Address of the VM and forwards the traffic to EGRESS\_LPORT\_DISPATCHER\_TABLE (Table 220). In that LocalNextHopGroup we will additionally setup source-mac-address for the frame. There are two cases to decide what source-mac-address should go into the frame:

- If the VM is on a subnet (on a network) for which a subnet gatewayip port exists, then the source-mac address of that subnet gateway port will be setup as the frame's source-mac inside the LocalNextHop group. This is typical of the case when a subnet is added to a router, as the router interface port created by neutron will be representing the subnet's gateway-ip address.
- If the VM is on a subnet (on a network), for which there is no subnet gatewayip port but that network is part of a BGPVPN, then the source-mac address would be that of the connected mac-address of the VM itself. The connected mac-address is nothing but the mac-address on the ovs-datapath for the VMs tapxxx/vhuxxx port on that hypervisor itself.

The implementation also applies to Extra-Routes (on a router) and Discovered Routes as they both use the LocalNextHopGroup in their last mile to send packets into their NextHop VM.

We need to note that when a network is already part of a BGPVPN, adding a subnet on such a network to a router is disallowed currently by NeutronVPN. And so the need to swap the mac-addresses inside the LocalNextHopGroup to reflect the subnet gatewayip port here does not arise.

For all the use-cases listed in the USE-CASES section above, proper source mac address will be filled-up in the frame before it enters the virtual endpoint.

## Pipeline changes

There are no pipeline changes.

The only change is in the NextHopGroup created by VPN Engine (i.e., VRFEntryListener). In the NextHopGroup we will additionally fill up the ethernet source mac address field with proper mac-address as outlined in the 'Proposed change' section.

Currently the LocalNextHopGroup is used in the following tables of VPN Pipeline:

- L3\_LFIB\_TABLE (Table 20) - Lands all routed packets from MPLSOverGRE tunnel into the virtual endpoint.
- INTERNAL\_TUNNEL\_TABLE (Table 36) - Lands all routed packets on Internal VXLAN Tunnel within the DC into the virtual end point.
- L3\_FIB\_TABLE (Table 21) - Lands all routed packets within a specific hypervisor into the virtual endpoint.

```
cookie=0x80000002, duration=50.676s, table=20, n_packets=0, n_bytes=0, priority=10,
↳mpls, mpls_label=70006 actions=write_actions(pop_mpls:0x0800, group:150000)
cookie=0x80000003, duration=50.676s, table=21, n_packets=0, n_bytes=0, priority=42, ip,
↳metadata=0x222f2/0xffffffff, nw_dst=10.1.1.3 actions=write_actions(group:150000)
cookie=0x9011176, duration=50.676s, table=36, n_packets=0, n_bytes=0, priority=5, tun_
↳id=0x11176 actions=write_actions(group:150000)

NEXTHOP GROUP:
group_id=150000, type=all, bucket=actions=set_field:fa:16:3e:01:1a:40->eth_src, set_
↳field:fa:16:3e:8b:c5:51->eth_dst, load:0x300->NXM_NX_REG6[], resubmit(, 220)
```

## Yang changes

None.

## Configuration impact

None.

## Clustering considerations

None.

## Other Infra considerations

None.

**Security considerations**

None.

**Scale and Performance Impact**

None

**Targeted Release**

Carbon/Boron

**Alternatives**

None.

**Usage**

N/A.

**Features to Install**

odl-netvirt-openstack

**REST API**

N/A.

**CLI**

N/A.

**Implementation****Assignee(s)**

Primary assignee:

- Achuth Maniyedath ([achuth.m@altencalsoftlabs.com](mailto:achuth.m@altencalsoftlabs.com))

Other contributors:

- Karthik Prasad ([karthik.p@altencalsoftlabs.com](mailto:karthik.p@altencalsoftlabs.com))
- Vivekanandan Narasimhan ([n.vivekanandan@ericsson.com](mailto:n.vivekanandan@ericsson.com))

## Work Items

<https://trello.com/c/IfAmnFFr/110-add-source-macs-in-frames-for-l3-routed-packets-before-such-frames-get-to-the-virtual-endpoint>

- Determine the smac address to be used for L3 packets forwarded to VMs.
- Update the LocalNextHopGroup table with proper ethernet source-mac parameter.

## Dependencies

No new dependencies.

## Testing

Verify the Source-MAC-Address setting on frames forwarded to Virtual endpoints in following cases.

Intra-Datacenter traffic to VMs (Intra/Inter subnet).

- VM to VM traffic within a hypervisor.
- VM to VM traffic across hypervisor over Internal VXLAN tunnel.

Inter-Datacenter traffic to/from VMs.

- External access to VMs using Floating IPs on MPLSOverGRE tunnels.
- External access to VMs using Floating IPs over VLAN provider networks.
- External access from VMs using SNAT over VLAN provider networks.
- External access from VMs using SNAT on MPLSOverGRE tunnels.

## Unit Tests

N/A.

## Integration Tests

N/A.

## CSIT

- Validate that router-interface src-mac is available on received frames within the VM when that VM is on a router-arm.
- Validate that connected-mac as src-mac available on received frames within the VM when that VM is on a network-driven L3 BGPVPN.

## Documentation Impact

N/A

## References

N/A

### Table of Contents

- *Support for TCP MD5 Signature Option configuration of Quagga BGP*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Pipeline changes*
    - \* *API changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*
  - *Usage*
    - \* *Features to Install*
    - \* *REST API*
    - \* *CLI*
  - *Implementation*
    - \* *Assignee(s)*
    - \* *Work Items*
  - *Dependencies*
    - \* *Internal*
    - \* *External*
  - *Testing*
    - \* *Unit Tests*
    - \* *Integration Tests*
    - \* *CSIT*
  - *Documentation Impact*
  - *References*

### 1.1.23 Support for TCP MD5 Signature Option configuration of Quagga BGP

<https://git.opendaylight.org/gerrit/#/q/topic:qbgp-tcp-md5-signature-option>

This functionality adds support to odl-netvirt-impl feature to configure the TCP MD5 Signature Option [RFC2385] password in Quagga BGPs [QBG].

#### Problem description

Quagga [QBG] supports TCP MD5 Signature Option [RFC2385] in BGP traffic but current odl-netvirt-impl feature implementation lacks support to configure the required passwords.

#### Use Cases

UC1: Protect (Quagga [QBG]) BGP and DC gateway BGP interface using TCP MD5 Signature Option [RFC2385].

#### Proposed change

The following components need to be enhanced:

- BGP Manager

#### Pipeline changes

No pipeline changes.

#### API changes

Changes will be needed in `ebgp.yang`, and `qbgp.thrift`.

#### YANG changes

A new optional leaf with the TCP MD5 Signature Option [RFC2385] password is added (by means of a choice) to list neighbors.

Listing 1.20: `ebgp.yang` additions

```
typedef tcp-md5-signature-password-type {
  type string {
    length 1..80;
  } // subtype string
  description
    "The shared secret used by TCP MD5 Signature Option. The length is
    limited to 80 chars because A) it is identified by the RFC as current
    practice and B) it is the maximum length accepted by Quagga
    implementation.";
  reference "RFC 2385";
} // typedef tcp-md5-signature-password-type

grouping tcp-security-option-grouping {
```



```

description "TCP security options.";
choice tcp-security-option {
    description "The tcp security option in use, if any.";

    case tcp-md5-signature-option {
        description "The connection uses TCP MD5 Signature Option.";
        reference "RFC 2385";
        leaf tcp-md5-signature-password {
            type tcp-md5-signature-password-type;
            description "The shared secret used to sign the packets.";
        } // leaf tcp-md5-signature-password
    } // case tcp-md5-signature-option

} // choice tcp-security-option
} // grouping tcp-security-option-grouping

```

Listing 1.21: ebgp.yang modifications

```

list neighbors {
    key "address";
    leaf address {
        type inet:ipv4-address;
        mandatory "true";
    }
    leaf remote-as {
        type uint32;
        mandatory "true";
    }
+   use tcp-security-option-grouping;

```

## Thrift changes

A new function `setPeerSecret` is added to the service `BgpConfigurator`.

Listing 1.22: qbgp.thrift modifications

```

--- a/vpnservice/bgpmanager/bgpmanager-impl/src/main/java/org/opendaylight/netvirt/
    ↪bgpmanager/thrift/idl/qbgp.thrift
+++ b/vpnservice/bgpmanager/bgpmanager-impl/src/main/java/org/opendaylight/netvirt/
    ↪bgpmanager/thrift/idl/qbgp.thrift
@@ -31,6 +31,8 @@ const i32 GET_RTS_NEXT = 1
    * ERR_NOT_ITER when GET_RTS_NEXT is called without
    *   initializing with GET_RTS_INIT
    * ERR_PARAM when there is an issue with params
+ * ERR_NOT_SUPPORTED when the server does not support
+ *   the operation.
    */

    const i32 BGP_ERR_FAILED = 1
@@ -38,6 +40,7 @@ const i32 BGP_ERR_ACTIVE = 10
    const i32 BGP_ERR_INACTIVE = 11
    const i32 BGP_ERR_NOT_ITER = 15
    const i32 BGP_ERR_PARAM = 100
+const i32 BGP_ERR_NOT_SUPPORTED = 200

    // these are the supported afi-safi combinations

```

```
enum af_afi {
@@ -122,6 +125,33 @@ service BgpConfigurator {
        6:i32 stalepathTime, 7:bool announceFlush),
    i32 stopBgp(1:i64 asNumber),
    i32 createPeer(1:string ipAddress, 2:i64 asNumber),
+
+ /* 'setPeerSecret' sets the shared secret needed to protect the peer
+  * connection using TCP MD5 Signature Option (see rfc 2385).
+  *
+  * Params:
+  *
+  * 'ipAddress' is the peer (neighbour) address. Mandatory.
+  *
+  * 'rfc2385_sharedSecret' is the secret. Mandatory. Length must be
+  * greater than zero.
+  *
+  * Return codes:
+  *
+  * 0 on success.
+  *
+  * BGP_ERR_FAILED if 'ipAddress' is missing or unknown.
+  *
+  * BGP_ERR_PARAM if 'rfc2385_sharedSecret' is missing or invalid (e.g.
+  * it is too short or too long).
+  *
+  * BGP_ERR_INACTIVE when there is no session.
+  *
+  * BGP_ERR_NOT_SUPPORTED when TCP MD5 Signature Option is not supported
+  * (e.g. the underlying TCP stack does not support it)
+  */
+ i32 setPeerSecret(1:string ipAddress, 2:string rfc2385_sharedSecret),
    i32 deletePeer(1:string ipAddress)
    i32 addVrf(1:layer_type l_type, 2:string rd, 3:list<string> irts, 4:list<string>_
↪erts),
    i32 delVrf(1:string rd),
```

An old server (i.e. using a previous version of `qbgp.thrift`) will return a `TApplicationException` with type `UNKNOWN_METHOD`. See [\[TBaseProcessor\]](#).

## Configuration impact

No configuration parameters deprecated.

New optional leaf `tcp-md5-signature-password` does not impact existing deployments.

The recommended AAA configuration (See [Security considerations](#)) may impact existing deployments.

## Clustering considerations

NA

## Other Infra considerations

### Signature mismatch

On signature mismatch TCP MD5 Signature Option [\[RFC2385\]](#) (page 2) specifies the following behaviour:

Listing 1.23: RFC 2385 page 2

```
Upon receiving a signed segment, the receiver must validate it by
calculating its own digest from the same data (using its own key) and
comparing the two digest. A failing comparison must result in the
segment being dropped and must not produce any response back to the
sender. Logging the failure is probably advisable.
```

A BGP will be unable to connect with a neighbor with a wrong password because the TCP SYN,ACK will be dropped. The neighbor state will bounce between “Active” and “Connect” while it retries.

### Security considerations

tcp-md5-signature-password is stored in clear in the datastore. This is a limitation of the proposed change.

Because tcp-md5-signature-password is stored in clear the REST access to neighbors list should be restricted. See the following AAA configuration examples:

Listing 1.24: etc/shiro.ini example

```
#
# DISCOURAGED since Carbon
#
/config/ebgp:bgp/neighbors/** = authBasic, roles[admin]
```

Listing 1.25: AAA MDSALDynamicAuthorizationFilter example

```
{ "aaa:policies":
  { "aaa:policies": [
    { "aaa:resource": "/restconf/config/ebgp:bgp/neighbors/**",
      "aaa:permissions": [
        { "aaa:role": "admin",
          "aaa:actions": [ "get", "post", "put", "patch", "delete" ]
        }
      ]
    }
  ]
}
```

If BgpConfigurator thrift service is not secured then tcp-md5-signature-password goes clear on the wire.

Quagga [\[QBGp\]](#) (up to version 1.0) keeps the password in memory in clear. The password can be retrieved through Quagga’s configuration interface.

### Scale and Performance Impact

Negligible scale or performance impacts.

- datastore: A bounded (<=80) string per configured neighbor.

- Traffic (thrift `BgpConfigurator` service): A bounded ( $\leq 80$ ) string field per neighbor addition operation.

## Targeted Release

Carbon

## Alternatives

Three alternatives have been considered in order to avoid storing the plain password in datastore: RPC, post-update, and transparent encryption. They are briefly described below.

The best alternative is transparent encryption, but in Carbon time-frame is not feasible.

The post-update alternative does not actually solve the limitation.

The RPC alternative is feasible in Carbon time-frame but, given that currently `BgpConfigurator` thrift service is not secured, to add an RPC does not pull its weight.

## RPC encryption

A new RPC `add-neighbor(address, as-number[, tcp-md5-signature-password])` is in charge of create `neighbors` elements. The password is salted and encrypted with `aaa-encryption-service`. Both the salt and the encrypted password are stored in the `neighbors` element.

## Post-update encryption

The `neighbors` element contains both a `plain-password` leaf and a `encrypted-password-with-salt` leaf. The listener `BgpConfigurationManager.NeighborsReactor` is in charge of encrypt and remove the `plain-password` leaf when it is present (and the encrypted one is not).

This alternative does not really solve the limitation because during a brief period the password is stored in plain.

## Transparent encryption

A plain value is provided in REST write operations but it is *automagically* encrypted before it reaches MD-SAL. Read operations never decrypts the encrypted values.

This alternative impacts at least `aaa`, `yangtools`, and `netconf` projects. It can not possibly be done in Carbon.

## Usage

## Features to Install

odl-netvirt-openstack

## REST API

The RESTful API for neighbors creation (`/restconf/config/ebgp:bgp/neighbors/{address}`) will be enhanced to accept an additional `tcp-md5-signature-password` attribute:

```
{ "neighbors": {
  "address": "192.168.50.2",
  "remote-as": "2791",
  "tcp-md5-signature-password": "password"
}}
```

## CLI

A new option `--tcp-md5-password` will be added to commands `odl:configure-bgp` and `odl:bpn-nbr`.

```
opendaylight-user@root> odl:configure-bgp -op add-neighbor --ip 192.168.50.2 --as-num_
↪2791 --tcp-md5-password password
opendaylight-user@root> odl:bpn-nbr --ip-address 192.168.50.2 --as-number 2791 --tcp-
↪md5-password password add
```

## Implementation

### Assignee(s)

**Primary assignee:** Jose-Santos Pulido, JoseSantos, jose.santos.pulido.garcia@ericsson.com

**Other contributors:** TBD

### Work Items

- <https://trello.com/c/87MAFjRf>
- 1. Spec
- 2. ebgp.yang
- 3. BgpConfigurator thrift service (both idl and client)
- 4. BgpConfigurationManager.NeighborsReactor
- 5. ConfigureBgpCli

## Dependencies

### Internal

No internal dependencies are added or removed.

## External

To enable TCP MD5 Signature Option [\[RFC2385\]](#) in a BGP the following conditions need to be met:

- BgpConfigurator thrift service provider (e.g. Zebra Remote Procedure Call [\[ZRPC\]](#)) must support the new function `setPeerSecret`.
- BGP's TCP stack must support TCP MD5 Signature Option (e.g. in linux the kernel option `CONFIG_TCP_MD5SIG` must be set).

## Testing

### Unit Tests

Currently `bgpmanager` has no unit tests related to configuration.

### Integration Tests

Currently `bgpmanager` has no integration tests.

### CSIT

Currently there is no CSIT test exercising `bgpmanager`.

### Documentation Impact

Currently there is no documentation related to `bgpmanager`.

## References

### Table of Contents

- *Support of VXLAN based L2 connectivity across Datacenters*
  - *Problem description*
    - \* *In scope*
    - \* *Out of scope*
    - \* *Use Cases*
      - *Datacenter access from another Datacenter over WAN via respective DC-Gateways (L2 DCI)*
  - *Proposed change*
    - \* *Pipeline changes*
      - *INTRA DC*
      - *Intra Subnet, Local DPN: VMs on the same subnet, same VPN, same DPN*

- *Intra Subnet, Remote DPN: VMs on two different DPNs, both VMs on the same subnet and same VPN*
- *Inter Subnet, Local DPN: VMs on different subnet, same VPN, same DPN*
- *Inter Subnet, Local DPN: VMs on different subnet, same VPN, same DPN*
- *INTER DC*
- *Intra subnet Traffic from DC-Gateway to Local DPN*
- *Intra subnet Traffic from Local DPN to DC-Gateway*
- *Inter subnet Traffic from Local DPN to DC-Gateway ( Symmetric IRB )*
- *Inter subnet Traffic from DC-Gateway to Local DPN ( Symmetric IRB )*
- *Inter subnet Traffic from Local DPN to DC-Gateway ( ASymmetric IRB )*
- *Intra subnet Traffic from DC-Gateway to Local DPN ( ASymmetric IRB )*
- *ARP Pipeline changes*
- *Local DPN: VMs on the same subnet, same DPN*
- *Intra Subnet, Local DPN: VMs on the same subnet, on remote DC*
- \* *Yang changes*
  - *ODL-L3VPN YANG changes*
  - *ODL-FIB YANG changes*
  - *NEUTRONVPN YANG changes*
  - *ELAN YANG changes*
- \* *Solution considerations*
  - *Proposed change in Openstack Neutron BGPVPN Driver*
  - *Proposed change in BGP Quagga Stack*
  - *Proposed change in OpenDaylight-specific features*
  - *Reboot Scenarios*
- \* *Configuration impact*
- \* *Clustering considerations*
- \* *Other Infra considerations*
- \* *Security considerations*
- \* *Scale and Performance Impact*
- \* *Targeted Release*
- \* *Alternatives*
- *Usage*
  - \* *Features to Install*
  - \* *REST API*
- *Implementation*
  - \* *Assignee(s)*

- \* *Work Items*
  - *Dependencies*
  - *Testing*
    - \* *Unit Tests*
    - \* *Integration Tests*
    - \* *CSIT*
  - *Documentation Impact*
  - *References*

### 1.1.24 Support of VXLAN based L2 connectivity across Datacenters

[https://git.opendaylight.org/gerrit/#/q/topic:EVPN\\_RT2](https://git.opendaylight.org/gerrit/#/q/topic:EVPN_RT2)

Enable realization of L2 connectivity over VXLAN tunnels using L2 BGPVPNs, internally taking advantage of EVPN as the BGP Control Plane mechanism.

#### Problem description

OpenDaylight NetVirt service today supports L3VPN connectivity over VXLAN tunnels. L2DCI communication is not possible so far.

This spec attempts to enhance the BGPVPN service in NetVirt to embrace inter-DC L2 connectivity over external VXLAN tunnels.

#### In scope

The scope primarily includes providing ability to support intra-subnet connectivity across DataCenters over VXLAN tunnels using BGP EVPN with type L2.

When we mention that we are using EVPN BGP Control plane, this spec proposes using the RouteType 2 as the primary means to provision the control plane to enable inter-DC connectivity over external VXLAN tunnels.

With this in place we will be able to support the following.

- Intra-subnet connectivity across dataCenters over VXLAN tunnels.

The following are already supported as part of the other spec(RT5) and will continue to function.

- Intra-subnet connectivity within a DataCenter over VXLAN tunnels.
- Inter-subnet connectivity within a DataCenter over VXLAN tunnels.
- Inter-subnet connectivity across dataCenters over VXLAN tunnels.

#### Out of scope

#### Use Cases

The following high level use-cases will be realized by the implementation of this Spec.



## Datacenter access from another Datacenter over WAN via respective DC-Gateways (L2 DCI)

This use-case involves providing intra-subnet connectivity between two DataCenters. Tenant VMs in one datacenter will be able to communicate with tenant VMs on the other datacenter provided they are part of the same BGP EVPN and they are on same subnets.

The dataplane between the tenant VMs themselves and between the tenant VMs towards the DC-Gateway will be over VXLAN Tunnels.

The dataplane between the DC-Gateway to its other WAN-based BGP Peers is transparent to this spec. It is usually MPLS-based EPVPN.

The BGP Control plane between the ODL Controller and the DC-Gateway will be via EVPN RouteType 2 as defined in EVPN\_RT2.

The control plane between the DC-Gateway and its other BGP Peers in the WAN is transparent to this spec, but can be EVPN IP-MPLS.

In this use-case:

1. We will have only a single DCGW for WAN connectivity
2. MAC IP prefix exchange between ODL controller and DC-GW (iBGP) using EVPN RT2
3. WAN control plane may use EVPN IP-MPLS for route exchange.
4. On the DC-Gateway, the VRF instance will be configured with two sets of import/export targets. One set of import/export route targets belong to EVPN inside DataCenter (realized using EVPN RT2) and the second set of import/export route target belongs to WAN control plane.
5. EVPN single homing to be used in all RT2 exchanges inside the DataCenter i.e., ESI=0 for all prefixes sent from DataCenter to the DC-Gateway.

## Proposed change

The following components of an Openstack-ODL-based solution need to be enhanced to provide intra-subnet and inter-subnet connectivity across DCs using EVPN MAC IP Advertisement (Route Type 2) mechanism (refer EVPN\_RT2):

- Openstack Neutron BGPVPN Driver
- OpenDaylight Controller (NetVirt)
- BGP Quagga Stack to support EVPN with RouteType 2 NLRI
- DC-Gateway BGP Neighbour that supports EVPN with RouteType 2 NLRI

The changes required in Openstack Neutron BGPVPN Driver and BGP Quagga Stack are captured in the Solution considerations section down below.

## Pipeline changes

### INTRA DC

#### Intra Subnet, Local DPN: VMs on the same subnet, same VPN, same DPN

There are no explicit pipeline changes for this use-case.

**Intra Subnet, Remote DPN: VMs on two different DPNs, both VMs on the same subnet and same VPN**

There are no explicit pipeline changes for this use-case.

**Inter Subnet, Local DPN: VMs on different subnet, same VPN, same DPN**

There are no explicit pipeline changes for this use-case.

**Inter Subnet, Local DPN: VMs on different subnet, same VPN, same DPN**

There are no explicit pipeline changes for this use-case.

**INTER DC****Intra subnet Traffic from DC-Gateway to Local DPN**

```
Classifier table (0) =>
Dispatcher table (17) match:  tunnel-type=vxlan =>
L2VNI_EXTERNAL_TUNNEL_DEMUX_TABLE (24) => match tunnel-id=l2vni, set
elan-tag
ELAN DMAC table (51) match:  elan-tag=vxlan-net-tag, dst-mac=vm2-mac set
reg6=vm-lport-tag =>
Egress table (220) match:  reg6=vm-lport-tag output to vm port
```

**Intra subnet Traffic from Local DPN to DC-Gateway**

```
Classifier table (0) =>
Dispatcher table (17) l3vpn service:  set vpn-id=router-id =>
GW Mac table (19) =>
Dispatcher table (17) l2vpn service:  set elan-tag=vxlan-net-tag =>
ELAN base table (48) =>
ELAN SMAC table (50) match:  elan-tag=vxlan-net-tag, src-mac=vm1-mac =>
ELAN DMAC table (51) match:
elan-tag=vxlan-net-tag, dst-mac=external-vm-mac set
tun-id=vxlan-net-tag group=next-hop-group
Next Hop Group bucket0 :set reg6=tunnel-lport-tag bucket1 :set
reg6=tunnel2-lport-tag
Egress table (220) match:  reg6=tunnel2-lport-tag output to tunnel2
```

**Inter subnet Traffic from Local DPN to DC-Gateway ( Symmetric IRB )**

```
Classifier Table (0) =>
Dispatcher table (17) l3vpn service:  set vpn-id=router-id =>
L3 Gateway MAC Table (19) match:  vpn-id=l3vpn-id,
dst-mac=vpn-subnet-gateway-mac-address =>
```

L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set  
 tun-id=l3vni output to nexthopgroup =>  
 NextHopGroup: set-eth-dst router-gw-vm, reg6=tunnel-lport-tag =>  
 Lport Egress Table (220) Output to tunnel port

### Inter subnet Traffic from DC-Gateway to Local DPN ( Symmetric IRB )

Classifier table (0) =>  
 Dispatcher table (17) match: tunnel-type=vxlan =>  
 L3VNI\_EXTERNAL\_TUNNEL\_DEMUX\_TABLE (23) => match tunnel-id=l3vni, set  
 l3vpn-id =>  
 L3 Gateway MAC Table (19) => match dst-mac=vpn-subnet-gateway-mac-address =>  
 FIB table (21) match: l3vpn-tag=l3vpn-id, dst-ip=vm2-ip set  
 reg6=vm-lport-tag goto=local-nexthop-group =>  
 local nexthop group set dst-mac=vm2-mac table=220 =>  
 Egress table (220) match: reg6=vm-lport-tag output to vm port

### Inter subnet Traffic from Local DPN to DC-Gateway ( ASymmetric IRB )

Classifier Table (0) =>  
 Dispatcher table (17) l3vpn service: set vpn-id=router-id =>  
 L3 Gateway MAC Table (19) match: vpn-id=l3vpn-id,  
 dst-mac=vpn-subnet-gateway-mac-address =>  
 L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set  
 tun-id=l2vni output to nexthopgroup =>  
 NextHopGroup: set-eth-dst dst-vm-mac, reg6=tunnel-lport-tag =>  
 Lport Egress Table (220) Output to tunnel port

### Intra subnet Traffic from DC-Gateway to Local DPN ( ASymmetric IRB )

Classifier table (0) =>  
 Dispatcher table (17) match: tunnel-type=vxlan =>  
 L2VNI\_EXTERNAL\_TUNNEL\_DEMUX\_TABLE (24) => match tunnel-id=l2vni, set  
 elan-tag  
 ELAN DMAC table (51) match: elan-tag=vxlan-net-tag, dst-mac=vm2-mac set  
 reg6=vm-lport-tag =>  
 Egress table (220) match: reg6=vm-lport-tag output to vm port

## ARP Pipeline changes

### Local DPN: VMs on the same subnet, same DPN

a. Introducing a new Table aka ELAN\_ARP\_SERVICE\_TABLE (Table 81). This table will be the first table in elan pipeline.

Classifier table (0) =>  
 Dispatcher table (17) elan service: set elan-id=vxlan-net-tag =>

```
Arp Service table (81) => match:  arp-op=req, dst-ip=vm-ip,  
ela-id=vxlan-net-tag inline arp reply
```

### Intra Subnet, Local DPN: VMs on the same subnet, on remote DC

```
Classifier table (0) =>  
Dispatcher table (17) elan service:  set elan-id=vxlan-net-tag =>  
Arp Service table (81) => match:  arp-op=req, dst-ip=vm-ip,  
ela-id=vxlan-net-tag inline arp reply
```

### Yang changes

Changes will be needed in `l3vpn.yang`, `odl-l3vpn.yang`, `odl-fib.yang` and `neutronvpn.yang` to start supporting EVPN functionality.

### ODL-L3VPN YANG changes

A new container `evpn-rd-to-networks` is added This holds the rd to networks mapping This will be useful to extract in which elan the received RT2 route can be injected into.

Listing 1.26: `odl-l3vpn.yang`

```
container evpn-rd-to-networks {  
  config false;  
  description "Holds the networks to which given evpn is attached to";  
  list evpn-rd-to-network {  
    key rd;  
    leaf rd {  
      type string;  
    }  
    list evpn-networks {  
      key network-id;  
      leaf network-id {  
        type string;  
      }  
    }  
  }  
}
```

### ODL-FIB YANG changes

A new field `macVrfEntries` is added to the container `fibEntries` This holds the RT2 routes received for the given rd

Listing 1.27: `odl-fib.yang`

```
grouping vrfEntryBase {  
  list vrfEntry {  
    key "destPrefix";  
    leaf destPrefix {  
      type string;  
      mandatory true;  
    }  
  }  
}
```

```

    }
    leaf origin {
        type string;
        mandatory true;
    }
    leaf encap-type {
        type enumeration {
            enum mplsgre {
                value "0";
                description "MPLSOverGRE";
            }
            enum vxlan {
                value "1";
                description "VNI";
            }
        }
        default "mplsgre";
    }
    leaf l3vni {
        type uint32;
    }
    list route-paths {
        key "nexthop-address";
        leaf nexthop-address {
            type string;
        }
        leaf label {
            type uint32;
        }
        leaf gateway_mac_address {
            type string;
        }
    }
}

grouping vrfEntries{
    list vrfEntry{
        key "destPrefix";
        uses vrfEntryBase;
    }
}

grouping macVrfEntries{
    list MacVrfEntry {
        key "mac_address";
        uses vrfEntryBase;
        leaf l2vni {
            type uint32;
        }
    }
}

container fibEntries {
    config true;
    list vrfTables {
        key "routeDistinguisher";
        leaf routeDistinguisher {type string;}
    }
}

```

```
    uses vrfEntries;
    uses macVrfEntries;//new field
  }
  container ipv4Table{
    uses ipv4Entries;
  }
}
```

## NEUTRONVPN YANG changes

A new rpc `createEVPN` is added Existing rpc `associateNetworks` is reused to attach a network to EVPN assuming `uuid` of L3VPN and EVPN does not collide with each other.

Listing 1.28: neutronvpn.yang

```
rpc createEVPN {
  description "Create one or more EVPN(s)";
  input {
    list evpn {
      uses evpn-instance;
    }
  }
  output {
    leaf-list response {
      type string;
      description "Status response for createVPN RPC";
    }
  }
}

rpc deleteEVPN{
  description "delete EVPNs for specified Id list";
  input {
    leaf-list id {
      type yang:uuid;
      description "evpn-id";
    }
  }
  output {
    leaf-list response {
      type string;
      description "Status response for deleteEVPN RPC";
    }
  }
}

grouping evpn-instance {

  leaf id {
    mandatory "true";
    type yang:uuid;
    description "evpn-id";
  }

  leaf name {
    type string;
  }
}
```

```

    description "EVPN name";
  }

  leaf tenant-id {
    type yang:uuid;
    description "The UUID of the tenant that will own the subnet.";
  }

  leaf-list route-distinguisher {
    type string;
    description
      "configures a route distinguisher (RD) for the EVPN instance.
      Format is ASN:nn or IP-address:nn.";
  }

  leaf-list import-RT {
    type string;
    description
      "configures a list of import route target.
      Format is ASN:nn or IP-address:nn.";
  }

  leaf-list export-RT{
    type string;
    description
      "configures a list of export route targets.
      Format is ASN:nn or IP-address:nn.";
  }

  leaf l2vni {
    type uint32;
  }
}

```

## ELAN YANG changes

Existing container `elan`-instances is augmented with `evpn` information.

A new list `external-teps` is added to `elan` container. This captures the broadcast domain of the given network/`elan`. When the first RT2 route is received from the dc gw, it's tep ip is added to the `elan` to which this RT2 route belongs to.

Listing 1.29: `elan.yang`

```

augment "/elan:elan-instances/elan:elan-instance" {
  ext:augment-identifier "evpn";
  leaf evpn-name {
    type string;
  }
  leaf l3vpn-name {
    type string;
  }
}

container elan-instances {
  list elan-instance {

```

```
        key "elan-instance-name";
        leaf elan-instance-name {
            type string;
        }
        //omitted other existing fields
        list external-teps {
            key tep-ip;
            leaf tep-ip {
                type inet:ip-address;
            }
        }
    }
}

container elan-interfaces {
    list elan-interface {
        key "name";
        leaf name {
            type leafref {
                path "/if:interfaces/if:interface/if:name";
            }
        }
        leaf elan-instance-name {
            mandatory true;
            type string;
        }
        list static-mac-entries {
            key "mac";
            leaf mac {
                type yang:phys-address;
            }
            leaf prefix { //new field
                mandatory false;
                type inet:ip-address;
            }
        }
    }
}

grouping forwarding-entries {
    list mac-entry {
        key "mac-address";
        leaf mac-address {
            type yang:phys-address;
        }
        leaf interface {
            type leafref {
                path "/if:interfaces/if:interface/if:name";
            }
        }
        leaf controllerLearnedForwardingEntryTimestamp {
            type uint64;
        }
        leaf isStaticAddress {
            type boolean;
        }
        leaf prefix { //new field
            mandatory false;
        }
    }
}
```



```

        type inet:ip-address;
    }
}

```

## Solution considerations

### Proposed change in Openstack Neutron BGPVPN Driver

The Openstack Neutron BGPVPN's ODL driver in Newton release is changed (mitaka release), so that it is able to relay the configured L2 BGPVPNs, to the OpenDaylight Controller.

The Newton changes for the BGPVPN Driver has merged and is here: <https://review.openstack.org/#/c/370547/>

### Proposed change in BGP Quagga Stack

The BGP Quagga Stack is a component that interfaces with ODL Controller to enable ODL Controller itself to become a BGP Peer. This BGP Quagga Stack need to be enhanced so that it is able to embrace EVPN with Route Type 5 on the following two interfaces:

- Thrift Interface where ODL pushes routes to BGP Quagga Stack
- Route exchanges from BGP Quagga Stack to other BGP Neighbors (including DC-GW).

### Proposed change in OpenDaylight-specific features

The following components within OpenDaylight Controller needs to be enhanced:

- NeutronvpnManager
- VPN Engine (VPN Manager)
- ELAN Manager
- FIB Manager
- BGP Manager

## Reboot Scenarios

This feature support all the following Reboot Scenarios for EVPN:

- Entire Cluster Reboot
- Leader PL reboot
- Candidate PL reboot
- OVS Datapath reboots
- Multiple PL reboots
- Multiple Cluster reboots
- Multiple reboots of the same OVS Datapath.
- Openstack Controller reboots

## Configuration impact

The following parameters have been initially made available as configurable for EVPN. These configurations can be made via the RESTful interface:

**1.Multi-homing-mode** – For multi-homing use cases where redundant DCGWs are used ODL can be configured with ‘none’, ‘all-active’ or ‘single-active’ multi-homing mode. Default will be ‘none’.

**2.IRB-mode** – Depending upon the support on DCGW, ODL can be configured with either ‘Symmetric’ or ‘Asymmetric’ IRB mode. Default is ‘Symmetric’.

There is another important parameter though it won’t be configurable:

**MAC Address Prefix for EVPN** – This MAC Address prefix represents the MAC Address prefix that will be hard-coded and that MACAddress will be used as the gateway mac address if it is not supplied from Openstack. This will usually be the case when networks are associated to an L3VPN with no gateway port yet configured in Openstack for such networks.

## Clustering considerations

The feature should operate in ODL Clustered environment reliably.

## Other Infra considerations

N.A.

## Security considerations

N.A.

## Scale and Performance Impact

Not covered by this Design Document.

## Targeted Release

Carbon.

## Alternatives

Alternatives considered and why they were not selected.

## Usage

## Features to Install

This feature can be used by installing odl-netvirt-openstack. This feature doesn’t add any new karaf feature.

## REST API

A new rpc is added to create and delete evpn:

```
{'input': {
  'evpn': [
    {'name': 'EVPN1',
     'export-RT': ['50:2'],
     'route-distinguisher': ['50:2'],
     'import-RT': ['50:2'],
     'id': '4ae8cd92-48ca-49b5-94e1-b2921a260007',
     'l2vni': '200',
     'tenant-id': 'a565b3ed854247f795c0840b0481c699'}
  ]
}}
```

There is no change in the REST API for associating networks to the EVPN.

On the Openstack-side configuration, the vni\_ranges configured in Openstack Neutron ml2\_conf.ini should not overlap with the L3VNI provided in the ODL RESTful API. In an inter-DC case, where both the DCs are managed by two different Openstack Controller Instances, the workflow will be to do the following:

1. Configure the DC-GW2 facing OSC2 (Openstack) and DC-GW1 facing OSC1 with the same BGP configuration parameters.
2. On first Openstack Controller (OSC1) create an L3VPN1 with RD1 and L3VNI1
3. On first Openstack Controller (OSC1) create an EVPN1 with RD2 and L2VNI1
4. Create a network Net1 and Associate that Network Net1 to L3VPN1
5. Create a network Net1 and Associate that Network Net1 to EVPN1
6. On second Openstack Controller (OSC2) create an L3VPN2 with RD1 with L3VNI1
7. On second Openstack Controller (OSC2) create an EVPN2 with RD2 with L2VNI1
8. Create a network Net2 on OSC2 with same cidr as the first one with a different allocation pool and associate that Network Net2 to L3VPN2.
9. Associate that Network Net2 to EVPN2.
10. Spin-off VM1 on Net1 in OSC1.
11. Spin-off VM2 on Net2 in OSC2.
12. Now VM1 and VM2 should be able to communicate.

## Implementation

### Assignee(s)

**Primary assignee:** Vyshakh Krishnan C H <vyshakh.krishnan.c.h@ericsson.com>

Yugandhar Reddy Kaku <yugandhar.reddy.kaku@ericsson.com>

Riyazahmed D Talikoti <riyazahmed.d.talikoti@ericsson.com>

**Other contributors:** K.V Suneelu Verma <k.v.suneelu.verma@ericsson.com>

## Work Items

Trello card details <https://trello.com/c/PysPZscm/150-evpn-evpn-rt2>.

## Dependencies

Requires a DC-GW that is supporting EVPN RT2 on BGP Control plane.

## Testing

Capture details of testing that will need to be added.

## Unit Tests

Appropriate UTs will be added for the new code coming in once framework is in place.

## Integration Tests

There won't be any Integration tests provided for this feature.

## CSIT

CSIT will be enhanced to cover this feature by providing new CSIT tests.

## Documentation Impact

This will require changes to User Guide and Developer Guide.

## References

- [1] [EVPN\\_RT5](#)
- [2] [Network Virtualization using EVPN](#)
- [3] [Integrated Routing and Bridging in EVPN](#)
- [4] [VXLAN DCI using EVPN](#)
- [5] [BGP MPLS-Based Ethernet VPN](#)
- [6] [Trello card details](#)

### Table of Contents

- *Support of VXLAN based connectivity across Datacenters*
  - *Problem description*
    - \* *In scope*
    - \* *Out of scope*

- \* *Use Cases*

- *DataCenter access from a WAN-client via DC-Gateway (Single Homing)*
- *Datacenter access from another Datacenter over WAN via respective DC-Gateways (L3 DCI)*

- *Proposed change*

- \* *Pipeline changes*

- *INTRA DC*
- *Intra Subnet, Local DPN: VMs on the same subnet, same VPN, same DPN*
- *Intra Subnet, Remote DPN: VMs on two different DPNs, both VMs on the same subnet and same VPN*
- *Inter Subnet, Local DPN: VMs on different subnet, same VPN, same DPN*
- *Inter Subnet, Remote DPN: VMs on two different DPNs, both VMs on different subnet, but same VPN*
- *INTER DC*
- *Intra Subnet*
- *Inter Subnet*
- *SNAT pipeline (Access to External Network Access over VXLAN)*
- *DNAT pipeline (Access from External Network over VXLAN)*

- \* *Yang changes*

- *L3VPN YANG changes*
- *ODL-L3VPN YANG changes*
- *ODL-FIB YANG changes*
- *NEUTRONVPN YANG changes*

- \* *Solution considerations*

- *Proposed change in Openstack Neutron BGPVPN Driver*
- *Proposed change in BGP Quagga Stack*
- *Proposed change in OpenDaylight-specific features*
- *Import Export RT support for EVPN*
- *SubnetRoute support on EVPN*
- *NAT Service support for EVPN*
- *ARP request/response and MIP handling Support for EVPN*
- *Tunnel state handling Support*
- *InterVPNLink support for EVPN*
- *Supporting VLAN Aware VMs (Trunk and SubPorts)*
- *VM Mobility with RT5*
- *Reboot Scenarios*

- \* *Configuration impact*

- \* *Clustering considerations*
- \* *Other Infra considerations*
- \* *Security considerations*
- \* *Scale and Performance Impact*
- \* *Targeted Release*
- \* *Alternatives*
- *Usage*
  - \* *Features to Install*
  - \* *REST API*
- *Implementation*
  - \* *Assignee(s)*
  - \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.25 Support of VXLAN based connectivity across Datacenters

[https://git.opendaylight.org/gerrit/#/q/topic:EVPN\\_RT5](https://git.opendaylight.org/gerrit/#/q/topic:EVPN_RT5)

Enable realization of L3 connectivity over VXLAN tunnels using L3 BGPVPNs, internally taking advantage of EVPN as the BGP Control Plane mechanism.

#### Problem description

OpenDaylight NetVirt service today supports VLAN-based, VXLAN-based connectivity and MPLSOverGRE-based overlays.

In this VXLAN-based underlay is supported only for traffic within the DataCenter. For all the traffic that need to go via the DC-Gateway the only supported underlay is MPLSOverGRE.

Though there is a way to provision an external VXLAN tunnel via the ITM service in Genius, the BGPVPN service in NetVirt does not have the ability to take advantage of such a tunnel to provide inter-DC connectivity.

This spec attempts to enhance the BGPVPN service (runs on top of the current L3 Forwarding service) in NetVirt to embrace inter-DC L3 connectivity over external VXLAN tunnels.

## In scope

The scope primarily includes providing ability to support Inter-subnet connectivity across DataCenters over VXLAN tunnels by modeling a new type of L3VPN which will realize this connectivity using EVPN BGP Control plane semantics.

When we mention that we are using EVPN BGP Control plane, this spec proposes using the RouteType 5 explained in [EVPN\\_RT5](#) as the primary means to provision the control plane and enable inter-DC connectivity over external VXLAN tunnels.

This new type of L3VPN will also inclusively support:

- Intra-subnet connectivity within a DataCenter over VXLAN tunnels.
- Inter-subnet connectivity within a DataCenter over VXLAN tunnels.

## Out of scope

- Does not cover providing VXLAN connectivity between hypervisors (with OVS Datapath) and Top-Of-Rack switches that might be positioned within such DataCenters.
- Does not cover providing intra-subnet connectivity across DCs.

Both the points above will be covered by another spec that will be Phase 2 of realizing intra-subnet inter-DC connectivity.

## Use Cases

The following high level use-cases will be realized by the implementation of this Spec.

### DataCenter access from a WAN-client via DC-Gateway (Single Homing)

This use case involves communication within the DataCenter by tenant VMs and also communication between the tenant VMs to a remote WAN-based client via DC-Gateway. The dataplane between the tenant VMs themselves and between the tenant VMs towards the DC-Gateway will be over VXLAN Tunnels.

The dataplane between the DC-Gateway to its other WAN-based BGP Peers is transparent to this spec. It is usually MPLS-based IPVPN.

The BGP Control plane between the ODL Controller and the DC-Gateway will be via EVPN RouteType 5 as defined in [EVPN\\_RT5](#).

The control plane between the DC-Gateway and its other BGP Peers in the WAN is transparent to this spec, but can be IP-MPLS.

In this use-case:

1. We will have only a single DCGW for WAN connectivity
2. IP prefix exchange between ODL controller and DC-GW (iBGP) using EVPN RT5
3. WAN control plane will use L3VPN IP-MPLS route exchange.
4. On the DC-Gateway, the VRF instance will be configured with two sets of import/export targets. One set of import/export route targets belong to L3VPN inside DataCenter (realized using EVPN RT5) and the second set of import/export route target belongs to WAN control plane.

5. EVPN single homing to be used in all RT5 exchanges inside the DataCenter i.e., ESI=0 for all prefixes sent from DataCenter to the DC-Gateway.
6. Inter AS option B is used at DCGW, route regeneration at DCGW

### **Datacenter access from another Datacenter over WAN via respective DC-Gateways (L3 DCI)**

This use-case involves providing inter-subnet connectivity between two DataCenters. Tenant VMs in one datacenter will be able to communicate with tenant VMs on the other datacenter provided they are part of the same L3VPN and they are on different subnets.

Both the Datacenters can be managed by different ODL Controllers, but the L3VPN configured on both ODL Controllers will use identical RDs and RTs.

### **Proposed change**

The following components of an Openstack-ODL-based solution need to be enhanced to provide intra-subnet and inter-subnet connectivity across DCs using EVPN IP Prefix Advertisement (Route Type 5) mechanism (refer [EVPN\\_RT5](#)):

- Openstack Neutron BGPVPN Driver
- OpenDaylight Controller (NetVirt)
- BGP Quagga Stack to support EVPN with RouteType 5 NLRI
- DC-Gateway BGP Neighbour that supports EVPN with RouteType 5 NLRI

The changes required in Openstack Neutron BGPVPN Driver and BGP Quagga Stack are captured in the Solution considerations section down below.

### **Pipeline changes**

For both the use-cases above, we have put together the required pipeline changes here. For ease of understanding, we have made subsections that talk about Intra-DC traffic and Inter-DC traffic.

#### **INTRA DC**

##### **Intra Subnet, Local DPN: VMs on the same subnet, same VPN, same DPN**

There are no explicit pipeline changes for this use-case. However the tables that a packet will traverse through is shown below for understanding purposes.

Classifier Table (0) =>

Lport Dispatcher Table (17) l3vpn service: set vpn-id=l3vpn-id =>

L3 Gateway MAC Table (19) tablemiss: goto\_table=17 =>

Lport Dispatcher Table (17) elan service: set elan-id=elan-tag =>

ELAN Source MAC Table (50) match: elan-id=elan-tag, src-mac=source-vm-mac =>

ELAN Destination MAC Table (51) match: elan-id=elan-tag, dst-mac=dst-vm-mac set output to port-of-dst-vm



## Intra Subnet, Remote DPN: VMs on two different DPNs, both VMs on the same subnet and same VPN

There are no explicit pipeline changes for this use-case. However the tables that a packet will traverse through is shown below for understanding purposes.

### VM sourcing the traffic (Ingress DPN)

Classifier Table (0) =>

Lport Dispatcher Table (17) l3vpn service: set vpn-id=l3vpn-id=>

L3 Gateway MAC Table (19) l3vpn service: tablemiss: goto\_table=17=>

Lport Dispatcher Table (17) elan service: set elan-id=elan-tag=>

ELAN Source MAC Table (50) match: elan-id=elan-tag, src-mac=source-vm-mac=>

ELAN Destination MAC Table (51) match: elan-id=elan-tag, dst-mac=dst-vm-mac set tun-id=dst-vm-lport-tag, output to vxlan-tun-port

### VM receiving the traffic (Egress DPN)

Classifier Table (0) =>

Internal Tunnel Table (36) match: tun-id=lport-tag set reg6=dst-vm-lport-tag=>

Lport Egress Table (220) Output to dst vm port

## Inter Subnet, Local DPN: VMs on different subnet, same VPN, same DPN

There are no explicit pipeline changes for this use-case. However the tables that a packet will traverse through is shown below for understanding purposes.

Classifier Table (0) =>

Lport Dispatcher Table (17) l3vpn service: set vpn-id=l3vpn-id=>

L3 Gateway MAC Table (19) match: vpn-id=l3vpn-id,  
dst-mac=vpn-subnet-gateway-mac-address=>

L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set output to nexthopgroup-dst-vm=>

NextHopGroup-dst-vm: set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag=>

Lport Egress Table (220) Output to dst vm port

```
cookie=0x8000000, table=0, priority=4, in_port=1 actions=write_metadata:0x10000000000/
↪0xffffffff0000000001, goto_table:17
cookie=0x8000001, table=17, priority=5, metadata=0x5000010000000000/0xffffffff0000000000,
↪actions=write_metadata:0x60000100000222e0/0xfffffffffffffffffe, goto_table:19
cookie=0x8000009, table=19, priority=20, metadata=0x222e0/0xfffffffffe, dl_
↪dst=de:ad:be:ef:00:01 actions=goto_table:21
cookie=0x8000003, table=21, priority=42, ip, metadata=0x222e0/0xfffffffffe, nw_dst=10.0.0.
↪2 actions=apply_actions(group:150001)
```

## Inter Subnet, Remote DPN: VMs on two different DPNs, both VMs on different subnet, but same VPN

For this use-case there is a change in the remote flow rule to L3 Forward the traffic to the remote VM. The flow-rule will use the LPortTag as the vxlan-tunnel-id, in addition to setting the destination mac address of the remote destination vm.

### VM sourcing the traffic (Ingress DPN)

Classifier Table (0) =>

Lport Dispatcher Table (17) l3vpn service: set vpn-id=l3vpn-id=>

L3 Gateway MAC Table (19) match: vpn-id=l3vpn-id,  
dst-mac=vpn-subnet-gateway-mac-address =>

L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set  
eth-dst-mac=dst-vm-mac, tun-id=dst-vm-lport-tag, output to vxlan-tun-port

```
cookie=0x8000000, table=0, priority=4, in_port=1 actions=write_metadata:0x10000000000/  
→0xffffffff0000000001, goto_table:17  
cookie=0x8000001, table=17, priority=5, metadata=0x5000010000000000/0xffffffff0000000000,  
→actions=write_metadata:0x60000100000222e0/0xffffffffffffffffffe, goto_table:19  
cookie=0x8000009, table=19, priority=20, metadata=0x222e0/0xfffffffffe, dl_  
→dst=de:ad:be:ef:00:01 actions=goto_table:21  
cookie=0x8000003, table=21, priority=42, ip, metadata=0x222e0/0xfffffffffe, nw_dst=10.0.0.  
→2 actions=apply_actions(group:150001)  
cookie=0x8000003, table=21, priority=42, ip, metadata=0x222e0/0xfffffffffe, nw_dst=10.0.0.  
→3 actions=apply_actions(set_field:fa:16:3e:f8:59:af->eth_dst, set_field:0x2->tun_id,  
→output:2)
```

As you can notice 0x2 set in the above flow-rule as tunnel-id is the LPortTag assigned to VM holding IP Address 10.0.0.3.

### VM receiving the traffic (Egress DPN)

Classifier Table (0) =>

Internal Tunnel Table (36) match: tun-id=lport-tag set reg6=lport-tag-dst-vm=>

Lport Egress Table (220) Output to dst vm port

```
cookie=0x8000001, table=0, priority=5, in_port=2 actions=write_metadata:0x40000000001/  
→0xffffffff0000000001, goto_table:36  
cookie=0x9000001, table=36, priority=5, tun_id=0x2 actions=load:0x400->NXM_NX_REG6[],  
→resubmit(, 220)
```

As you notice, 0x2 tunnel-id match in the above flow-rule in INTERNAL\_TUNNEL\_TABLE (Table 36), is the LPort-Tag assigned to VM holding IP Address 10.0.0.3.

## INTER DC

### Intra Subnet

Not supported in this Phase

### Inter Subnet

For this use-case we are doing a couple of pipeline changes:

- a. Introducing a new Table aka **L3VNI\_EXTERNAL\_TUNNEL\_DEMUX\_TABLE** (Table 23). **L3VNI\_EXTERNAL\_TUNNEL\_DEMUX\_TABLE** (Table 23) - This table is a new table in the L3VPN pipeline and will be responsible only to process VXLAN packets coming from External VXLAN tunnels.

The packets coming from External VXLAN Tunnels (note: not Internal VXLAN Tunnels), would be directly punted to this new table from the CLASSIFIER TABLE (Table 0) itself. Today when multiple services bind to a tunnel port on GENIUS, the service with highest priority binds directly to Table 0 entry for the tunnel port. So such a service should make sure to provide a fallback to Dispatcher Table so that subsequent service interested in that tunnel traffic would be given the chance.

The new table **L3VNI\_EXTERNAL\_TUNNEL\_DEMUX\_TABLE** will have flows to match on VXLAN VNIs that are L3VNIs. On a match, their action is to fill the metadata with the VPNID, so that further tables in the L3VPN pipeline would be able to continue and operate with the VPNID metadata seamlessly. After filling the metadata, the packets are resubmitted from this new table to the **L3\_GW\_MAC\_TABLE** (Table 19). The TableMiss in **L3VNI\_EXTERNAL\_TUNNEL\_DEMUX\_TABLE** will resubmit the packet to **LPORT\_DISPATCHER\_TABLE** to enable next service if any to process the packet ingressing from the external VXLAN tunnel.

- b. For all packets going from VMs within the DC, towards the external gateway device via the External VXLAN Tunnel, we are setting the VXLAN Tunnel ID to the L3VNI value of VPNInstance to which the VM belongs to.

### Traffic from DC-Gateway to Local DPN (SYMMETRIC IRB)

Classifier Table (0) =>

L3VNI External Tunnel Demux Table (23) match: tun-id=l3vni set vpn-id=l3vpn-id=>

L3 Gateway MAC Table (19) match: vpn-id=l3vpn-id,  
dst-mac=vpn-subnet-gateway-mac-address =>

L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set output to  
nexthopgroup-dst-vm =>

NextHopGroup-dst-vm: set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag =>

Lport Egress Table (220) Output to dst vm port

```
cookie=0x80000001, table=0, priority=5, in_port=9 actions=write_metadata:0x700000000001/
->0x1ffffff0000000001, goto_table:23
cookie=0x80000001, table=19, priority=20, metadata=0x222e0/0xffffffff, dl_
->dst=de:ad:be:ef:00:06 actions=goto_table:21
cookie=0x80000001, table=23, priority=5, tun_id=0x16 actions= write_metadata:0x222e0/
->0xfffffffffe, resubmit(19)
cookie=0x80000001, table=23, priority=0, resubmit(17)
cookie=0x80000003, table=21, priority=42, ip, metadata=0x222e0/0xfffffffffe, nw_dst=10.0.0.
->2 actions=apply_actions(group:150001)
cookie=0x80000003, table=21, priority=42, ip, metadata=0x222e0/0xfffffffffe, nw_dst=10.0.0.
->3 actions=apply_actions(set_field:fa:16:3e:f8:59:af->eth_dst, set_field:0x2->tun_id,
->output:2)
```

In the above flow rules, Table 23 is the new L3VNI\_EXTERNAL\_TUNNEL\_DEMUX\_TABLE. The in\_port=9 represents an external VXLAN Tunnel port.

### Traffic from Local DPN to DC-Gateway (SYMMETRIC IRB)

Classifier Table (0) =>

Lport Dispatcher Table (17) l3vpn service: set vpn-id=l3vpn-id=>

L3 Gateway MAC Table (19) match: vpn-id=l3vpn-id,  
dst-mac=vpn-subnet-gateway-mac-address =>

L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=ext-ip-address set  
eth-dst-mac=dst-mac-address, tun-id=l3vni, output to ext-vxlan-tun-port

```
cookie=0x70000001, table=0, priority=5, in_port=8, actions=write_metadata:0x600000000001/  
->0x1fffff0000000001, goto_table:17  
cookie=0x70000001, table=17, priority=5, metadata=0x600000000001/0x1fffff0000000001,  
->actions=goto_table:19  
cookie=0x70000001, table=19, priority=20, metadata=0x222e0/0xffffffff, dl_  
->dst=de:ad:be:ef:00:06 actions=goto_table:21  
cookie=0x70000001, table=23, priority=5, tun_id=0x16 actions= write_metadata:0x222e0/  
->0xfffffffffe, resubmit (19)  
cookie=0x70000001, table=23, priority=0, resubmit (17)  
cookie=0x70000003, table=21, priority=42, ip, metadata=0x222e0/0xfffffffffe, nw_dst=10.0.0.  
->2 actions=apply_actions(group:150001)  
cookie=0x70000003, table=21, priority=42, ip, metadata=0x222e0/0xfffffffffe, nw_dst=10.0.0.  
->3 actions=apply_actions(set_field:fa:16:3e:f8:59:af->eth_dst, set_field:0x2->tun_id,  
->output:2)
```

### SNAT pipeline (Access to External Network Access over VXLAN)

#### SNAT Traffic from Local DPN to External IP (assuming this DPN is NAPT Switch)

Classifier Table (0) =>

Lport Dispatcher Table (17) l3vpn service: set vpn-id=l3vpn-id=>

L3 Gateway MAC Table (19) match: vpn-id=l3vpn-id,  
dst-mac=vpn-subnet-gateway-mac-address =>

L3 FIB Table (21) match: vpn-id=l3vpn-id=>

PSNAT Table (26) match: vpn-id=l3vpn-id=>

Outbound NAPT Table (46) match: nw-src=vm-ip, port=int-port set  
src-ip=router-gateway-ip, src-mac=external-router-gateway-mac-address,  
vpn-id=external-vpn-id, port=ext-port  
=>

NAPT PFIB Table (47) match: vpn-id=external-vpn-id=>

L3 FIB Table (21) match: vpn-id=external-vpn-id nw-dst=external-entity-ip set  
eth-dst=external-entity-mac tun-id=external-l3vni, output to  
ext-vxlan-tun-port

**SNAT Reverse Traffic from External IP to Local DPN (assuming this DPN is NAPT Switch)**

Classifier Table (0) =>

L3VNI External Tunnel Demux Table (23) match: tun-id=external-l3vni set  
vpn-id=external-vpn-id =>

L3 Gateway MAC Table (19) match: vpn-id=external-vpn-id,  
dst-mac=external-router-gateway-mac-address =>

Inbound NAPT Table (44) match: vpn-id=external-vpn-id nw-dst=router-gateway-ip  
port=ext-port set vpn-id=l3vpn-id, dst-ip=vm-ip

NAPT PFIB Table (47) match: vpn-id=l3vpn-id =>

L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set output to  
nexthopgroup-dst-vm =>

NextHopGroup-dst-vm: set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag =>

Lport Egress Table (220) Output to dst vm port

**DNAT pipeline (Access from External Network over VXLAN)****DNAT Traffic from External IP to Local DPN**

Classifier Table (0) =>

L3VNI External Tunnel Demux Table (23) match: tun-id=external-l3vni set  
vpn-id=external-vpn-id =>

L3 Gateway MAC Table (19) match: vpn-id=external-vpn-id,  
eth-dst=floating-ip-dst-vm-mac-address =>

PDNAT Table (25) match: nw-dst=floating-ip, eth-dst=floating-ip-dst-vm-mac-address  
set ip-dst=dst-vm-ip, vpn-id=l3vpn-id =>

DNAT Table (27) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip =>

L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip set output to  
nexthopgroup-dst-vm =>

NextHopGroup-dst-vm: set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag =>

Lport Egress Table (220) Output to dst vm port

**DNAT Reverse Traffic from Local DPN to External IP**

Classifier Table (0) =>

Lport Dispatcher Table (17) l3vpn service: set vpn-id=l3vpn-id =>

L3 Gateway MAC Table (19) match: vpn-id=l3vpn-id,  
dst-mac=vpn-subnet-gateway-mac-address =>

L3 FIB Table (21) match: vpn-id=l3vpn-id =>

PSNAT Table (26) match: vpn-id=l3vpn-id nw-src=src-vm-ip set  
ip-src=floating-ip-src-vm, vpn-id=external-vpn-id =>

SNAT Table (28) match: vpn-id=external-vpn-id nw-src=floating-ip-src-vm set  
eth-src=floating-ip-src-vm-mac-address =>

L3 FIB Table (21) match: vpn-id=external-vpn-id nw-dst=external-floating-ip set  
eth-dst=external-mac-address tun-id=external-l3vni, output to  
ext-vxlan-tun-port

## DNAT to DNAT Traffic (Intra DC)

### 1. FIP VM to FIP VM on Different Hypervisor

#### DPN1:

Classifier Table (0) =>

Lport Dispatcher Table (17) l3vpn service: set vpn-id=l3vpn-id=>

L3 Gateway MAC Table (19) match: vpn-id=l3vpn-id,  
dst-mac=vpn-subnet-gateway-mac-address =>

L3 FIB Table (21) match: vpn-id=l3vpn-id=>

PSNAT Table (26) match: vpn-id=l3vpn-id nw-src=src-vm-ip set  
ip-src=floating-ip-src-vm, vpn-id=external-vpn-id=>

SNAT Table (28) match: vpn-id=external-vpn-id nw-src=floating-ip-src-vm set  
eth-src=floating-ip-src-vm-mac-address =>

L3 FIB Table (21) match: vpn-id=external-vpn-id nw-dst=destination-floating-ip set  
eth-dst=floating-ip-dst-vm-mac-address tun-id=external-l3vni, output to  
vxlan-tun-port

#### DPN2:

Classifier Table (0) =>

Internal Tunnel Table (36) match: tun-id= external-l3vni =>

PDNAT Table (25) match: nw-dst=floating-ip eth-dst=floating-ip-dst-vm-mac-address  
set ip-dst=dst-vm-ip, vpn-id=l3vpn-id=>

DNAT Table (27) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip =>

L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip set output to  
nexthopgroup-dst-vm=>

NextHopGroup-dst-vm: set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag=>

Lport Egress Table (220) Output to dst vm port

In the above flow rules INTERNAL\_TUNNEL\_TABLE`` (table=36) will take the packet to the ``PDNAT\_TABLE`` (table 25) for an exact match with floating-ip and floating-ip-dst-vm-mac-address in PDNAT\_TABLE.

In case of a successful floating-ip and floating-ip-dst-vm-mac-address match, PDNAT\_TABLE`` will set IP destination as VM IP and VPN ID as internal l3 VPN ID then it will pointing to ``DNAT\_TABLE (table=27)

In case of no match, the packet will be redirected to the SNAT pipeline towards the “INBOUND\_NAPT\_TABLE” (table=44). This is the use-case where “DPN2” also acts as the NAPT DPN.

In summary, on an given NAPT switch, if both DNAT and SNAT are configured, the incoming traffic will first be sent to the PDNAT\_TABLE`` and if there is no FIP and FIP Mac match found, then it will be forwarded to ``INBOUND\_NAPT\_TABLE`` for SNAT translation. As part of the response, the ``external-l3vni`` will be used as ``tun\_id`` to reach floating IP VM on ``DPN1.

### 2. FIP VM to FIP VM on same Hypervisor

Classifier Table (0) =>

Lport Dispatcher Table (17) l3vpn service: set vpn-id=l3vpn-id=>

L3 FIB Table (21) match: vpn-id=l3vpn-id=>

PSNAT Table (26) match: vpn-id=l3vpn-id nw-src=src-vm-ip set  
ip-src=floating-ip-src-vm, vpn-id=external-vpn-id=>

SNAT Table (28) match: vpn-id=external-vpn-id nw-src=floating-ip-src-vm set  
eth-src=floating-ip-src-vm-mac-address =>

L3 FIB Table (21) match: vpn-id=external-vpn-id nw-dst=destination-floating-ip set  
eth-dst= floating-ip-dst-vm-mac-address =>

PDNAT Table (25) match: nw-dst=floating-ip eth-dst=floating-ip-dst-vm-mac-address  
set ip-dst=dst-vm-ip, vpn-id=l3vpn-id=>

DNAT Table (27) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip =>

L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip set output to  
nexthopgroup-dst-vm=>

NextHopGroup-dst-vm: set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag=>

Lport Egress Table (220) Output to dst vm port

## SNAT to DNAT Traffic (Intra DC)

### SNAT Hypervisor:

Classifier Table (0) =>

Lport Dispatcher Table (17) l3vpn service: set vpn-id=l3vpn-id=>

L3 Gateway MAC Table (19) match: vpn-id=l3vpn-id,  
dst-mac=vpn-subnet-gateway-mac-address =>

L3 FIB Table (21) match: vpn-id=l3vpn-id=>

PSNAT Table (26) match: vpn-id=l3vpn-id=>

Outbound NAPT Table (46) match: nw-src=vm-ip, port=int-port set  
src-ip=router-gateway-ip, src-mac=external-router-gateway-mac-address,  
vpn-id=external-vpn-id, port=ext-port  
=>

NAPT PFIB Table (47) match: vpn-id=external-vpn-id=>

L3 FIB Table (21) match: vpn-id=external-vpn-id nw-dst=destination-floating-ip set  
eth-dst=floating-ip-dst-vm-mac-address tun-id=external-l3vni, output to  
vxlan-tun-port

### DNAT Hypervisor:

Classifier Table (0) =>

Internal Tunnel Table (36) match: tun-id= external-l3vni=>

PDNAT Table (25) “match: nw-dst=floating-ip eth-dst= floating-ip-dst-vm-mac-address set ip-dst=dst-vm-ip,  
vpn-id=l3vpn-id“=>

DNAT Table (27) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip=>

L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip set output to  
nexthopgroup-dst-vm=>

NextHopGroup-dst-vm: set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag=>

Lport Egress Table (220) Output to dst vm port

## Non-NAPT to NAPT Forward Traffic (Intra DC)

### Non-NAPT Hypervisor:

Classifier Table (0) =>

Lport Dispatcher Table (17) l3vpn service: set vpn-id=l3vpn-id =>

L3 Gateway MAC Table (19) match: vpn-id=l3vpn-id,  
dst-mac=vpn-subnet-gateway-mac-address =>

L3 FIB Table (21) match: vpn-id=l3vpn-id =>

PSNAT Table (26) match: vpn-id=l3vpn-id set tun-id=router-lport-tag, group =>  
group: output to NAPT vxlan-tun-port

### NAPT Hypervisor:

Classifier Table (0) =>

Internal Tunnel Table (36) match: tun-id=router-lport-tag =>

Outbound NAPT Table (46) match: nw-src=vm-ip, port=int-port set  
src-ip=router-gateway-ip, src-mac=external-router-gateway-mac-address,  
vpn-id=external-vpn-id, port=ext-port  
=>

NAPT PFIB Table (47) match: vpn-id=external-vpn-id =>

L3 FIB Table (21) match: vpn-id=external-vpn-id nw-dst=external-entity-ip set  
eth-dst=external-entity-mac tun-id=external-l3vni, output to  
ext-vxlan-tun-port

For forwarding the traffic from Non-NAPT to NAPT DPN the tun-id will be setting with “router-lport-tag” which will be carved out per router.

## NAPT to Non-NAPT Reverse Traffic (Intra DC)

### NAPT Hypervisor:

Classifier Table (0) =>

L3VNI External Tunnel Demux Table (23) match: tun-id=external-l3vni set  
vpn-id=external-vpn-id =>

L3 Gateway MAC Table (19) match: vpn-id=external-vpn-id,  
dst-mac=external-router-gateway-mac-address =>

Inbound NAPT Table (44) match: vpn-id=external-vpn-id nw-dst=router-gateway-ip  
port=ext-port set vpn-id=l3vpn-id, dst-ip=vm-ip =>

NAPT PFIB Table (47) match: vpn-id=l3vpn-id =>

L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set  
eth-dst-mac=dst-vm-mac, tun-id=dst-vm-lport-tag, output to vxlan-tun-port

### Non-NAPT Hypervisor:

Classifier Table (0) =>



Internal Tunnel Table (36) match: tun-id=dst-vm-lport-tag=>  
 L3 FIB Table (21) match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip set output to  
 nexthopgroup-dst-vm=>  
 NextHopGroup-dst-vm: set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag=>  
 Lport Egress Table (220) Output to dst vm port

More details of the NAT pipeline changes are in the NAT Service section of this spec.

## Yang changes

Changes will be needed in `l3vpn.yang`, `odl-l3vpn.yang`, `odl-fib.yang` and `neutronvpn.yang` to start supporting EVPN functionality.

## L3VPN YANG changes

A new leaf `l3vni` and a new leaf type will be added to container `vpn-instances`

Listing 1.30: `l3vpn.yang`

```
leaf type {
  description
    "The type of the VPN Instance.
    ipvpn indicates it is an L3VPN.
    evpn indicates it is EVPN";

  type enumeration {
    enum ipvpn {
      value "0";
      description "L3VPN";
    }
    enum evpn {
      value "1";
      description "EVPN";
    }
  }
  default "ipvpn";
}

leaf l3vni {
  description
    "The L3 VNI to use for this L3VPN Instance.
    If this attribute is non-zero, it indicates
    this L3VPN will do L3Forwarding over VXLAN.
    If this value is non-zero, and the type field is 'l2',
    it is an error.
    If this value is zero, and the type field is 'l3', it is
    the legacy L3VPN that will do L3Forwarding
    with MPLSoverGRE.
    If this value is zero, and the type field is 'l2', it
    is an EVPN that will provide L2 Connectivity with
    Openstack supplied VNI".

  type uint24;
  mandatory false;
```

```
}
```

The **\*\*type\*\*** value comes from Openstack BGPVPN ODL Driver based on what type of ↵BGPVPN is orchestrated by the tenant. That same **\*\*type\*\*** value must be retrieved and stored ↵into VPNInstance model above maintained by NeutronvpnManager.

## ODL-L3VPN YANG changes

A new leaf l3vni and a new leaf type will be added to container vpn-instance-op-data

Listing 1.31: odl-l3vpn.yang

```
leaf type {
    description
        "The type of the VPN Instance.
        ipvpn indicates it is an L3VPN.
        evpn indicates it is EVPN";

    type enumeration {
        enum ipvpn {
            value "0";
            description "L3VPN";
        }
        enum evpn {
            value "1";
            description "EVPN";
        }
    }
    default "ipvpn";
}

leaf l3vni {
    description
        "The L3 VNI to use for this L3VPN Instance.
        If this attribute is non-zero, it indicates
        this L3VPN will do L3Forwarding over VXLAN.
        If this value is non-zero, and the type field is 'l2',
        it is an error.
        If this value is zero, and the type field is 'l3', it is
        the legacy L3VPN that will do L3Forwarding
        with MPLSoverGRE.
        If this value is zero, and the type field is 'l2', it
        is an EVPN that will provide L2 Connectivity with
        Openstack supplied VNI".

    type uint24;
    mandatory false;
}
```

For every interface in the cloud that is part of an L3VPN which has an L3VNI setup, ↵we should extract that L3VNI from the config VPNInstance and use that to both program the flows ↵as well as advertise to BGP Neighbour using RouteType 5 BGP Route exchange.

Fundamentally, what we are accomplishing is L3 Connectivity over VXLAN tunnels by using the EVPN RT5 mechanism.

## ODL-FIB YANG changes

Few new leafs like `mac_address` , `gateway_mac_address` , `l2vni` , `l3vni` and a leaf `encap-type` will be added to container `fibEntries`

Listing 1.32: odl-fib.yang

```
leaf encap-type {
  description
    "This flag indicates how to interpret the existing label field.
    A value of mpls indicates that the label will continue to
    be considered as an MPLS Label.
    A value of vxlan indicates that vni should be used to
    advertise to bgp.
  type enumeration {
    enum mplsgre {
      value "0";
      description "MPLSOverGRE";
    }
    enum vxlan {
      value "1";
      description "VNI";
    }
  }
  default "mplsgre";
}

leaf mac_address {
  type string;
  mandatory false;
}

leaf l3vni {
  type uint24;
  mandatory false;
}

leaf l2vni {
  type uint24;
  mandatory false;
}

leaf gateway_mac_address {
  type string;
  mandatory false;
}
Augment:parent_rd {
  type string;
  mandatory false;
}
```

The `encap-type` indicates whether an `MPLSOverGre` or `VXLAN` encapsulation should be used for this route. If the

encapType is MPLSOVerGre then the usual label field will carry the MPLS Label to be used in datapath for traffic to/from this VRFEntry IP prefix.

If the encapType is VXLAN, the VRFEntry implicitly refers that this route is reachable via a VXLAN tunnel. The L3VNI will carry the VRF VNI and there will also be an L2VNI which represents the VNI of the network to which the VRFEntry belongs to.

Based on whether Symmetric IRB (or) Asymmetric IRB is configured to be used by the CSC (see section on Configuration Impact below). If Symmetric IRB is configured, then the L3VNI should be used to program the flows rules. If Asymmetric IRB is configured, then L2VNI should be used in the flow rules.

The mac\_address field must be filled for every route in an EVPN. This mac\_address field will be used for support intra-DC communication for both inter-subnet and intra-subnet routing.

The gateway\_mac\_address must always be filled for every route in an EVPN.[AKMA7] [NV8] This gateway\_mac\_address will be used for all packet exchanges between DC-GW and the DPN in the DC to support L3 based forwarding with Symmetric IRB.

## NEUTRONVPN YANG changes

One new leaf l3vni will be added to container grouping vpn-instance

Listing 1.33: odl-fib.yang

```
leaf l3vni {  
    type uint32;  
    mandatory false;  
}
```

## Solution considerations

### Proposed change in Openstack Neutron BGPVPN Driver

The Openstack Neutron BGPVPN's ODL driver in Newton release needs to be changed, so that it is able to relay the configured L2 BGPVPNs, to the OpenDaylight Controller. As of Mitaka release, only L3 BGPVPNs configured in Openstack are being relayed to the OpenDaylight Controller. So in addition to addressing the ODL BGPVPN Driver changes in Newton, we will provide a Mitaka based patch that will integrate into Openstack.

The Newton changes for the BGPVPN Driver has merged and is here: <https://review.openstack.org/#/c/370547/>

### Proposed change in BGP Quagga Stack

The BGP Quagga Stack is a component that interfaces with ODL Controller to enable ODL Controller itself to become a BGP Peer. This BGP Quagga Stack need to be enhanced so that it is able to embrace EVPN with Route Type 5 on the following two interfaces:

- Thrift Interface where ODL pushes routes to BGP Quagga Stack
- Route exchanges from BGP Quagga Stack to other BGP Neighbors (including DC-GW).

### Proposed change in OpenDaylight-specific features

The following components within OpenDaylight Controller needs to be enhanced:

- NeutronvpnManager
- VPN Engine (VPN Manager and VPN Interface Manager)
- FIB Manager
- BGP Manager
- VPN SubnetRoute Handler
- NAT Service

### Import Export RT support for EVPN

Currently Import/Export logic for L3VPN uses a LabelRouteInfo structure to build information about imported prefixes using MPLS Label as the key. However, this structure cannot be used for EVPN as the L3VNI will be applicable for an entire EVPN Instance instead of the MPLS Label. In lieu of LabelRouteInfo, we will maintain an IPPrefixInfo keyed structure that can be used for facilitating Import/Export of VRFEntries across both EVPNs and L3VPNs.

Listing 1.34: odl-fib.yang

```
list ipprefix-info {
    key "prefix, parent-rd"
    leaf prefix {
        type string;
    }

    leaf parent-rd {
        type string;
    }

    leaf label {
        type uint32;
    }

    leaf dpn-id {
        type uint64;
    }

    leaf-list next-hop-ip-list {
        type string;
    }

    leaf-list vpn-instance-list {
        type string;
    }

    leaf parent-vpnid {
        type uint32;
    }

    leaf vpn-interface-name {
        type string;
    }

    leaf elan-tag {
        type uint32;
    }
}
```

```
leaf is-subnet-route {
    type boolean;
}

leaf encap-type {
    description
        "This flag indicates how to interpret the existing label field.
        A value of mpls indicates that the l3label should be considered as an MPLS
        Label.
        A value of vxlan indicates that l3label should be considered as an VNI."
    type enumeration {
        enum mplsgre {
            value "0";
            description "MPLSOverGRE";
        }
        enum vxlan {
            value "1";
            description "VNI";
        }
        default "mplsgre";
    }
}

leaf l3vni {
    type uint24;
    mandatory false;
}

leaf l2vni {
    type uint24;
    mandatory false;
}

leaf gateway_mac_address {
    type string;
    mandatory false;
}
}
```

### SubnetRoute support on EVPN

The subnetRoute feature will continue to be supported on EVPN and we will use RT5 to publish subnetRoute entries with either the router-interface-mac-address if available (or) if not available use the pre-defined hardcoded MAC Address described in section Configuration Impact. For both ExtraRoutes and MIPs (invisible IPs) discovered via subnetroute, we will continue to use RT5 to publish those prefixes.[AKMA9] [NV10] On the dataplane, VXLAN packets from the DC-GW will carry the MAC Address of the gateway-ip for the subnet in the inner DMAC.

### NAT Service support for EVPN

However, since external network NAT should continue to be supported on VXLAN, making NAT service work on L3VPNs that use VXLAN as the tunnel type becomes imperative.

Existing SNAT/DNAT design assumed internetVpn to be using mplsogre as the connectivity from external network towards DCGW. This needs to be changed such that it can handle even EVPN case with VXLAN connectivity as well.

As of the implementation required for this specification, the workflow will be to create InternetVPN with and associate a single external network to that is of VXLAN Provider Type. The Internet VPN itself will be an L3VPN that will be created via the ODL RESTful API and during creation an L3VNI parameter will be supplied to enable this L3VPN to operate on a VXLAN dataplane. The L3VNI provided to the Internet VPN can be different from the VXLAN segmentation ID associated to the external network.

However, it will be a more viable use-case in the community if we mandate in our workflow that both the L3VNI configured for Internet VPN and the VXLAN segmentation id of the associated external network to the Internet VPN be the same. NAT service can use vpninstance-op-data model to classify the DCGW connectivity for internetVpn.

For the Pipeline changes for NAT Service, please refer to ‘Pipeline changes’ section.

SNAT to start using Router Gateway MAC, in translated entry in table 46 (Outbound SNAT table) and in table 19 (L3\_GW\_MAC\_Table). Presently Router gateway mac is already stored in odl-nat model in External Routers.

DNAT to start using Floating MAC, in table 28 (SNAT table) and in table 19 (L3\_GW\_MAC Table). Change in pipeline mainly reverse traffic for SNAT and DNAT so that when packet arrives from DCGW, it goes to 0->38->17->19 and based on Vni and MAC matching, take it back to SNAT or DNAT pipelines.

Also final Fib Entry pointing to DCGW in forward direction also needs modification where we should start using VXLAN’s vni, FloatingIPMAC (incase of DNAT) and ExternalGwMacAddress(incase of SNAT) and finally encapsulation type as VXLAN.

For SNAT advertise to BGP happens during external network association to Vpn and during High availability scenarios where you need to re-advertise the NAPT switch. For DNAT we need to advertise when floating IP is associated to the VM. For both SNAT and DNAT this IS mandates that we do only RT5 based advertisement. That RT5 advertisement must carry the external gateway mac address assigned for the respective Router for SNAT case while for DNAT case the RT5 will carry the floating-ip-mac address.

## ARP request/response and MIP handling Support for EVPN

Will not support ARP across DCs, as we donot support intra-subnet inter-DC scenarios.

- For intra-subnet intra-DC scenarios, the ARPs will be serviced by existing ELAN pipeline.
- For inter-subnet intra-DC scenarios, the ARPs will be processed by ARP Responder implementation that is already pursued in Carbon.
- For inter-subnet inter-DC scenarios, ARP requests won’t be generated by DC-GW. Instead the DC-GW will use ‘gateway mac’ extended attribute MAC Address information and put that directly into DSTMAC field of Inner MAC Header by the DC-GW for all packets sent to VMs within the DC.
- As quoted, intra-subnet inter-DC scenario is not a supported use-case as per this Implementation Spec.

## Tunnel state handling Support

We have to handle both the internal and external tunnel events for L3VPN (with L3VNI) the same way it is handled for current L3VPN.

## InterVPNLink support for EVPN

Not supported as this is not a requirement for this Spec.

## Supporting VLAN Aware VMs (Trunk and SubPorts)

Not supported as this is not a requirement for this Spec.

## VM Mobility with RT5

We will continue to support cold migration of VMs across hypervisors across L3VPNs as supported already in current ODL Carbon Release.

## Reboot Scenarios

This feature support all the following Reboot Scenarios for EVPN:

- Entire Cluster Reboot
- Leader PL reboot
- Candidate PL reboot
- OVS Datapath reboots
- Multiple PL reboots
- Multiple Cluster reboots
- Multiple reboots of the same OVS Datapath.
- Openstack Controller reboots

## Configuration impact

The following parameters have been initially made available as configurable for EVPN. These configurations can be made via the RESTful interface:

**1.Multi-homing-mode** – For multi-homing use cases where redundant DCGWs are used ODL can be configured with ‘none’, ‘all-active’ or ‘single-active’ multi-homing mode. Default will be ‘none’.

**2.IRB-mode** – Depending upon the support on DCGW, ODL can be configured with either ‘Symmetric’ or ‘Asymmetric’ IRB mode. Default is ‘Symmetric’.

There is another important parameter though it won’t be configurable:

**MAC Address Prefix for EVPN** – This MAC Address prefix represents the MAC Address prefix that will be hard-coded and that MACAddress will be used as the gateway mac address if it is not supplied from Openstack. This will usually be the case when networks are associated to an L3VPN with no gateway port yet configured in Openstack for such networks.

## Clustering considerations

The feature should operate in ODL Clustered environment reliably.

## Other Infra considerations

N.A.



## Security considerations

N.A.

## Scale and Performance Impact

Not covered by this Design Document.

## Targeted Release

Carbon.

## Alternatives

Alternatives considered and why they were not selected.

## Usage

### Features to Install

This feature can be used by installing odl-netvirt-openstack. This feature doesn't add any new karaf feature.

## REST API

The creational RESTful API for the L3VPN will be enhanced to accept the L3VNI as an additional attribute as in the below request format:

```
{'input': {
  'l3vpn': [
    {'name': 'L3VPN2',
      'export-RT': ['50:2'],
      'route-distinguisher': ['50:2'],
      'import-RT': ['50:2'],
      'id': '4ae8cd92-48ca-49b5-94e1-b2921a260007',
      'l3vni': '200',
      'tenant-id': 'a565b3ed854247f795c0840b0481c699'}
  ]
}}
```

There is no change in the REST API for associating networks, associating routers (or) deleting the L3VPN.

On the Openstack-side configuration, the vni\_ranges configured in Openstack Neutron ml2\_conf.ini should not overlap with the L3VNI provided in the ODL RESTful API. In an inter-DC case, where both the DCs are managed by two different Openstack Controller Instances, the workflow will be to do the following:

1. Configure the DC-GW2 facing OSC2 and DC-GW1 facing OSC1 with the same BGP configuration parameters.
2. On first Openstack Controller (OSC1) create an L3VPN1 with RD1 and L3VNI1
3. Create a network Net1 and Associate that Network Net1 to L3VPN1
4. On second Openstack Controller (OSC2) create an L3VPN2 with RD1 with L3VNI2
5. Create a network Net2 on OSC2 and associate that Network Net2 to L3VPN2.

6. Spin-off VM1 on Net1 in OSC1.
7. Spin-off VM2 on Net2 in OSC2.
8. Now VM1 and VM2 should be able to communicate.

## **Implementation**

### **Assignee(s)**

**Primary assignee:** Kiran N Upadhyaya ([kiran.n.upadhyaya@ericsson.com](mailto:kiran.n.upadhyaya@ericsson.com))

Sumanth MS ([sumanth.ms@ericsson.com](mailto:sumanth.ms@ericsson.com))

Basavaraju Chickmath ([basavaraju.chickmath@ericsson.com](mailto:basavaraju.chickmath@ericsson.com))

**Other contributors:** Vivekanandan Narasimhan ([n.vivekanandan@ericsson.com](mailto:n.vivekanandan@ericsson.com))

### **Work Items**

The Trello cards have already been raised for this feature under the EVPN\_RT5.

Here is the link for the Trello Card: <https://trello.com/c/Tfpr3ezF/33-evpn-evpn-rt5>

New tasks into this will be added to cover Java UT and CSIT.

### **Dependencies**

Requires a DC-GW that is supporting EVPN RT5 on BGP Control plane.

### **Testing**

Capture details of testing that will need to be added.

### **Unit Tests**

Appropriate UTs will be added for the new code coming in once framework is in place.

### **Integration Tests**

There won't be any Integration tests provided for this feature.

### **CSIT**

CSIT will be enhanced to cover this feature by providing new CSIT tests.

## Documentation Impact

This will require changes to User Guide and Developer Guide.

User Guide will need to add information on how OpenDaylight can be used to deploy L3 BGPVPNs and enable communication across datacenters between virtual endpoints in such L3 BGPVPN.

Developer Guide will capture the ODL L3VPN API changes to enable management of an L3VPN that can use VXLAN overlay to enable communication across datacenters.

## References

[1] [EVPN\\_RT5](#)

[2] [Network Virtualization using EVPN](#)

[3] [Integrated Routing and Bridging in EVPN](#)

[4] [VXLAN DCI using EVPN](#)

[5] [BGP MPLS-Based Ethernet VPN](#)

- <http://docs.opendaylight.org/en/latest/documentation.html>
- [https://wiki.opendaylight.org/view/Genius:Carbon\\_Release\\_Plan](https://wiki.opendaylight.org/view/Genius:Carbon_Release_Plan)

### 1.1.26 Temporary Source MAC Learning

<https://git.opendaylight.org/gerrit/#/q/topic:temp-smac-learning>

Temporary source MAC learning introduces two new tables to the ELAN service, for OVS-based source MAC learning using a learn action, to reduce a large scale of packets punted to the controller for an unlearned source MAC.

#### Problem description

Currently any packet originating from an unknown source MAC address is punted to the controller from the ELAN service (L2 SMAC table 50).

This behavior continues for each packet from this source MAC until ODL properly processes this packet and adds an explicit source MAC rule to this table.

During the time that is required to punt a packet, process it by the ODL and create an appropriate flow, it is not necessary to punt any other packet from this source MAC, as it causes an unnecessary load.

#### Use Cases

Any L2 traffic from unknown source MACs passing through the ELAN service.

#### Proposed change

A preliminary logic will be added prior to the SMAC learning table, that will use OpenFlow learn action to add a temporary rule for each source MAC after the first packet is punted.

## Pipeline changes

Two new tables will be introduced to the ELAN service:

**Table 48** for resubmitting to tables 49 and 50 (trick required to use the learned flows, similar to the ACL implementation).

**Table 49** for setting a register value to mark that this SMAC was already punted to the ODL for learning. The flows in this table will be generated automatically by OVS.

**Table 50** will be modified, with a new flow, which has a lower priority than the existing known SMAC flows but a higher priority than the default flow. This flow passes packets marked with the register directly to the DMAC table 51 without punting to the controller, as it is already being processed. In addition, the default flow that punts packets to the controller, will also have a new learn action, temporarily adding a flow matching this source MAC to table 49.

### Example of flows after change:

```
cookie=0x8040000, duration=1575.755s, table=17, n_packets=7865, n_
↳bytes=1451576, priority=6, metadata=0x6000020000000000/0xffffffff00000000,
↳actions=write_metadata:0x7000021389000000/0xfffffffffffffffe, goto_table:48
cookie=0x8500000, duration=1129.530s, table=48, n_packets=4149, n_
↳bytes=729778, priority=0 actions=resubmit(,49), resubmit(,50)
cookie=0x8600000, duration=6.875s, table=49, n_packets=0, n_bytes=0, hard_
↳timeout=60, priority=0, dl_src=fa:16:3e:2f:73:61 actions=load:0x1->NXM_NX_
↳REG4[0..7]
cookie=0x8051389, duration=7.078s, table=50, n_packets=0, n_bytes=0,
↳priority=20, metadata=0x21389000000/0xfffffffff000000, dl_
↳src=fa:16:3e:2f:73:61 actions=goto_table:51
cookie=0x8050000, duration=440.925s, table=50, n_packets=49, n_bytes=8030,
↳priority=10, reg4=0x1 actions=goto_table:51
cookie=0x8050000, duration=124.209s, table=50, n_packets=68, n_bytes=15193,
↳priority=0 actions=CONTROLLER:65535, learn(table=49, hard_timeout=60,
↳priority=0, cookie=0x8600000, NXM_OF_ETH_SRC[], load:0x1->NXM_NX_REG4[0..7]),
↳goto_table:51
```

## Yang changes

None.

## Configuration impact

None.

## Clustering considerations

None.

## Other Infra considerations

None.

## Security considerations

None.

## Scale and Performance Impact

This change should substantially reduce the packet in load from SMAC learning, resulting in a reduced load of the ODL in high performance traffic scenarios.

## Targeted Release

Due to scale and performance criticality, and the low risk of this feature, suggest to target this functionality for Boron.

## Alternatives

None.

## Usage

N/A.

## Features to Install

odl-netvirt-openstack

## REST API

N/A.

## CLI

N/A.

## Implementation

### Assignee(s)

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

**Primary assignee:** Olga Schukin ([olga.schukin@hpe.com](mailto:olga.schukin@hpe.com))

**Other contributors:** Alon Kochba ([alonko@hpe.com](mailto:alonko@hpe.com))

### Work Items

N/A.

## Dependencies

No new dependencies. Learn action is already in use in netvirt pipeline and has been available in OVS since early versions. However this is a non-standard OpenFlow feature.

## Testing

Existing source MAC learning functionality should be verified.

## Unit Tests

N/A.

## Integration Tests

N/A.

## CSIT

N/A.

## Documentation Impact

Pipeline documentation should be updated accordingly to reflect the changes to the ELAN service.

### Table of Contents

- *Enhancement to VLAN Provider Network Support*
  - *Problem description*
    - \* *Use Cases*
  - *Proposed change*
    - \* *Pipeline changes*
    - \* *Yang changes*
    - \* *Configuration impact*
    - \* *Clustering considerations*
    - \* *Other Infra considerations*
    - \* *Security considerations*
    - \* *Scale and Performance Impact*
    - \* *Targeted Release*
    - \* *Alternatives*
  - *Usage*

- \* *Features to Install*
- \* *REST API*
- \* *CLI*
- *Implementation*
  - \* *Assignee(s)*
  - \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*
- *Documentation Impact*
- *References*

### 1.1.27 Enhancement to VLAN Provider Network Support

<https://git.opendaylight.org/gerrit/#/q/topic:vlan-provider-network>

This feature aims to enhance the support for VLAN provider networks that are not of type external. As part of this enhancement, ELAN pipeline processing for the network will be done on the switch only if there is at least one VM port in the network on the switch. The behavior of VLAN provider networks of type external and flat networks will remain unchanged as of now. The optimization for external network is out of scope of this spec and will be handled as part of future releases.

#### Problem description

Current ODL implementation supports all configured VLAN segments corresponding to VLAN provider networks on a particular patch port on all Open vSwitch which are part of the network. This could have adverse performance impacts because every provider patch port will receive and processes broadcast traffic for all configured VLAN segments even in cases when the switch doesn't have a VM port in the network. Furthermore, for unknown SMACs it leads to unnecessary punts from ELAN pipeline to controller for source MAC learning from all the switches.

#### Use Cases

L2 forwarding between OVS switches using provider type VLAN over L2 segment of the underlay fabric

#### Proposed change

Instead of creating the VLAN member interface on the patch port at the time of network creation, VLAN member interface creation will be deferred until a VM port comes up in the switch in the VLAN provider network. Switch pipeline will not process broadcast traffic on this switch in a VLAN provider network until VM port is added to the network. This will be applicable to VLAN provider network without external router attribute set.

Elan service binding will also be done at the time of VLAN member interface creation. Since many neutron ports on same switch can belong to a single VLAN provider network, the flow rule should be created only once when first VM comes up and should be deleted when there are no more neutron ports in the switch for the VLAN provider network.

### Pipeline changes

None.

### Yang changes

elan:elan-instances container will be enhanced with information whether an external router is attached to VLAN provider network.

Listing 1.35: elan.yang

```
container elan-instances {
  description
    "elan instances configuration parameters. Elan instances support both the_
    ↪VLAN and VNI based elans.";

  list elan-instance {
    max-elements "unbounded";
    min-elements "0";
    key "elan-instance-name";
    description
      "Specifies the name of the elan instance. It is a string of 1 to 31
      case-sensitive characters.";
    leaf elan-instance-name {
      type string;
      description "The name of the elan-instance.";
    }
    ...

    leaf external {
      description "indicates whether the network has external router attached_
      ↪to it";
      type boolean;
      default "false";
    }
  }
}
```

### Configuration impact

None

### Clustering considerations

None



**Other Infra considerations**

N.A.

**Security considerations**

None.

**Scale and Performance Impact**

Performance will improve because of the following:

1. Switch will drop packets if it doesn't have a VM port in the VLAN on which packet is received.
2. Unnecessary punts to the controller from ELAN pipeline for source mac learning will be prevented.

**Targeted Release**

Carbon.

**Alternatives**

N.A.

**Usage****Features to Install**

This feature can be used by installing odl-netvirt-openstack. This feature doesn't add any new karaf feature.

**REST API****CLI****Implementation****Assignee(s)****Primary assignee:**

- Ravindra Nath Thakur ([ravindra.nath.thakur@ericsson.com](mailto:ravindra.nath.thakur@ericsson.com))
- Naveen Kumar Verma ([naveen.kumar.verma@ericsson.com](mailto:naveen.kumar.verma@ericsson.com))

**Other contributors:**

- Ravi Sundareswaran ([ravi.sundareswaran@ericsson.com](mailto:ravi.sundareswaran@ericsson.com))

## Work Items

N.A.

## Dependencies

This doesn't add any new dependencies.

## Testing

Capture details of testing that will need to be added.

## Unit Tests

## Integration Tests

## CSIT

## Documentation Impact

This feature will not require any change in User Guide.

## References

[1] <https://trello.com/c/A6Km6J3D/110-flat-and-vlan-network-type>

### Table of Contents

- *VNI based L2 switching, L3 forwarding and NATing*
  - *Problem description*
    - \* *In Scope*
    - \* *Out of Scope*
    - \* *Use Cases*
      - *L2 switching use cases*
      - *L3 forwarding use cases*
      - *NAT use cases*
  - *Proposed change*
    - \* *Pipeline changes*
      - *L2 Switching*
      - *Unicast*
      - *Within hypervisor*
      - *Across hypervisors*

- *Broadcast*
- *Across hypervisors*
- *L3 Forwarding*
- *Between VMs on a single OVS*
- *Between VMs on two different OVS*
- *VM sourcing the traffic (Ingress OVS)*
- *VM receiving the traffic (Egress OVS)*
- *NAT Service*
- *Inter DC*
- *SNAT*
- *DNAT*
- *Intra DC*
- *DNAT to DNAT*
- *SNAT to DNAT*
- \* *YANG changes*
- \* *Configuration impact*
- \* *Clustering considerations*
- \* *Other Infra considerations*
- \* *Security considerations*
- \* *Scale and Performance Impact*
- \* *Targeted Release(s)*
- \* *Known Limitations*
- \* *Alternatives*
- *Usage*
  - \* *Features to Install*
  - \* *REST API*
  - \* *CLI*
- *Implementation*
  - \* *Assignee(s)*
  - \* *Work Items*
- *Dependencies*
- *Testing*
  - \* *Unit Tests*
  - \* *Integration Tests*
  - \* *CSIT*

- *Documentation Impact*
- *References*

### 1.1.28 VNI based L2 switching, L3 forwarding and NATing

<https://git.opendaylight.org/gerrit/#/q/topic:vni-based-l2-l3-nat>

**Important:** All gerrit links raised for this feature will have topic name as **vni-based-l2-l3-nat**

This feature attempts to realize the use of VxLAN VNI (Virtual Network Identifier) for VxLAN tenant traffic flowing on the cloud data-network. This is applicable to L2 switching, L3 forwarding and NATing for all VxLAN based provider networks. In doing so, it eliminates the presence of `LPort tags`, `ELAN tags` and `MPLS labels` on the wire and instead, replaces them with VNIs supplied by the tenant's OpenStack.

This will be selectively done for the use-cases covered by this spec and hence, its implementation won't completely remove the usage of the above entities. The usage of `LPort tags` and `ELAN tags` within an OVS datapath (not on the wire) of the hypervisor will be retained, as eliminating it completely is a large redesign and can be pursued incrementally later.

This spec is the first step in the direction of enforcing datapath semantics that uses tenant supplied VNI values on VxLAN Type networks created by tenants in OpenStack Neutron.

**Note:** The existing L3 BGPVPN control-path and data-path semantics will continue to use L3 labels on the wire as well as inside the OVS datapaths of the hypervisor to realize both intra-dc and inter-dc connectivity.

#### Problem description

OpenDaylight NetVirt service today supports the following types of networks:

- Flat
- VLAN
- VxLAN
- GRE

Amongst these, VxLAN-based overlay is supported only for traffic within the DataCenter. External network accesses over the DC-Gateway are supported via VLAN or GRE type external networks. For rest of the traffic over the DC-Gateway, the only supported overlay is GRE.

Today, for VxLAN enabled networks by the tenant, the labels are generated by L3 forwarding service and used. Such labels are re-used for inter-DC use-cases with BGPVPN as well. This does not honor and is not in accordance with the datapath semantics from an orchestration point of view.

**This spec attempts to change the datapath semantics by enforcing the VNIs** (unique for every VxLAN enabled network in the cloud) **as dictated by the tenant's OpenStack configuration for L2 switching, L3 forwarding and NATing.**

This implementation will remove the reliance on using the following (on the wire) within the DataCenter:

- Labels for L3 forwarding
- LPort tags for L2 switching

More specifically, the traffic from source VM will be routed in source OVS by the L3VPN / ELAN pipeline. After that, the packet will travel as a switched packet in the VxLAN underlay within the DC, containing the VNI in the VxLAN header instead of MPLS label / LPort tag. In the destination OVS, the packet will be collected and sent to the destination VM through the existing ELAN pipeline.

In the nodes themselves, the LPort tag will continue to be used when pushing the packet from ELAN / L3VPN pipeline towards the VM as ACLService continues to use LPort tags.

Similarly ELAN tags will continue to be used for handling L2 broadcast packets:

- locally generated in the OVS datapath
- remotely received from another OVS datapath via internal VxLAN tunnels

LPort tag uses 8 bits and ELAN tag uses 21 bits in the metadata. The existing use of both in the metadata will remain unaffected.

## In Scope

Since VNIs are provisioned only for VxLAN based underlays, this feature has in its scope the use-cases pertaining to **intra-DC connectivity over internal VxLAN tunnels only**.

On the cloud data network wire, all the VxLAN traffic for basic L2 switching within a VxLAN network and L3 forwarding across VxLAN-type networks using routers will use tenant supplied VNI values for such VxLAN networks.

Inter-DC connectivity over external VxLAN tunnels is covered by the [EVPN\\_RT5](#) spec.

## Out of Scope

- Complete removal of use of LPort tags everywhere in ODL: Use of LPort tags within the OVS Datapath of a hypervisor, for streaming traffic to the right virtual endpoint on that hypervisor (note: not on the wire) will be retained
- Complete removal of use of ELAN tags everywhere in ODL: Use of ELAN tags within the OVS Datapath to handle local/remote L2 broadcasts (note: not on the wire) will be retained
- Complete removal of use of MPLS labels everywhere in ODL: Use of MPLS labels for realizing an L3 BGPVPN (regardless of type of networks put into such BGPVPN that may include networks of type VxLAN) both on the wire and within the OVS Datapaths will be retained.
- Addressing or testing IPv6 use-cases
- Intra DC NAT usecase where no explicit Internet VPN is created for VxLAN based external provider networks: Detailed further in Intra DC subsection in NAT section below.

Complete removal of use of LPort tags, ELAN tags and MPLS labels for VxLAN-type networks has large scale design/pipeline implications and thus need to be attempted as future initiatives via respective specs.

## Use Cases

This feature involves amendments/testing pertaining to the following:

### L2 switching use cases

1. L2 Unicast frames exchanged within an OVS datapath
2. L2 Unicast frames exchanged over OVS datapaths that are on different hypervisors
3. L2 Broadcast frames transmitted within an OVS datapath
4. L2 Broadcast frames received from remote OVS datapaths

### **L3 forwarding use cases**

1. Router realized using VNIs for networks attached to a new router (with network having pre-created VMs)
2. Router realized using VNIs for networks attached to a new router (with new VMs booted later on the network)
3. Router updated with one or more extra route(s) to an existing VM.
4. Router updated to remove previously added one/more extra routes.

### **NAT use cases**

The provider network types for external networks supported today are:

- External VLAN Provider Networks (transparent Internet VPN)
- External Flat Networks (transparent Internet VPN)
- Tenant-orchestrated Internet VPN of type GRE (actually MPLSOverGRE)

Following are the SNAT/DNAT use-cases applicable to the network types listed above:

1. SNAT functionality.
2. DNAT functionality.
3. DNAT to DNAT functionality (Intra DC)
  - FIP VM to FIP VM on same hypervisor
  - FIP VM to FIP VM on different hypervisors
4. SNAT to DNAT functionality (Intra DC)
  - Non-FIP VM to FIP VM on the same NAPT hypervisor
  - Non-FIP VM to FIP VM on the same hypervisor, but NAPT on different hypervisor
  - Non-FIP VM to FIP VM on different hypervisors (with NAPT on FIP VM hypervisor)
  - Non-FIP VM to FIP VM on different hypervisors (with NAPT on Non-FIP VM hypervisor)

### **Proposed change**

The following components within OpenDaylight Controller needs to be enhanced:

- NeutronVPN Manager
- ELAN Manager
- VPN Engine (VPN Manager, VPN Interface Manager and VPN Subnet Route Handler)
- FIB Manager
- NAT Service

## Pipeline changes

### L2 Switching

### Unicast

### Within hypervisor

There are no explicit pipeline changes for this use-case.

### Across hypervisors

- *Ingress OVS*

Instead of setting the destination LPort tag, destination network VNI will be set in the `tun_id` field in `L2_DMACH_FILTER_TABLE` (table 51) while egressing the packet on the tunnel port.

The modifications in flows and groups on the ingress OVS are illustrated below:

```
cookie=0x8000000, duration=65.484s, table=0, n_packets=23, n_bytes=2016,
→priority=4,in_port=6actions=write_metadata:0x30000000000/0xffffffff0000000001,
→goto_table:17
cookie=0x6900000, duration=63.106s, table=17, n_packets=23, n_bytes=2016,
→priority=1,metadata=0x30000000000/0xffffffff0000000000 actions=write_
→metadata:0x2000030000000000/0xfffffffffffffffffe,goto_table:40
cookie=0x6900000, duration=64.135s, table=40, n_packets=4, n_bytes=392,
→priority=61010,ip,dll_src=fa:16:3e:86:59:fd,nw_src=12.1.0.4 actions=ct(table=41,
→zone=5002)
cookie=0x6900000, duration=5112.542s, table=41, n_packets=21, n_bytes=2058,
→priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,17)
cookie=0x8040000, duration=62.125s, table=17, n_packets=15, n_bytes=854,
→priority=6,metadata=0x6000030000000000/0xffffffff0000000000 actions=write_
→metadata:0x700003138a000000/0xfffffffffffffffffe,goto_table:48
cookie=0x8500000, duration=5113.124s, table=48, n_packets=24, n_bytes=3044,
→priority=0 actions=resubmit(,49),resubmit(,50)
cookie=0x805138a, duration=62.163s, table=50, n_packets=15, n_bytes=854,
→priority=20,metadata=0x3138a000000/0xfffffffff000000,dll_src=fa:16:3e:86:59:fd
→actions=goto_table:51
cookie=0x803138a, duration=62.163s, table=51, n_packets=6, n_bytes=476,
→priority=20,metadata=0x138a000000/0xfffff000000,dll_dst=fa:16:3e:31:fb:91
→actions=set_field:**0x710**->tun_id,output:1
```

- *Egress OVS*

On the egress OVS, for the packets coming in via the internal VxLAN tunnel (OVS - OVS), `INTERNAL_TUNNEL_TABLE` currently matches on destination LPort tag for unicast packets. Since the incoming packets will now contain the network VNI in the VxLAN header, the `INTERNAL_TUNNEL_TABLE` will match on this VNI, set the ELAN tag in the metadata and forward the packet to `L2_DMACH_FILTER_TABLE` so as to reach the destination VM via the ELAN pipeline.

The modifications in flows and groups on the egress OVS are illustrated below:

```
cookie=0x8000001, duration=5136.996s, table=0, n_packets=12601, n_bytes=899766,
→priority=5,in_port=1,actions=write_metadata:0x10000000001/0xfffff00000000001,
→goto_table:36
cookie=0x9000004, duration=1145.594s, table=36, n_packets=15, n_bytes=476,
→priority=5,**tun_id=0x710,actions=write_metadata:0x138a000001/0xfffffffff000000,
→goto_table:51**
```

```

cookie=0x803138a, duration=62.163s, table=51, n_packets=9, n_bytes=576,
→priority=20,metadata=0x138a000001/0xffff000000,d1_dst=fa:16:3e:86:59:fd,
→actions=load:0x300->NXM_NX_REG6[],resubmit(,220)
cookie=0x6900000, duration=63.122s, table=220, n_packets=9, n_bytes=1160,
→priority=6,reg6=0x300actions=load:0x70000300->NXM_NX_REG6[],write_
→metadata:0x7000030000000000/0xfffffffffffffffffe,goto_table:251
cookie=0x6900000, duration=65.479s, table=251, n_packets=8, n_bytes=392,
→priority=61010,ip,d1_dst=fa:16:3e:86:59:fd,nw_dst=12.1.0.4 actions=ct(table=252,
→zone=5002)
cookie=0x6900000, duration=5112.299s, table=252, n_packets=19, n_bytes=1862,
→priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,220)
cookie=0x8000007, duration=63.123s, table=220, n_packets=8, n_bytes=1160,
→priority=7,reg6=0x70000300actions=output:6

```

## Broadcast

### Across hypervisors

The ARP broadcast by the VM will be a (local + remote) broadcast.

For the local broadcast on the VM's OVS itself, the packet will continue to get flooded to all the VM ports by setting the destination LPort tag in the local broadcast group. Hence, there are no explicit pipeline changes for when a packet is transmitted within the source OVS via a local broadcast.

The changes in pipeline for the remote broadcast are illustrated below:

- *Ingress OVS*

Instead of setting the ELAN tag, network VNI will be set in the `tun_id` field as part of bucket actions in remote broadcast group while egressing the packet on the tunnel port.

The modifications in flows and groups on the ingress OVS are illustrated below:

```

cookie=0x8000000, duration=65.484s, table=0, n_packets=23, n_bytes=2016,
→priority=4,in_port=6actions=write_metadata:0x300000000000/0xffffffff0000000001,
→goto_table:17
cookie=0x6900000, duration=63.106s, table=17, n_packets=23, n_bytes=2016,
→priority=1,metadata=0x300000000000/0xffffffff0000000000 actions=write_
→metadata:0x2000030000000000/0xfffffffffffffffffe,goto_table:40
cookie=0x6900000, duration=64.135s, table=40, n_packets=4, n_bytes=392,
→priority=61010,ip,d1_src=fa:16:3e:86:59:fd,nw_src=12.1.0.4 actions=ct(table=41,
→zone=5002)
cookie=0x6900000, duration=5112.542s, table=41, n_packets=21, n_bytes=2058,
→priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,17)
cookie=0x8040000, duration=62.125s, table=17, n_packets=15, n_bytes=854,
→priority=6,metadata=0x6000030000000000/0xffffffff0000000000 actions=write_
→metadata:0x700003138a000000/0xfffffffffffffffffe,goto_table:48
cookie=0x8500000, duration=5113.124s, table=48, n_packets=24, n_bytes=3044,
→priority=0 actions=resubmit(,49),resubmit(,50)
cookie=0x805138a, duration=62.163s, table=50, n_packets=15, n_bytes=854,
→priority=20,metadata=0x3138a000000/0xffffffffff000000,d1_src=fa:16:3e:86:59:fd,
→actions=goto_table:51
cookie=0x8030000, duration=5112.911s, table=51, n_packets=18, n_bytes=2568,
→priority=0 actions=goto_table:52
cookie=0x870138a, duration=62.163s, table=52, n_packets=9, n_bytes=378,
→priority=5,metadata=0x138a000000/0xffff000001 actions=write_
→actions(group:210004)

```



```
group_id=210004,type=all,bucket=actions=group:210003,bucket=actions=set_
↳field:**0x710**->tun_id,output:1
```

- *Egress OVS*

On the egress OVS, for the packets coming in via the internal VxLAN tunnel (OVS - OVS), INTERNAL\_TUNNEL\_TABLE currently matches on ELAN tag for broadcast packets. Since the incoming packets will now contain the network VNI in the VxLAN header, the INTERNAL\_TUNNEL\_TABLE will match on this VNI, set the ELAN tag in the metadata and forward the packet to L2\_DMACH\_FILTER\_TABLE to be broadcasted via the local broadcast groups traversing the ELAN pipeline.

The TUNNEL\_INGRESS\_BIT being set in the CLASSIFIER\_TABLE (table 0) ensures that the packet is always sent to the local broadcast group only and hence, remains within the OVS. This is necessary to avoid switching loop back to the source OVS.

The modifications in flows and groups on the egress OVS are illustrated below:

```
cookie=0x8000001, duration=5136.996s, table=0, n_packets=12601, n_bytes=899766,
↳priority=5,in_port=1,actions=write_metadata:0x1000000001/0xffffffff000000001,
↳goto_table:36
cookie=0x9000004, duration=1145.594s, table=36, n_packets=15, n_bytes=476,
↳priority=5,**tun_id=0x710,actions=write_metadata:0x138a00001/0xffffffff000000,
↳goto_table:51**
cookie=0x8030000, duration=5137.609s, table=51, n_packets=9, n_bytes=1293,
↳priority=0 actions=goto_table:52
cookie=0x870138a, duration=1145.592s, table=52, n_packets=0, n_bytes=0,
↳priority=5,metadata=0x138a00001/0xffff00001 actions=apply_
↳actions (group:210003)

group_id=210003,type=all,bucket=actions=set_field:0x4->tun_id,resubmit(,55)

cookie=0x8800004, duration=1145.594s, table=55, n_packets=9, n_bytes=378,
↳priority=9,tun_id=0x4,actions=load:0x400->NXM_NX_REG6[],resubmit(,220)
cookie=0x6900000, duration=63.122s, table=220, n_packets=9, n_bytes=1160,
↳priority=6,reg6=0x300actions=load:0x70000300->NXM_NX_REG6[],write_
↳metadata:0x7000030000000000/0xfffffffffffffffffe,goto_table:251
cookie=0x6900000, duration=65.479s, table=251, n_packets=8, n_bytes=392,
↳priority=61010,ip,d1_dst=fa:16:3e:86:59:fd,nw_dst=12.1.0.4 actions=ct(table=252,
↳zone=5002)
cookie=0x6900000, duration=5112.299s, table=252, n_packets=19, n_bytes=1862,
↳priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,220)
cookie=0x8000007, duration=63.123s, table=220, n_packets=8, n_bytes=1160,
↳priority=7,reg6=0x70000300actions=output:6
```

The ARP response will be a unicast packet, and as indicated above, for unicast packets, there are no explicit pipeline changes.

## L3 Forwarding

### Between VMs on a single OVS

There are no explicit pipeline changes for this use-case. The destination LPort tag will continue to be set in the nexthop group since when The EGRESS\_DISPATCHER\_TABLE sends the packet to EGRESS\_ACL\_TABLE, it is used by the ACL service.

## Between VMs on two different OVS

L3 forwarding between VMs on two different hypervisors is asymmetric forwarding since the traffic is routed in the source OVS datapath while it is switched over the wire and then all the way to the destination VM on the destination OVS datapath.

### VM sourcing the traffic (Ingress OVS)

L3\_FIB\_TABLE will set the destination network VNI in the `tun_id` field instead of the MPLS label.

```
CLASSIFIER_TABLE => DISPATCHER_TABLE => INGRESS_ACL_TABLE =>
DISPATCHER_TABLE => L3_GW_MAC_TABLE =>
L3_FIB_TABLE (set destination MAC, **set tunnel-ID as destination network VNI**)
=> Output to tunnel port
```

The modifications in flows and groups on the ingress OVS are illustrated below:

```
cookie=0x8000000, duration=128.140s, table=0, n_packets=25, n_bytes=2716, priority=4,
->in_port=5 actions=write_metadata:0x500000000000/0xffffffff0000000001, goto_table:17
cookie=0x8000000, duration=4876.599s, table=17, n_packets=0, n_bytes=0, priority=0,
->metadata=0x5000000000000000/0xf000000000000000 actions=write_
->metadata:0x6000000000000000/0xf000000000000000, goto_table:80
cookie=0x1030000, duration=4876.563s, table=80, n_packets=0, n_bytes=0, priority=0,
->actions=resubmit(,17)
cookie=0x6900000, duration=123.870s, table=17, n_packets=25, n_bytes=2716, priority=1,
->metadata=0x500000000000/0xffffffff0000000000 actions=write_metadata:0x2000050000000000/
->0xffffffffffffffffffe, goto_table:40
cookie=0x6900000, duration=126.056s, table=40, n_packets=15, n_bytes=1470,
->priority=61010, ip, dl_src=fa:16:3e:63:ea:0c, nw_src=10.1.0.4 actions=ct(table=41,
->zone=5001)
cookie=0x6900000, duration=4877.057s, table=41, n_packets=17, n_bytes=1666,
->priority=62020, ct_state=-new+est-rel-inv+trk actions=resubmit(,17)
cookie=0x6800001, duration=123.485s, table=17, n_packets=28, n_bytes=3584, priority=2,
->metadata=0x2000050000000000/0xffffffff0000000000 actions=write_
->metadata:0x5000050000000000/0xffffffffffffffffffe, goto_table:60
cookie=0x6800000, duration=3566.900s, table=60, n_packets=24, n_bytes=2184,
->priority=0 actions=resubmit(,17)
cookie=0x8000001, duration=123.456s, table=17, n_packets=17, n_bytes=1554, priority=5,
->metadata=0x5000050000000000/0xffffffff0000000000 actions=write_
->metadata:0x60000500000222e0/0xffffffffffffffffffe, goto_table:19
cookie=0x8000009, duration=124.815s, table=19, n_packets=15, n_bytes=1470,
->priority=20, metadata=0x222e0/0xfffffffffe, dl_dst=fa:16:3e:51:da:ee actions=goto_
->table:21
cookie=0x8000003, duration=125.568s, table=21, n_packets=9, n_bytes=882, priority=42,
->ip, metadata=0x222e0/0xfffffffffe, nw_dst=12.1.0.3 actions=**set_field:0x710->tun_id**,
->set_field:fa:16:3e:31:fb:91->eth_dst, output:1
```

### VM receiving the traffic (Egress OVS)

On the egress OVS, for the packets coming in via the VxLAN tunnel, INTERNAL\_TUNNEL\_TABLE currently matches on MPLS label and sends it to the nexthop group to be taken to the destination VM via EGRESS\_ACL\_TABLE. Since the incoming packets will now contain network VNI in the VxLAN header, the INTERNAL\_TUNNEL\_TABLE will match on the VNI, set the ELAN tag in the metadata and forward the packet to L2\_DMATCH\_FILTER\_TABLE, from where it will be taken to the destination VM via the ELAN pipeline.

```

CLASSIFIER_TABLE => INTERNAL_TUNNEL_TABLE (Match on network VNI, set ELAN tag in the
↳ metadata)
=> L2_DMATCH_FILTER_TABLE (Match on destination MAC) => EGRESS_DISPATCHER_TABLE
=> EGRESS_ACL_TABLE => Output to destination VM port

```

The modifications in flows and groups on the egress OVS are illustrated below:

```

cookie=0x80000001, duration=4918.647s, table=0, n_packets=12292, n_bytes=877616,
↳ priority=5, in_port=1 actions=write_metadata:0x10000000001/0xffffffff000000001, goto_
↳ table:36
cookie=0x90000004, duration=927.245s, table=36, n_packets=8234, n_bytes=52679,
↳ priority=5, **tun_id=0x710, actions=write_metadata:0x138a000001/0xffffffff000000,
↳ goto_table:51**
cookie=0x803138a, duration=62.163s, table=51, n_packets=9, n_bytes=576, priority=20,
↳ metadata=0x138a000001/0xffffffff000000, dl_dst=fa:16:3e:86:59:fd actions=load:0x300->NXM_
↳ NX_REG6[], resubmit(,220)
cookie=0x6900000, duration=63.122s, table=220, n_packets=9, n_bytes=1160, priority=6,
↳ reg6=0x300 actions=load:0x70000300->NXM_NX_REG6[], write_metadata:0x7000030000000000/
↳ 0xffffffffffffffffffe, goto_table:251
cookie=0x6900000, duration=65.479s, table=251, n_packets=8, n_bytes=392,
↳ priority=61010, ip, dl_dst=fa:16:3e:86:59:fd, nw_dst=12.1.0.4 actions=ct(table=252,
↳ zone=5002)
cookie=0x6900000, duration=5112.299s, table=252, n_packets=19, n_bytes=1862,
↳ priority=62020, ct_state=-new+est-rel-inv+trk actions=resubmit(,220)
cookie=0x80000007, duration=63.123s, table=220, n_packets=8, n_bytes=1160, priority=7,
↳ reg6=0x70000300 actions=output:6

```

## NAT Service

For NAT, we need VNIs to be used in two scenarios:

- When packet is forwarded from non-NAPT to NAPT hypervisor (VNI per router)
- Between hypervisors (intra DC) over Internet VPN (VNI per Internet VPN)

Hence, a pool titled `opendaylight-vni-ranges`, non-overlapping with the OpenStack Neutron `vni_ranges` configuration, needs to be configured by the OpenDaylight Controller Administrator.

This `opendaylight-vni-ranges` pool will be used to carve out a unique VNI per router to be then used in the datapath for traffic forwarding from non-NAPT to NAPT switch for this router.

Similarly, for MPLSOverGRE based external networks, the `opendaylight-vni-ranges` pool will be used to carve out a unique VNI per Internet VPN (GRE-provider-type) to be then used in the datapath for traffic forwarding for SNAT-to-DNAT and DNAT-to-DNAT cases within the DataCenter. Only one external network can be associated to Internet VPN today and this spec doesn't attempt to address that limitation.

A NeutronVPN configuration API will be exposed to the administrator to configure the lower and higher limit for this pool. If the administrator doesn't configure this explicitly, then the pool will be created with default values of lower limit set to 70000 and upper limit set to 100000, during the first NAT session configuration.

**FIB Manager changes:** For external network of type GRE, it is required to use Internet VPN VNI for intra-DC communication, but we still require MPLS labels to reach SNAT/DNAT VMs from external entities via MPLSOverGRE. Hence, we will make use of the `l3vni` attribute added to `fibEntries` container as part of `EVPN_RT5` spec. NAT will populate both `label` and `l3vni` values for `fibEntries` created for floating-ips and external-fixed-ips with external network of type GRE. This `l3vni` value will be used while programming remote FIB flow entries (on all the switches which are part of the same VRF). But still, MPLS label will be used to advertise prefixes and in `L3_LFIB_TABLE` taking the packet to `INBOUND_NAPT_TABLE` and `PDNAT_TABLE`.

For SNAT/DNAT use-cases, we have following provider network types for External Networks:

1. VLAN - not VNI based
2. Flat - not VNI based
3. VxLAN - VNI based (covered by the [EVPN\\_RT5](#) spec)
4. GRE - not VNI based (will continue to use MPLS labels)

## Inter DC

### SNAT

- From a VM on a NAPT switch to reach Internet, and reverse traffic reaching back to the VM

There are no explicit pipeline changes.

- From a VM on a non-NAPT switch to reach Internet, and reverse traffic reaching back to the VM

On the non-NAPT switch, PSNAT\_TABLE (table 26) will be set with `tun_id` field as Router Based VNI allocated from the pool and send to group to reach NAPT switch.

On the NAPT switch, INTERNAL\_TUNNEL\_TABLE (table 36) will match on the `tun_id` field which will be Router Based VNI and send the packet to OUTBOUND\_NAPT\_TABLE (table 46) for SNAT Translation and to be taken to Internet.

#### – Non-NAPT switch

```
cookie=0x8000006, duration=2797.179s, table=26, n_packets=47, n_bytes=3196,
↳priority=5, ip, metadata=0x23a50/0xffffffff actions=**set_field:0x710->tun_
↳id**, group:202501

group_id=202501, type=all, bucket=actions=output:1
```

#### – NAPT switch

```
cookie=0x8000001, duration=4918.647s, table=0, n_packets=12292, n_
↳bytes=877616, priority=5, in_port=1, actions=write_metadata:0x10000000001/
↳0xffffffff0000000001, goto_table:36
cookie=0x9000004, duration=927.245s, table=36, n_packets=8234, n_bytes=52679,
↳priority=10, ip, **tun_id=0x710**, actions=write_metadata:0x23a50/0xffffffff,
↳goto_table:46
```

As part of the response from NAPT switch, the packet will be taken to the Non-NAPT switch after SNAT reverse translation using destination VMs Network VNI.

### DNAT

There is no NAT specific explicit pipeline change for DNAT traffic to DC-gateway.

## Intra DC

- VLAN Provider External Networks: VNI is not applicable on the external VLAN Provider network. However, the Router VNI will be used for datapath traffic from non-NAPT switch to NAPT-switch over the internal VxLAN tunnel.

- VxLAN Provider External Networks:
  - **Explicit creation of Internet VPN:** An L3VNI, mandatorily falling within the `opendaylight-vni-ranges`, will be provided by the Cloud admin (or tenant). This VNI will be used uniformly for all packet transfer over the VxLAN wire for this Internet VPN (uniformly meaning all the traffic on Internal or External VXLAN Tunnel, except the non-NAPT to NAPT communication). This usecase is covered by [EVPN\\_RT5](#) spec
  - **No explicit creation of Internet VPN:** A transparent Internet VPN having UUID same as that of the corresponding external network UUID is created implicitly and the VNI configured for this external network should be used on the VxLAN wire. This usecase is **out of scope** from the perspective of this spec, and the same is indicated in [Out of Scope](#) section.
- GRE Provider External Networks: Internet VPN VNI will be carved per Internet VPN using `opendaylight-vni-ranges` to be used on the wire.

## DNAT to DNAT

- FIP VM to FIP VM on different hypervisors

After DNAT translation on the first hypervisor DNAT-OVS-1, the traffic will be sent to the `L3_FIB_TABLE` (table=21) in order to reach the floating IP VM on the second hypervisor DNAT-OVS-2. Here, the `tun_id` action field will be set as the `INTERNET VPN VNI` value.

### – DNAT-OVS-1

```
cookie=0x80000003, duration=518.567s, table=21, n_packets=0, n_bytes=0,
↳priority=42, ip, metadata=0x222e8/0xffffffff, nw_dst=172.160.0.200,
↳actions=**set_field:0x11178->tun_id**, output:9
```

### – DNAT-OVS-2

```
cookie=0x9011177, duration=411685.075s, table=36, n_packets=2, n_bytes=196,
↳priority=**6**, **tun_id=0x11178**actions=resubmit(,25)
cookie=0x9011179, duration=478573.171s, table=36, n_packets=2, n_bytes=140,
↳priority=5, **tun_id=0x11178**, actions=goto_table:44

cookie=0x80000004, duration=408145.805s, table=25, n_packets=600, n_
↳bytes=58064, priority=10, ip, nw_dst=172.160.0.100, **eth_
↳dst=fa:16:3e:e6:e3:c6** actions=set_field:10.0.0.5->ip_dst, write_
↳metadata:0x222e0/0xffffffff, goto_table:27
cookie=0x80000004, duration=408145.805s, table=25, n_packets=600, n_
↳bytes=58064, priority=10, ipactions=goto_table:44
```

First, the `INTERNAL_TUNNEL_TABLE` (table=36) will take the packet to the `PDNAT_TABLE` (table 25) for an exact FIP match in `PDNAT_TABLE`.

- In case of a successful FIP match, `PDNAT_TABLE` will further match on floating IP MAC. This is done as a security prerogative since in DNAT usecases, the packet can land to the hypervisor directly from the external world. Hence, better to have a second match criteria.
- In case of no match, the packet will be redirected to the SNAT pipeline towards the `INBOUND_NAPT_TABLE` (table=44). This is the use-case where DNAT-OVS-2 also acts as the NAPT switch.

In summary, on an given NAPT switch, if both DNAT and SNAT are configured, the incoming traffic will first be sent to the `PDNAT_TABLE` and if there is no FIP match found, then it will be forwarded to `INBOUND_NAPT_TABLE` for SNAT translation.

As part of the response, the Internet VPN VNI will be used as `tun_id` to reach floating IP VM on DNAT-OVS-1.

- FIP VM to FIP VM on same hypervisor

The pipeline changes will be similar as are for different hypervisors, the only difference being that `INTERNAL_TUNNEL_TABLE` will never be hit in this case.

## SNAT to DNAT

- Non-FIP VM to FIP VM on different hypervisors (with NAPT elected as the FIP VM hypervisor)

The packet will be sent to the NAPT hypervisor from non-FIP VM (for SNAT translation) using Router VNI (similar to as described in [SNAT](#) section). As part of the response from the NAPT switch after SNAT reverse translation, the packet is forwarded to non-FIP VM using destination VM's Network VNI.

- Non-FIP VM to FIP VM on the same NAPT hypervisor

There are no explicit pipeline changes for this use-case.

- Non-FIP VM to FIP VM on the same hypervisor, but a different hypervisor elected as NAPT switch

### – NAPT hypervisor

The packet will be sent to the NAPT hypervisor from non-FIP VM (for SNAT translation) using Router VNI (similar to as described in [SNAT](#) section). On the NAPT switch, the `INTERNAL_TUNNEL_TABLE` will match on the Router VNI in the `tun_id` field and send the packet to `OUTBOUND_NAPT_TABLE` for SNAT translation (similar to as described in [SNAT](#) section).

```
cookie=0x8000005, duration=5073.829s, table=36, n_packets=61, n_bytes=4610,
↳priority=10, ip, **tun_id=0x11170**, actions=write_metadata:0x222e0/0xfffffffffe,
↳goto_table:46
```

The packet will later be sent back to the FIP VM hypervisor from `L3_FIB_TABLE` with `tun_id` field set as the Internet VPN VNI.

```
cookie=0x8000003, duration=518.567s, table=21, n_packets=0, n_bytes=0,
↳priority=42, ip, metadata=0x222e8/0xfffffffffe, nw_dst=172.160.0.200,
↳actions=**set_field:0x11178->tun_id**, output:9
```

### – FIP VM hypervisor

On reaching the FIP VM Hypervisor, the packet will be sent for DNAT translation. The `INTERNAL_TUNNEL_TABLE` will match on the Internet VPN VNI in the `tun_id` field and send the packet to `PDNAT_TABLE`.

```
cookie=0x9011177, duration=411685.075s, table=36, n_packets=2, n_bytes=196,
↳priority=**6**, **tun_id=0x11178**, actions=resubmit(,25)
cookie=0x8000004, duration=408145.805s, table=25, n_packets=600, n_
↳bytes=58064, priority=10, ip, nw_dst=172.160.0.100, **eth_
↳dst=fa:16:3e:e6:e3:c6** actions=set_field:10.0.0.5->ip_dst, write_
↳metadata:0x222e0/0xfffffffffe, goto_table:27
```

Upon FIP VM response, DNAT reverse translation happens and traffic is sent back to the NAPT switch for SNAT translation. The `L3_FIB_TABLE` will be set with Internet VPN VNI in the `tun_id` field.

```
cookie=0x8000003, duration=95.300s, table=21, n_packets=2, n_bytes=140,
↳priority=42, ip, metadata=0x222ea/0xfffffffffe, nw_dst=172.160.0.3 actions=**set_
↳field:0x11178->tun_id**, output:5
```

- *NAPT hypervisor*

On NAPT hypervisor, the `INTERNAL_TUNNEL_TABLE` will match on the `Internet VPN VNI` in the `tun_id` field and send the packet to “`INBOUND_NAPT_TABLE`” for SNAT reverse translation (external fixed IP to VM IP). The packet will then be sent back to the non-FIP VM using destination VM’s Network VNI.

- Non-FIP VM to FIP VM on different hypervisors (with NAPT elected as the non-FIP VM hypervisor)

After SNAT Translation, `Internet VPN VNI` will be used to reach FIP VM. On FIP VM hypervisor, the `INTERNAL_TUNNEL_TABLE` will take the packet to the `PDNAT_TABLE` to match on `Internet VPN VNI` in the `tun_id` field for DNAT translation.

Upon response from FIP, DNAT reverse translation happens and uses `Internet VPN VNI` to reach back to the non-FIP VM.

## YANG changes

- `opendaylight-vni-ranges` and `enforce-openstack-semantic` leaf elements will be added to `neutronvpn-config` container in `neutronvpn-config.yang`:
  - `opendaylight-vni-ranges` will be introduced to accept inputs for the VNI range pool from the configurator via the corresponding exposed REST API. In case this is not defined, the default value defined in `netvirt-neutronvpn-config.xml` will be used to create this pool.
  - `enforce-openstack-semantic` will be introduced to have the flexibility to enable or disable OpenStack semantics in the dataplane for this feature. It will be defaulted to true, meaning these semantics will be enforced by default. In case it is set to false, the dataplane will continue to be programmed with LPort tags / ELAN tags for switching and with labels for routing use-cases. Once this feature gets stabilized and the semantics are in place to use VNIs on the wire for BGPVPN based forwarding too, this config can be permanently removed if deemed fit.

Listing 1.36: `neutronvpn-config.yang`

```
container neutronvpn-config {
    config true;
    ...
    ...
    leaf opendaylight-vni-ranges {
        type string;
        default "70000:99999";
    }
    leaf enforce-openstack-semantic {
        type boolean;
        default true;
    }
}
```

- Provider network-type and provider segmentation-ID need to be propagated to FIB Manager to manipulate flows based on the same. Hence:
  - A new grouping `network-attributes` will be introduced in `neutronvpn.yang` to hold network type and segmentation ID. This grouping will replace the leaf-node `network-id` in subnetmaps MD-SAL configuration datastore:

Listing 1.37: neutronvpn.yang

```

grouping network-attributes {
    leaf network-id {
        type yang:uuid;
        description "UUID representing the network";
    }
    leaf network-type {
        type enumeration {
            enum "FLAT";
            enum "VLAN";
            enum "VXLAN";
            enum "GRE";
        }
    }
    leaf segmentation-id {
        type uint32;
        description "Optional. Isolated segment on the physical network.
            If segment-type is vlan, this ID is a vlan identifier.
            If segment-type is vxlan, this ID is a vni.
            If segment-type is flat/gre, this ID is set to 0";
    }
}

container subnetmaps {
    ...
    ...
    uses network-attributes;
}

```

- These attributes will be propagated upon addition of a router-interface or addition of a subnet to a BGPVPN to VPN Manager module via the `subnet-added-to-vpn` notification modelled in `neutronvpn.yang`. Hence, the following node will be added:

Listing 1.38: neutronvpn.yang

```

notification subnet-added-to-vpn {
    description "new subnet added to vpn";
    ...
    ...
    uses network-attributes;
}

```

- `VpnSubnetRouteHandler` will act on these notifications and store these attributes in `subnet-op-data` MD-SAL operational datastore as described below. FIB Manager will get to retrieve the `subnet ID` from the primary adjacency of the concerned VPN interface. This `subnet ID` will be used as the key to retrieve `network-attributes` from `subnet-op-data` datastore.

Listing 1.39: odl-l3vpn.yang

```

import neutronvpn {
    prefix nvpn;
    revision-date "2015-06-02";
}

container subnet-op-data {
    ...
    ...
}

```



```

    uses nvpn:network-attributes;
}

```

- `subnetID` and `nat-prefix` leaf elements will be added to `prefix-to-interface` container in `odl-l3vpn.yang`:
  - For NAT use-cases where the VRF entry is not always associated with a VPN interface (eg. for NAT entries such as floating IP and router-gateway-IPs for external VLAN / flat networks), `subnetID` leaf element will be added to make it possible to retrieve the `network-attributes`.
  - To distinguish a non-NAT prefix from a NAT prefix, `nat-prefix` leaf element will be added. This is a boolean attribute indicating whether the prefix is a NAT prefix (meaning a floating IP, or an external-fixed-ip of a router-gateway). The VRFEntry corresponding to the NAT prefix entries here may carry both the MPLS label and the Internet VPN VNI. For SNAT-to-DNAT within the datacenter, where the Internet VPN contains an MPLSOverGRE based external network, this VRF entry will publish the MPLS label to BGP while the Internet VPN VNI (also known as L3VNI) will be used to carry intra-DC traffic on the external segment within the datacenter.

Listing 1.40: odl-l3vpn.yang

```

container prefix-to-interface {
    config false;
    list vpn-ids {
        key vpn-id;
        leaf vpn-id {type uint32;}
        list prefixes {
            key ip_address;
            ...
            ...
            leaf subnet-id {
                type yang:uuid;
            }
            leaf nat-prefix {
                type boolean;
                default false;
            }
        }
    }
}

```

## Configuration impact

- We have to make sure that we do not accept configuration of VxLAN type provider networks without the `segmentation-ID` available in them since we are using it to represent the VNI on the wire and in the flows/groups.

## Clustering considerations

No specific additional clustering considerations to be adhered to.

## Other Infra considerations

None.

## Security considerations

None.

## Scale and Performance Impact

None.

## Targeted Release(s)

Carbon.

## Known Limitations

None.

## Alternatives

N.A.

## Usage

### Features to Install

odl-netvirt-openstack

### REST API

No new changes to the existing REST APIs.

### CLI

No new CLI is being added.

## Implementation

### Assignee(s)

**Primary assignee:** Abhinav Gupta <abhinav.gupta@ericsson.com> Vivekanandan Narasimhan  
<n.vivekanandan@ericsson.com>

**Other contributors:** Chetan Arakere Gowdru <chetan.arakere@altencalsoftlabs.com> Karthikeyan Krishnan  
<karthikeyan.k@altencalsoftlabs.com> Yugandhar Sarraju <yugandhar.s@altencalsoftlabs.com>

## Work Items

Trello card: <https://trello.com/c/PfARbEmU/84-enforce-vni-on-the-wire-for-l2-switching-l3-forwarding-and-nating-on-vxlan-overlay>

1. Code changes to alter the pipeline and e2e testing of the use-cases mentioned.
2. Add Documentation

## Dependencies

This doesn't add any new dependencies.

## Testing

### Unit Tests

Appropriate UTs will be added for the new code coming in once framework is in place.

### Integration Tests

There won't be any Integration tests provided for this feature.

## CSIT

No new testcases to be added, existing ones should continue to succeed.

## Documentation Impact

This will require changes to the Developer Guide.

Developer Guide needs to capture how this feature modifies the existing Netvirt L3 forwarding service implementation.

## References

- <http://docs.opendaylight.org/en/latest/documentation.html>
- [https://wiki.opendaylight.org/view/Genius:Carbon\\_Release\\_Plan](https://wiki.opendaylight.org/view/Genius:Carbon_Release_Plan)
- EVPN\_RT5



## CHAPTER 2

---

### NetVirt Developer Guide

---



## CHAPTER 3

---

### NetVirt Installation Guide

---





## 4.1 OpenStack with NetVirt

### Table of Contents

- *OpenStack with NetVirt*
  - *Installing OpenDaylight on an existing OpenStack*
  - *Installing OpenStack and OpenDaylight using DevStack*
  - *Troubleshooting*
  - *Useful Links*

**Prerequisites:** OpenDaylight requires Java 1.8.0 and Open vSwitch >= 2.5.0

### 4.1.1 Installing OpenDaylight on an existing OpenStack

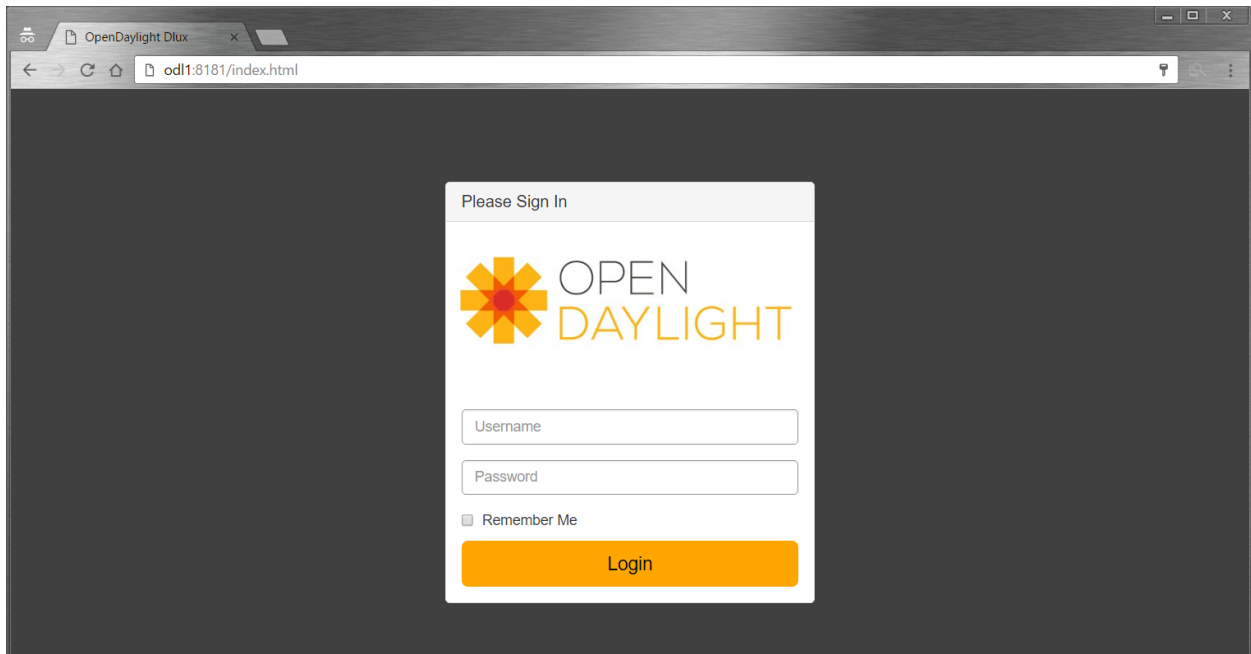
- On the control host, [Download the latest OpenDaylight release](#)
- Uncompress it as root, and start OpenDaylight (you can start OpenDaylight by running karaf directly, but exiting from the shell will shut it down):

```
tar xvfz distribution-karaf-0.5.1-Boron-SR1.tar.gz
cd distribution-karaf-0.5.1-Boron-SR1
./bin/start # Start OpenDaylight as a server process
```

- Connect to the Karaf shell, and install the odl-netvirt-openstack bundle, dlux and their dependencies:

```
./bin/client # Connect to OpenDaylight with the client
opendaylight-user@root> feature:install odl-netvirt-openstack odl-dlux-core odl-
↪mdsal-apidocs
```

- If everything is installed correctly, you should now be able to log in to the dlux interface on [http://CONTROL\\_HOST:8181/index.html](http://CONTROL_HOST:8181/index.html) - the default username and password is “admin/admin” (see screenshot below)



### Optional - Advanced OpenDaylight Installation - Configurations and Clustering

- ACL Implementation - Security Groups - Stateful:
  - Default implementation used is stateful, requiring OVS compiled with conntrack modules.
  - This requires using a linux kernel that is  $\geq 4.3$
  - To check if OVS is running with conntrack support:

```
root@devstack:~/# lsmod | grep conntrack | grep openvswitch
nf_conntrack          106496  9 xt_CT,openvswitch,nf_nat,nf_nat_ipv4,xt_
↪conntrack,nf_conntrack_netlink,xt_connmark,nf_conntrack_ipv4,nf_conntrack_
↪ipv6
```

- If the conntrack modules are not installed for OVS, either recompile/install an OVS version with conntrack support, or alternatively configure OpenDaylight to use a non-stateful implementation.
- OpenvSwitch 2.5 with conntrack support can be acquired from this repository for yum based linux distributions:

```
yum install -y http://rdoproject.org/repos/openstack-newton/rdo-release-
↪newton.rpm
yum install -y --nogpgcheck openvswitch
```

- ACL Implementations - Alternative options:
  - “learn” - semi-stateful implementation that does not require conntrack support. This is the most complete non-conntrack implementation.
  - “stateless” - naive security group implementation for TCP connections only. UDP and ICMP packets are allowed by default.

- “transparent” - no security group support. all traffic is allowed, this is the recommended mode if you don’t need to use security groups at all.
- To configure one of these alternative implementations, the following needs to be done prior to running OpenDaylight:

```
mkdir -p <ODL_FOLDER>/etc/.opendaylight/datastore/initial/config/
export CONFFILE=`find <ODL_FOLDER> -name "\*aclservice\*config.xml"`
cp \CONFFILE <ODL_FOLDER>/etc/.opendaylight/datastore/initial/config/netvirt-
↪aclservice-config.xml
sed -i s/stateful/<learn/transparent>/ <ODL_FOLDER>/etc/.opendaylight/
↪datastore/initial/config/netvirt-aclservice-config.xml
cat <ODL_FOLDER>/etc/.opendaylight/datastore/initial/config/netvirt-aclservice-
↪config.xml
```

- Running multiple OpenDaylight controllers in a cluster:
  - For redundancy, it is possible to run OpenDaylight in a 3-node cluster.
  - More info on Clustering available [here](#).
  - To configure OpenDaylight in clustered mode, run <ODL\_FOLDER>/bin/configure\_cluster.sh on each node prior to running OpenDaylight. This script is used to configure cluster parameters on this controller. The user should restart controller to apply changes.

```
Usage: ./configure_cluster.sh <index> <seed_nodes_list>
- index: Integer within 1..N, where N is the number of seed nodes.
- seed_nodes_list: List of seed nodes, separated by comma or space.
```

- The address at the provided index should belong this controller. When running this script on multiple seed nodes, keep the seed\_node\_list same, and vary the index from 1 through N.
- Optionally, shards can be configured in a more granular way by modifying the file “custom\_shard\_configs.txt” in the same folder as this tool. Please see that file for more details.

---

**Note:** OpenDaylight should be restarted after applying any of the above changes via configuration files.

---

## Ensuring OpenStack network state is clean

When using OpenDaylight as the Neutron back-end, OpenDaylight expects to be the only source of truth for Neutron configurations. Because of this, it is necessary to remove existing OpenStack configurations to give OpenDaylight a clean slate.

- Delete instances:

```
nova list
nova delete <instance names>
```

- Remove links from subnets to routers:

```
neutron subnet-list
neutron router-list
neutron router-port-list <router name>
neutron router-interface-delete <router name> <subnet ID or name>
```

- Delete subnets, networks, routers:

```
neutron subnet-delete <subnet name>
neutron net-list
neutron net-delete <net name>
neutron router-delete <router name>
```

- Check that all ports have been cleared - at this point, this should be an empty list:

```
neutron port-list
```

## Ensure Neutron is stopped

While Neutron is managing the OVS instances on compute and control nodes, OpenDaylight and Neutron can be in conflict. To prevent issues, we turn off Neutron server on the network controller, and Neutron's Open vSwitch agents on all hosts.

- Turn off neutron-server on control node:

```
systemctl stop neutron-server
systemctl stop neutron-l3-agent
```

- On each node in the cluster, shut down and disable Neutron's agent services to ensure that they do not restart after a reboot:

```
systemctl stop neutron-openvswitch-agent
systemctl disable
neutron-openvswitch-agent
systemctl stop neutron-l3-agent
systemctl disable neutron-l3-agent
```

## Configuring Open vSwitch to be managed by OpenDaylight

On each host (both compute and control nodes) we will clear the pre-existing Open vSwitch config and set OpenDaylight to manage the switch:

- Stop the Open vSwitch service, and clear existing OVSDb (OpenDaylight expects to manage vSwitches completely):

```
systemctl stop openvswitch
rm -rf /var/log/openvswitch/*
rm -rf /etc/openvswitch/conf.db
systemctl start openvswitch
```

- At this stage, your Open vSwitch configuration should be empty:

```
[root@odl-compute2 ~]# ovs-vsctl show
9f3b38cb-eefc-4bc7-828b-084b1f66fbfd
    ovs_version: "2.5.1"
```

- Set OpenDaylight as the manager on all nodes:

```
ovs-vsctl set-manager tcp:{CONTROL_HOST}:6640
```

- Set the IP to be used for VXLAN connectivity on all nodes. This IP must correspond to an actual linux interface on each machine.

```
sudo ovs-vsctl set Open_vSwitch . other_config:local_ip=<ip>
```

- You should now see a new section in your Open vSwitch configuration showing that you are connected to the OpenDaylight server via OVSDB, and OpenDaylight will automatically create a br-int bridge that is connected via OpenFlow to the controller:

```
[root@odl-compute2 ~]# ovs-vsctl show
9f3b38cb-eefc-4bc7-828b-084b1f66fbfd
    Manager "tcp:172.16.21.56:6640"
        is_connected: true
    Bridge br-int
        Controller "tcp:172.16.21.56:6633"
            is_connected: true
        fail_mode: secure
        Port br-int
            Interface br-int
    ovs_version: "2.5.1"

[root@odl-compute2 ~]# ovs-vsctl get Open_vSwitch . other_config
{local_ip="10.0.42.161"}
```

- If you do not see the result above (specifically, if you do not see “is\_connected: true” in the Manager section or in the Controller section), you may not have a security policies in place to allow Open vSwitch remote administration.

---

#### Note:

There might be iptables restrictions - if so the relevant ports should be opened (6640, 6653).

If SELinux is running on your linux, set to permissive mode on all nodes and ensure it stays that way after boot.

```
setenforce 0
sed -i -e 's/SELINUX=enforcing/SELINUX=permissive/g' /etc/selinux/config
```

- Make sure all nodes, including the control node, are connected to OpenDaylight.
- If you reload DLUX, you should now see that all of your Open vSwitch nodes are now connected to OpenDaylight.
- If something has gone wrong, check data/log/karaf.log under the OpenDaylight distribution directory. If you do not see any interesting log entries, set logging for netvirt to TRACE level inside Karaf and try again:

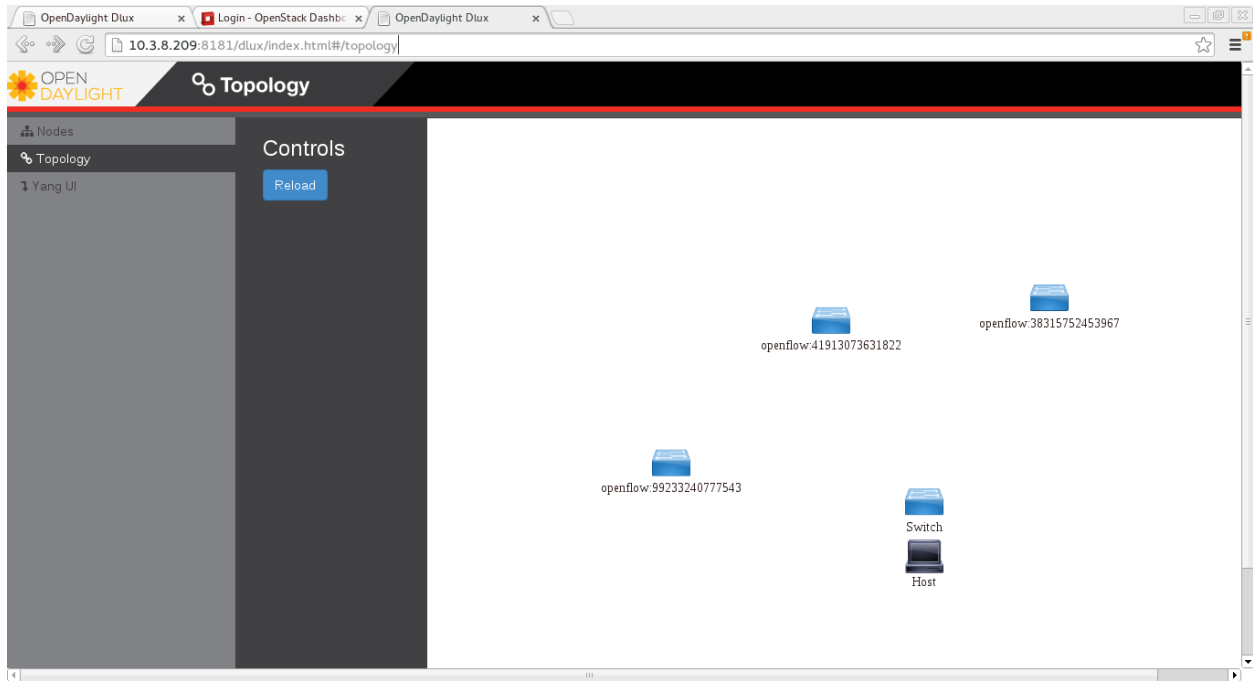
```
log:set TRACE netvirt
```

## Configuring Neutron to use OpenDaylight

Once you have configured the vSwitches to connect to OpenDaylight, you can now ensure that OpenStack Neutron is using OpenDaylight.

This requires the neutron networking-odl module to be installed. `| pip install networking-odl`

First, ensure that port 8080 (which will be used by OpenDaylight to listen for REST calls) is available. By default, swift-proxy-service listens on the same port, and you may need to move it (to another port or another host), or disable that service. It can be moved to a different port (e.g. 8081) by editing /etc/swift/proxy-server.conf and /etc/cinder/cinder.conf, modifying iptables appropriately, and restarting swift-proxy-service. Alternatively,



OpenDaylight can be configured to listen on a different port, by modifying the `jetty.port` property value in `etc/jetty.conf`.

```
<Set name="port">
  <Property name="jetty.port" default="8080" />
</Set>
```

- Configure Neutron to use OpenDaylight's ML2 driver:

```
crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 mechanism_drivers_
↪opendaylight
crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 tenant_network_types vxlan

cat <<EOT>> /etc/neutron/plugins/ml2/ml2_conf.ini
[ml2_odl]
url = http://{CONTROL_HOST}:8080/controller/nb/v2/neutron
password = admin
username = admin
EOT
```

- Configure Neutron to use OpenDaylight's odl-router service plugin for L3 connectivity:

```
crudini --set /etc/neutron/plugins/neutron.conf DEFAULT service_plugins odl-router
```

- Configure Neutron DHCP agent to provide metadata services:

```
crudini --set /etc/neutron/plugins/dhcp_agent.ini DEFAULT force_metadata True
```

#### Note:

If the OpenStack version being used is Newton, this workaround should be applied, configuring the Neutron DHCP agent to use `vsctl` as the OVSDDB interface:

```
crudini --set /etc/neutron/plugins/dhcp_agent.ini OVS ovsdb_interface vsctl
```

- Reset Neutron's database

```
mysql -e "DROP DATABASE IF EXISTS neutron;"
mysql -e "CREATE DATABASE neutron CHARACTER SET utf8;"
/usr/local/bin/neutron-db-manage --config-file /etc/neutron/neutron.conf --config-
file /etc/neutron/plugins/ml2/ml2_conf.ini upgrade head
```

- Restart neutron-server:

```
systemctl start neutron-server
```

## Verifying it works

- Verify that OpenDaylight's ML2 interface is working:

```
curl -u admin:admin http://{CONTROL_HOST}:8080/controller/nb/v2/neutron/networks
{
  "networks" : [ ]
}
```

If this does not work or gives an error, check Neutron's log file in `/var/log/neutron/server.log`. Error messages here should give some clue as to what the problem is in the connection with OpenDaylight.

- Create a network, subnet, router, connect ports, and start an instance using the Neutron CLI:

```
neutron router-create router1
neutron net-create private
neutron subnet-create private --name=private_subnet 10.10.5.0/24
neutron router-interface-add router1 private_subnet
nova boot --flavor <flavor> --image <image id> --nic net-id=<network id> test1
nova boot --flavor <flavor> --image <image id> --nic net-id=<network id> test2
```

At this point, you have confirmed that OpenDaylight is creating network end-points for instances on your network and managing traffic to them.

VMs can be reached using Horizon console, or alternatively by issuing `nova get-vnc-console <vm> novnc`. Through the console, connectivity between VMs can be verified.

## Adding an external network for floating IP connectivity

- In order to connect to the VM using a floating IP, we need to configure external network connectivity, by creating an external network and subnet. This external network must be linked to a physical port on the machine, which will provide connectivity to an external gateway.

```

sudo ovs-vsctl set Open_vSwitch . other_config:provider_mappings=physnet1:eth1
neutron net-create public-net -- --router:external --is-default --
↪provider:network_type=flat --provider:physical_network=physnet1
neutron subnet-create --allocation-pool start=10.10.10.2,end=10.10.10.254 --
↪gateway 10.10.10.1 --name public-subnet public-net 10.10.0.0/16 -- --enable_
↪dhcp=False
neutron router-gateway-set router1 public-net

neutron floatingip-create public-net
nova floatingip-associate test1 <floating_ip>

```

### 4.1.2 Installing OpenStack and OpenDaylight using DevStack

The easiest way to load and OpenStack setup using OpenDaylight is by using devstack, which does all the steps mentioned in previous sections. `git clone https://git.openstack.org/openstack-dev/devstack`

- The following lines need to be added to your local.conf:

```

enable_plugin networking-odl http://git.openstack.org/openstack/networking-odl
↪<branch>
ODL_MODE=allinone
Q_ML2_PLUGIN_MECHANISM_DRIVERS=opendaylight,logger
ODL_GATE_SERVICE_PROVIDER=vpnservice
disable_service q-l3
ML2_L3_PLUGIN=odl-router
ODL_PROVIDER_MAPPINGS={PUBLIC_PHYSICAL_NETWORK}<external linux interface>

```

- More details on using devstack can be found in the following links:
  - Devstack All-In-One Single Machine Tutorial
  - Devstack networking-odl README

### 4.1.3 Troubleshooting

#### VM DHCP Issues

- Trigger DHCP requests - access VM console:
  - View log: `nova console-log <vm>`
  - Access using VNC console: `nova get-vnc-console <vm> novnc`
  - Trigger DHCP requests: `sudo ifdown eth0 ; sudo ifup eth0`

```

udhcpd (v1.20.1) started
Sending discover...
Sending select for 10.0.123.3...
Lease of 10.0.123.3 obtained, lease time 86400 # This only happens when DHCP
↪is properly obtained.

```

- Check if the DHCP requests are reaching the qdhcp agent using the following commands on the OpenStack controller:

```

sudo ip netns
sudo ip netns exec qdhcp-xxxxxx ifconfig # xxxx is the neutron network id

```



```

sudo ip netns exec qdhcp-xxxxx tcpdump -nei tapxxxxx # xxxxx is the neutron port_
↳ id

# Valid request and response:
15:08:41.684932 fa:16:3e:02:14:bb > ff:ff:ff:ff:ff:ff, ethertype IPv4 (0x0800),
↳ length 329: 0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from_
↳ fa:16:3e:02:14:bb, length 287
15:08:41.685152 fa:16:3e:79:07:98 > fa:16:3e:02:14:bb, ethertype IPv4 (0x0800),
↳ length 354: 10.0.123.2.67 > 10.0.123.3.68: BOOTP/DHCP, Reply, length 312

```

- If the requests aren't reaching qdhcp:

- Verify VXLAN tunnels exist between compute and control nodes by using `ovs-vsctl show`
- Run the following commands to debug the OVS processing of the DHCP request packet:
 

```

ovs-ofctl -OOpenFlow13 dump-ports-desc br-int # retrieve VMs ofport and MAC
ovs-appctl ofproto/trace br-int in_port=<ofport>,dl_src=<mac>,
dl_dst=ff:ff:ff:ff:ff:ff,udp,ip_src=0.0.0.0,ip_dst=255.255.255.255 |
grep "Rule\|action"

```

```

root@devstack:~# ovs-appctl ofproto/trace br-int in_port=1,dl_
↳ src=fe:16:3e:33:8b:d8,dl_dst=ff:ff:ff:ff:ff:ff,udp,ip_src=0.0.0.0,ip_
↳ dst=255.255.255.255 | grep "Rule\|action"
    Rule: table=0 cookie=0x8000000 priority=1,in_port=1
    OpenFlow actions=write_metadata:0x20000000001/0xffffffff000000001,goto_
↳ table:17
    Rule: table=17 cookie=0x8000001 priority=5,metadata=0x20000000000/
↳ 0xffffffff0000000000
    OpenFlow actions=write_metadata:0xc0000200000222e2/0xfffffffffffffffffe,
↳ goto_table:19
    Rule: table=19 cookie=0x1080000 priority=0
    OpenFlow actions=resubmit(,17)
    Rule: table=17 cookie=0x8040000 priority=6,
↳ metadata=0xc000020000000000/0xffffffff0000000000
    OpenFlow actions=write_metadata:0xe00002138a000000/
↳ 0xffffffffffffffffffe,goto_table:50
    Rule: table=50 cookie=0x8050000 priority=0
    OpenFlow actions=CONTROLLER:65535,goto_table:51
    Rule: table=51 cookie=0x8030000 priority=0
    OpenFlow actions=goto_table:52
    Rule: table=52 cookie=0x870138a priority=5,
↳ metadata=0x138a000001/0xffff000001
    OpenFlow actions=write_actions(group:210003)
    Datapath actions: drop

root@devstack:~# ovs-ofctl -OOpenFlow13 dump-groups br-int | grep 'group_
↳ id=210003'
    group_id=210003,type=all

```

- If the requests are reaching qdhcp, but the response isn't arriving to the VM:

- Locate the compute the VM is residing on (can use `nova show <vm>`).
  - \* If the VM is on the same node as the qdhcp namespace, `ofproto/trace` can be used to track the packet:
 

```

ovs-appctl ofproto/trace br-int
in_port=<dhcp_ofport>,dl_src=<dhcp_port_mac>,dl_dst=<vm_port_mac>,
udp,ip_src=<dhcp_port_ip>,ip_dst=<vm_port_ip> | grep

```

"Rule\|action"

```

root@devstack:~# ovs-appctl ofproto/trace br-int in_port=2,dl_
↪src=fa:16:3e:79:07:98,dl_dst=fa:16:3e:02:14:bb,udp,ip_src=10.0.123.2,ip_
↪dst=10.0.123.3 | grep "Rule\|action"
    Rule: table=0 cookie=0x8000000 priority=4,in_port=2
    OpenFlow actions=write_metadata:0x10000000000/0xffffffff0000000001,goto_
↪table:17
        Rule: table=17 cookie=0x80000001 priority=5,metadata=0x10000000000/
↪0xffffffff0000000000
        OpenFlow actions=write_metadata:0x60000100000222e0/
↪0xffffffffffffffffffe,goto_table:19
            Rule: table=19 cookie=0x1080000 priority=0
            OpenFlow actions=resubmit(,17)
                Rule: table=17 cookie=0x8040000 priority=6,
↪metadata=0x6000010000000000/0xffffffff0000000000
                OpenFlow actions=write_metadata:0x7000011389000000/
↪0xffffffffffffffffffe,goto_table:50
                    Rule: table=50 cookie=0x8051389 priority=20,
↪metadata=0x11389000000/0xffffffff000000,dl_src=fa:16:3e:79:07:98
                    OpenFlow actions=goto_table:51
                        Rule: table=51 cookie=0x8031389 priority=20,
↪metadata=0x1389000000/0xffff000000,dl_dst=fa:16:3e:02:14:bb
                        OpenFlow actions=load:0x300->NXM_NX_REG6[],
↪resubmit(,220)
                            Rule: table=220 cookie=0x8000007 priority=7,
↪reg6=0x300
                                OpenFlow actions=output:3

```

\* If the VM isn't on the same node as the qdhcp namespace:

- Check if the packet is arriving via VXLAN by running `tcpdump -nei <vxlan_port> port 4789`
- If it is arriving via VXLAN, the packet can be tracked on the compute node rules, using `ofproto/trace` in a similar manner to the previous section. Note that packets arriving from a tunnels have a unique `tunnel_id` (VNI) that should be used as well in the trace, due to the special processing of packets arriving from a VXLAN tunnel.

## Floating IP Issues

- If you have assigned an external network and associated a floating IP to a VM but there is still no connectivity:
  - Verify the external gateway IP is reachable through the provided provider network port.
  - Verify OpenDaylight has successfully resolved the MAC address of the external gateway IP. This can be verified by searching for the line "Installing ext-net group" in the `karaf.log`.
  - Locate the compute the VM is residing on (can use `nova show <vm>`).
  - Run a ping to the VM floating IP.
  - If the ping fails, execute a flow dump of `br-int`, and search for the flows that are relevant to the VM's floating IP address: `ovs-ofctl -OOpenFlow13 dump-flows br-int | grep "<floating_ip>"`
- \* Are there packets on the incoming flow (matching `dst_ip=<floating_ip>`)?
 

If not this probably means the provider network has not been set up properly, verify `provider_mappings` configuration and the configured external network `physical_network` value

match. Also verify that the Flat/VLAN network configured is actually reachable via the configured port.

- \* Are there packets on the outgoing flow (matching `src_ip=<floating_ip>`)?  
If not, this probably means that OpenDaylight is failing to resolve the MAC of the provided external gateway, required for forwarding packets to the external network.
- \* Are there packets being sent on the external network port?  
This can be checked using `tcpdump <port>` or by viewing the appropriate OpenFlow rules. The mapping between the OpenFlow port number and the linux interface can be acquired using `ovs-ofctl dump-ports-desc br-int`

```
ovs-ofctl -OOpenFlow13 dump-flows br-int | grep "<floating_ip>"
cookie=0x80000003, duration=436.710s, table=21, n_packets=190, n_
↳bytes=22602, priority=42, ip, metadata=0x222e2/0xfffffffffe, nw_dst=10.64.98.
↳17 actions=goto_table:25
cookie=0x80000004, duration=436.739s, table=25, n_packets=190, n_
↳bytes=22602, priority=10, ip, nw_dst=10.64.98.17 actions=set_field:10.0.
↳123.3->ip_dst, write_metadata:0x222e0/0xfffffffffe, goto_table:27
cookie=0x80000004, duration=436.730s, table=26, n_packets=120, n_
↳bytes=15960, priority=10, ip, metadata=0x222e0/0xfffffffffe, nw_src=10.0.123.
↳3 actions=set_field:10.64.98.17->ip_src, write_metadata:0x222e2/
↳0xfffffffffe, goto_table:28
cookie=0x80000004, duration=436.728s, table=28, n_packets=120, n_
↳bytes=15960, priority=10, ip, metadata=0x222e2/0xfffffffffe, nw_src=10.64.98.
↳17 actions=set_field:fa:16:3e:ec:a8:84->eth_src, group:200000
```

#### 4.1.4 Useful Links

- [NetVirt Tables Pipeline](#)
- [NetVirt Wiki Page](#)
- [NetVirt Basic Tutorial \(OpenDaylight Summit 2016\)](#)
- [NetVirt Advanced Tutorial \(OpenDaylight Summit 2016\)](#)
- [Other OpenDaylight Documentation](#)



## 5.1 L3VPN Service: User Guide

### 5.1.1 Overview

L3VPN Service in OpenDaylight provides a framework to create L3VPN based on BGP-MP. It also helps to create Network Virtualization for DC Cloud environment.

### 5.1.2 Modules & Interfaces

L3VPN service can be realized using the following modules -

#### VPN Service Modules

1. **VPN Manager** : Creates and manages VPNs and VPN Interfaces
2. **BGP Manager** : Configures BGP routing stack and provides interface to routing services
3. **FIB Manager** : Provides interface to FIB, creates and manages forwarding rules in Dataplane
4. **Nexthop Manager** : Creates and manages nexthop egress pointer, creates egress rules in Dataplane
5. **Interface Manager** : Creates and manages different type of network interfaces, e.g., VLAN, l3tunnel etc.,
6. **Id Manager** : Provides cluster-wide unique ID for a given key. Used by different modules to get unique IDs for different entities.
7. **MD-SAL Util** : Provides interface to MD-SAL. Used by service modules to access MD-SAL Datastore and services.

All the above modules can function independently and can be utilized by other services as well.

## Configuration Interfaces

The following modules expose configuration interfaces through which user can configure L3VPN Service.

1. BGP Manager
2. VPN Manager
3. Interface Manager
4. FIB Manager

## Configuration Interface Details

1. Data Node Path : */config/bgp:bgp-router/*
  - (a) Fields :
    - i. local-as-identifier
    - ii. local-as-number
  - (b) REST Methods : GET, PUT, DELETE, POST
2. Data Node Path : */config/bgp:bgp-neighbors/*
  - (a) Fields :
    - i. List of bgp-neighbor
  - (b) REST Methods : GET, PUT, DELETE, POST
3. Data Node Path : */config/bgp:bgp-neighbors/bgp-neighbor/{as-number}/*
  - (a) Fields :
    - i. as-number
    - ii. ip-address
  - (b) REST Methods : GET, PUT, DELETE, POST
1. Data Node Path : */config/l3vpn:vpn-instances/*
  - (a) Fields :
    - i. List of vpn-instance
  - (b) REST Methods : GET, PUT, DELETE, POST
2. Data Node Path : */config/l3vpn:vpn-interfaces/vpn-instance*
  - (a) Fields :
    - i. name
    - ii. route-distinguisher
    - iii. import-route-policy
    - iv. export-route-policy
  - (b) REST Methods : GET, PUT, DELETE, POST
3. Data Node Path : */config/l3vpn:vpn-interfaces/*
  - (a) Fields :

- i. List of vpn-interface
- (b) REST Methods : GET, PUT, DELETE, POST
- 4. Data Node Path : */config/l3vpn:vpn-interfaces/vpn-interface*
  - (a) Fields :
    - i. name
    - ii. vpn-instance-name
  - (b) REST Methods : GET, PUT, DELETE, POST
- 5. Data Node Path : */config/l3vpn:vpn-interfaces/vpn-interface/'{name}'/adjacency*
  - (a) Fields :
    - i. ip-address
    - ii. mac-address
  - (b) REST Methods : GET, PUT, DELETE, POST
- 1. Data Node Path : */config/lf:interfaces/interface*
  - (a) Fields :
    - i. name
    - ii. type
    - iii. enabled
    - iv. of-port-id
    - v. tenant-id
    - vi. base-interface
  - (b) type specific fields
    - i. when type = *l2vlan*
      - A. vlan-id
    - ii. when type = *stacked\_vlan*
      - A. stacked-vlan-id
    - iii. when type = *l3tunnel*
      - A. tunnel-type
      - B. local-ip
      - C. remote-ip
      - D. gateway-ip
    - iv. when type = *mpls*
      - A. list labelStack
      - B. num-labels
  - (c) REST Methods : GET, PUT, DELETE, POST
- 1. Data Node Path : */config/odl-fib:fibEntries/vrfTables*
  - (a) Fields :

- i. List of vrfTables
- (b) REST Methods : GET, PUT, DELETE, POST
- 2. Data Node Path : `/config/odl-fib:fibEntries/vrfTables/{routeDistinguisher}/`
  - (a) Fields :
    - i. route-distinguisher
    - ii. list vrfEntries
      - A. destPrefix
      - B. label
      - C. nexthopAddress
  - (b) REST Methods : GET, PUT, DELETE, POST
- 3. Data Node Path : `/config/odl-fib:fibEntries/ipv4Table`
  - (a) Fields :
    - i. list ipv4Entry
      - A. destPrefix
      - B. nexthopAddress
  - (b) REST Methods : GET, PUT, DELETE, POST

### 5.1.3 Provisioning Sequence & Sample Configurations

#### Installation

1. Edit `etc/custom.properties` and set the following property: `vpnservice.bgpspeaker.host.name = <bgpserver-ip>`  
`<bgpserver-ip>` here refers to the IP address of the host where BGP is running.
2. Run ODL and install VPN Service `feature:install odl-vpnservice-core`

Use REST interface to configure L3VPN service

#### Pre-requisites:

1. BGP stack with VRF support needs to installed and configured
  - (a) *Configure BGP as specified in Step 1 below.*
2. Create pairs of GRE/VxLAN Tunnels (using `ovsdb/ovs-vsctl`) between each switch and between each switch to the Gateway node
  - (a) *Create `*l3tunnel` interfaces corresponding to each tunnel in interfaces DS as specified in Step 2 below.\**

#### Step 1 : Configure BGP

##### 1. Configure BGP Router

REST API : `PUT /config/bgp:bgp-router/`

Sample JSON Data



```
{
  "bgp-router": {
    "local-as-identifier": "10.10.10.10",
    "local-as-number": 108
  }
}
```

## 2. Configure BGP Neighbors

**REST API :** *PUT /config/bgp:bgp-neighbors/*

**Sample JSON Data**

```
{
  "bgp-neighbor" : [
    {
      "as-number": 105,
      "ip-address": "169.144.42.168"
    }
  ]
}
```

## Step 2 : Create Tunnel Interfaces

Create l3tunnel interfaces corresponding to all GRE/VxLAN tunnels created with ovsdb (*refer Prerequisites*). Use following REST Interface -

**REST API :** *PUT /config/lf:interfaces/lf:interfacce*

**Sample JSON Data**

```
{
  "interface": [
    {
      "name" : "GRE_192.168.57.101_192.168.57.102",
      "type" : "odl-interface:l3tunnel",
      "odl-interface:tunnel-type": "odl-interface:tunnel-type-gre",
      "odl-interface:local-ip" : "192.168.57.101",
      "odl-interface:remote-ip" : "192.168.57.102",
      "odl-interface:portId" : "openflow:1:3",
      "enabled" : "true"
    }
  ]
}
```

**Following is expected as a result of these configurations**

1. Unique If-index is generated
2. *Interface-state* operational DS is updated
3. Corresponding Nexthop Group Entry is created

### Step 3 : OS Create Neutron Ports and attach VMs

At this step user creates VMs.

### Step 4 : Create VM Interfaces

Create l2vlan interfaces corresponding to VM created in step 3

**REST API** : *PUT /config/if:interfaces/if:interface*

**Sample JSON Data**

```
{
  "interface": [
    {
      "name" : "dpn1-dp1.2",
      "type" : "l2vlan",
      "odl-interface:of-port-id" : "openflow:1:2",
      "odl-interface:vlan-id" : "1",
      "enabled" : "true"
    }
  ]
}
```

### Step 5: Create VPN Instance

**REST API** : *PUT /config/l3vpn:vpn-instances/l3vpn:vpn-instance/*

**Sample JSON Data**

```
{
  "vpn-instance": [
    {
      "description": "Test VPN Instance 1",
      "vpn-instance-name": "testVpn1",
      "ipv4-family": {
        "route-distinguisher": "4000:1",
        "export-route-policy": "4000:1,5000:1",
        "import-route-policy": "4000:1,5000:1",
      }
    }
  ]
}
```

**Following is expected as a result of these configurations**

1. VPN ID is allocated and updated in data-store
2. Corresponding VRF is created in BGP
3. If there are vpn-interface configurations for this VPN, corresponding action is taken as defined in step 5

### Step 5 : Create VPN-Interface and Local Adjacency

*this can be done in two steps as well*

## 1. Create vpn-interface

**REST API :** *PUT /config/l3vpn:vpn-interfaces/l3vpn:vpn-interface/*

**Sample JSON Data**

```
{
  "vpn-interface": [
    {
      "vpn-instance-name": "testVpn1",
      "name": "dpn1-dp1.2",
    }
  ]
}
```

**Note:** name here is the name of VM interface created in step 3, 4

## 2. Add Adjacencies on vpn-interafce

**REST API :** *PUT /config/l3vpn:vpn-interfaces/l3vpn:vpn-interface/dpn1-dp1.3/adjacency*

**Sample JSON Data**

```
{
  "adjacency" : [
    {
      "ip-address" : "169.144.42.168",
      "mac-address" : "11:22:33:44:55:66"
    }
  ]
}
```

its a **list**, user can define more than one adjacency on a vpn\\_interface

Above steps can be carried out in a single step as following

```
{
  "vpn-interface": [
    {
      "vpn-instance-name": "testVpn1",
      "name": "dpn1-dp1.3",
      "odl-l3vpn:adjacency": [
        {
          "odl-l3vpn:mac_address": "11:22:33:44:55:66",
          "odl-l3vpn:ip_address": "11.11.11.2",
        }
      ]
    }
  ]
}
```

## Following is expected as a result of these configurations

1. Prefix label is generated and stored in DS
2. Ingress table is programmed with flow corresponding to interface
3. Local Egress Group is created
4. Prefix is added to BGP for advertisement
5. BGP pushes route update to FIB YANG Interface
6. FIB Entry flow is added to FIB Table in OF pipeline

## 5.2 Support

### Table of Contents

- *Support*
  - *Verified Combinations*
  - *Open vSwitch Kernel and DPDK Modes*

### 5.2.1 Verified Combinations

This section describes which versions of OpenStack and Open vSwitch are expected to work with with OpenDaylight. Using combinations outside this list may work but have not been verified.

**Note:** Verified is defined as combinations that are actively tested and maintained. OpenDaylight, OpenStack and Open vSwitch are very active and quickly adding new features that makes it difficult to verify all the different release combinations. Different combinations are likely to work but support will be limited.

The following table details the expected supported combinations.

Table 5.1: Supported Version Matrix

OpenDaylight	OpenStack	Open vSwitch	Sync	Notes
Boron	Newton	2.6	S	
Carbon	Ocata	2.7		Combination drops when Pike releases
Carbon	Pike	2.7	S	
Nitrogen	Ocata	2.7		Combination drops when Pike releases
Nitrogen	Pike	2.7		Combination drops when Queens releases
Nitrogen	Queens	2.8/2.9	S	
Oxygen	Pike	2.7		Combination drops when Queens releases
Oxygen	Queens	2.8/2.9		Combination drops when OpenStack R releases
Oxygen	R	2.9	S	

- (S): in the Sync column indicates the final supported combination for that OpenDaylight release.
- Differing release schedules will lead to short-lived combinations that will drop as the releases line up. An example is with Carbon that releases before Pike so for a period of time Carbon is supported with Ocata.

- The current OpenDaylight version and the previous will be supported. Boron support will drop when Nitrogen releases; Carbon support will drop when Oxygen releases.

## 5.2.2 Open vSwitch Kernel and DPDK Modes

The table below lists the Open vSwitch requirements for the Carbon release.

Table 5.2: Kernel and DPDK Modes

Feature	OVS 2.6 kernel mode	OVS 2.6 dpdk mode
Conntrack - security groups	yes	yes
Conntrack - NAT	yes	no (target 2.8*)
Security groups stateful	yes (conntrack)	yes(conntrack)
Security groups learn	yes (but not needed)	yes (but not needed)
IPV4 NAT (without pkt punt to controller)	yes (conntrack)	no (target 2.8*)
IPV4 NAT (with pkt punt to controller)	not needed	yes (until 2.8*)

(\*) support is tentatively scheduled for Open vSwitch 2.8

## 5.3 Bridge Configuration

### Table of Contents

- *Bridge Configuration*
  - *The “br-int” Bridge*
  - *Provider Networks*

The following describes OVS bridge configurations supported by OpenDaylight.

### 5.3.1 The “br-int” Bridge

If the br-int bridge is not configured prior to the ovsdb manager connection with ODL, ODL will create it. If br-int exists prior to the ovsdb manager connection, ODL will retain any existing configurations on br-int. Note that if you choose to create br-int prior to connecting to ODL, `disable-in-band` MUST be set to true and any flows configured may interfere with the flows ODL will create. ODL will add the following configuration items to br-int:

1. ODL will set itself as br-int’s controller
2. Any provider network configuration (see section “Provider Networks” below)

It is important to note that once the ovsdb manager connection is established with ODL, ODL “owns” br-int and other applications should not modify its settings.

### 5.3.2 Provider Networks

Provider networks should be configured prior to OVSDb connecting to ODL. These are configured in the Open\_vSwitch table’s other\_Config column and have the format `<physnet>:<connector>` where `<physnet>` is the name of the provider network and `<connector>` is one of the following three options:

1. The name of a local interface (ODL will add this port to br-int)

2. The name of a bridge on OpenVSwitch (ODL will create patch ports between br-int and this bridge)
3. The name of a port already present on br-int (ODL will use that port)

For example, assume your provider network is called extnet and it is attached to the eth0 interface on your host you can set this in OVSDB using the following command:

```
sudo ovs-vsctl set Open_vSwitch . Other_Config:provider_mappings=extnet:eth0
```

If instead of eth0 the provider network is accesable via on OVS bridge called br-int, eth0 in the above command would be substituted with br-int.

---

## Bibliography

---

[QBGp] Quagga Routing Suite

[RFC2385] IETF RFC 2385: Protection of BGP Sessions via the TCP MD5 Signature Option

[TBaseProcessor] thrift java library's TBaseProcessor.process

[ZRPC] Zebra Remote Procedure Call