ODL NetVirt

Release master

Vishal Thapar, Andre Fredette, Sam Hague

Mar 20, 2020

CONTENTS

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

Contents:

CHAPTER

ONE

NETVIRT CONTRIBUTOR GUIDE

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

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: 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.

1.1.2 NetVirt Design Specifications

Contents:

Table of Contents

Enable discovery of Virtual IPs (or Movable IPs) holding Virtual MACs

https://git.opendaylight.org/gerrit/#/q/topic:vpn_vmac_movement

Enable discovery of Movable (or Virtual IPs) which have their own sticky Virtual MACs, in an L3VPN domain.

The discovery of these virtual IPs will be triggered by either IP traffic originated from a source outsidethe DC (same L3VPN domain) towards such Virtual IPs (or) from IP traffic originated from a source within theDC (again within same L3VPN domain) towards such Virtual IPs.

Also ensure to enforce discovery of new locations of such Virtual IPs (along with their Virtual MACs) whenthey move inside the DC and program the routes appropriately thereby providing continual L3 connectivity towards such Virtual IPs, both from within and from outside DC.

Virtual IP and Moveable IP are used interchangeably in this spec and both of them simply represent a moveable virtual IP.

Problem description

Discovery of Virtual IPs , which hold the MAC address of their hosted Virtual Ports (instead of holding their own Virtual MAC) is already supported by L3VPN service today via the SubnetRoute feature.

In NFVI production environments that run VNFs, for providing high-availability for those VNFs , VRRP is run between two such VNF instances. With VRRP configuration, a Virtual IP is chosen and that Virtual IP is used to access the VNF by the tenants. This Virtual IP will move around between the two VNF instances based on whichever is alive (or whichever has higher priority). If only a Virtual IP is used by VRRP with the MAC being the same as the hosting VNF interface MAC, then that use-case is already supported by SubnetRoute feature.

However, it is possible to configure Virtual MAC to such Virtual IPs wherein those Virtual MACs are different from the VNF hosting interface MAC Addresses. When this is being done, L3VPN service is unable to provide L3 connectivity to such IPs because L3VPN service points up the incorrect hosting interface MAC Address (in lieu of using the Virtual MAC itself) in the packets sent to Virtual IP which makes the Virtual IP ignore such packets.

This feature is thence an attempt to enhance the existing VIP discovery (and L3 plumbing of such VIPs) such that L3VPN service can support Virtual IPs that carry their own Virtual MACs. In addition to the same this feature will track movement of such Virtual IPs in the cloud will be tracked and appropriate flows would be pushed in, to provide continual L3 connectivity (intra+inter-dc) to such Virtual IPs within an L3VPN domain.

VNF movement inside a Data Center is supported today. There are VNF's which do not change the MAC address even after movement between OF-port's.

The netvirt/vpnManaager expects a different MAC address when same VNF (which was earlier detected by ARP-response/GARP). This satisfies requirements of few VNF's, but VNF's like "load-balancer, high-availability", require the vMAC/IP remain same even after multiple incarnation at different OF-ports. ODL MUST provide seamless connectivity to all these VNF's.

Use Cases

1. Discovery of Virtual IPs that have sticky Virtual MACs ending up with providing L3 connectivity for such Virtual IPs.
2. Discovery of new location of Virtual IPs (with sticky Virtual MACs), when such Virtual IPs move within an L3VPN domain in the cloud, ending up with continual L3 connectivity maintained for such Virtual IPs.
3. Virtual IPs with sticky Virtual MACs to work on L3VPNs that have export/import relationship with other L3VPNs configured in the cloud, ending up with L3 connectivity for such Virtual IPs from related L3VPNs.
4. When a Virtual IP belongs to only one VM port and VM migrates to a new location carrying along the Virtual IP, ending up with L3 connecitivty for such Virtual IPs retained intact.
5. Tear down of such Virtual IPs (ie., VIP route) when the only hosting VM port is deleted, ending up with removal of routes representing the Virtual IP.
6. If a VM port (and its Virtual IP) remained intact during cluster reboot, then both during cluster reboot and after cluster reboot there should be no dataplane traffic loss to/from the Virtual IP within the L3VPN domain in the cloud.
7. Discovery of Virtual IPs (without Virtual MACs) should happen, ending up with L3 connectivity established for such Virtual IPs.
8. All the above steps will be certified for discovered IPv6 addresses with Virtual MAC too as the following spec addresses IPv6 address discovery: https://git.opendaylight.org/gerrit/70912

Proposed change

As the current code stands, we are using the same GroupId in the dataplane for both the primary VM and MIP discovered on that VM.

With this feature, we will be carving out a new Group for specifically managing MIPs and this new Group will be different from the datapalane Group used to steer traffic to primary hosting interface for the MIP.

- GARP/ARP-response packet handler: shall detect movement of movable IP between port based on connected OF-port & DPN combination.
- Packet destined to movable-IP, shall contain the destination MAC as vMAC published by VNF (which was received in GARP/ARP-response).

All along current code has been using the MAC-address in the GARP request packet-Ins to decide if a VIP movement has happened. This logic is ok and works for VIPs that donot hold any Virtual MACs of their own.

But going forward as this spec support VIPs holding Virtual MACs, we will be using the interface-name (or neutron-port-id) from which the GARP packets are received by the NetVirt ARP Handler service to track VIP movements in general thereby replacing the logic that used MAC-Addresses in incoming GARP packet used earlier.

Pipeline changes

none

Yang changes

none

Configuration impact

none

Clustering considerations

connectivity to all movable ip (vip) with vmac, should remain intact during cluster reboot. connectivity to all movable ip (vip) with vmac, should remain intact after cluster reboot. connectivity to all movable ip (vip) with vmac, should remain intact during cluster upgrade. connectivity to all movable ip (vip) with vmac, should remain intact after cluster upgrade. connectivity to all movable ip (vip) without vmac, should remain intact during cluster reboot. connectivity to all movable ip (vip) without vmac, should remain intact after cluster reboot. connectivity to all movable ip (vip) without vmac, should remain intact during cluster upgrade. connectivity to all movable ip (vip) without vmac, should remain intact after cluster upgrade. Connectivity to all movable ip (VNF's) remain intact even after cluster reboot. Connectivity to all movable ip (VNF's) remain intact even after single node failure (leader/non-leader).

Other Infra considerations

none

Security considerations

none

Scale and Performance Impact

none

Targeted Release

Flourine

Alternatives

N.A.

Usage

none

Features to Install

odl-netvirt-openstack

REST API

none

CLI

Implementation

Assignee(s)

Primary assignee: Siva Kumar Perumalla, <sivakumar.perumalla@ericsson.com>

Other contributors: Akash Sahu, <a.k.sahu@ericsson.com>

Work Items

- GARP/ARP-response packet handler: shall detect movement of movable IP between port based on connected OF-port & DPN combination.
- Packet destined to movable-IP, shall contain the destination MAC as vMAC published by VNF (which was received in GARP/ARP-response).
- Enhance ArpNotificationHandler to detect MIP movement based on Interface from which the GARP Packet / ARP Response is received.
- Enhance VRFEngine to create and manage a separate group for MIPs (regardless of whether they hold a VMAC or not).
- Make sure this separate group works for Import/Export related VPNs and push any changes are needed for the same.
- Make sure Aliveness Monitor uses the Virtual MAC owned by VirtualIP instead of the hosting interface IP, and continues to retain its functionality of VIP expiry logic for these new types of VIPs.

Dependencies

none.

Unit Tests

- Verification of MAC movement (using generated MAC, not port MAC).
- Hypervisor disconnection (hosting VNF) from ODL, Data Path shall be intact, till aliveness monitor detects.
- Hypervisor reboot (hosting VNF) from ODL, Data Path shall be intact (hypervisor comes-up within aliveness monitor time interval).
- VNF reboot: data path shall be intact after reboot (assuming VNF generates GARP).

Integration Tests

none

CSIT

- enhance the current ARP learning suite with the new use-cases quooted in the use-case section above thereby providing CSIT coverage for this feature.

Documentation Impact

none

References

Table of Contents

Evpn and l2gw integration

https://git.opendaylight.org/gerrit/#/q/topic:EVPN_L2GW

Enable support of l2gw integration with evpn.

Problem description

OpenDaylight NetVirt service today supports EVPN over VXLAN tunnels. L2DCI communication was made possible with this.

It also supports l2gw use cases.

This spec attempts to integrate these two services.

In scope

Only L2 support will be added now.

Out of scope

L3 communication is not supported.

Use Cases

All the use cases supported by RT2 feature will be supported.

In addition the following use cases will be supported.

1) Intra subnet communication between sriov vms across datacenters.

To achieve this remote mcast table should have a tep pointing towards the datacenter gateway, Remote ucast table should have the other datacenter sriov mac reachable via the datacenter gateway nexthop. The l2vni used in the following configuration is the vxlan segmentation id of the network to which l2gw and evpn instances are attached to.

Assume the following datapath and control path connectivity respectively.

VM1(MAC1) <-> TOR1 <-> DC-GW1(DATACENTER-GATEWAY) <-> DC-GW2 <-> TOR2 <-> VM2(MAC2)

TOR1 <-> ODL1 <-> DC-GW1 <-> DC-GW2 <-> ODL2 <-> TOR2

The following table programing is performed to achieve the datapath connectivity.

Remote Mcast Table of TOR1 unknown-mac Logicalswitch1 [DC-GW1-TEP, OTHER-COMPUTES-TEPS-ON-SAME-NETWORK]

Remote Ucast Table of TOR1 MAC2 Logicalswitch1 DC-GW1-TEP

Remote Mcast Table of TOR2 unknown-mac Logicalswitch1 [DC-GW2-TEP, OTHER-COMPUTES-TEPS-ON-SAME-NETWORK]

Remote Ucast Table of TOR2 MAC1 Logicalswitch1 [DC-GW2-TEP]

ODL1 will advertize RT2 route to DC-GW1 for MAC1 with TOR1 TEP as the the next hop

ODL2 will advertize RT2 route to DC-GW2 for MAC2 with TOR2 TEP as the the next hop

The following is the sequence of steps an ARP packet will take.

- Initial ARP packet from VM1 will be looked up in Remote mcast table of TOR1 and will be flooded to DC-GW1

- DC-GW1 will flood it to DC-GW2
- DC-GW2 will flood it to TOR2
- TOR2 will flood it to VM2
- VM2 will respond with unicast ARP response.
- Unicast ARP response will be looked up in Remote ucast table of TOR2 and will be forwarded to DC-GW2 (vxlan encapsulated with l2vni)
- DC-GW2 will look up in mac-vrf table forward it to DC-GW1
- DC-GW1 will look up in mac-vrf table and forward it to TOR1
- TOR1 will look up the mac in Local Ucast Table and send it to VM1

The following is the sequence of steps a Unicast packet will take.

- Unicast IP packets will be simply looked up in Remote Ucast table of the TOR1 and forwarded to DC-GW1 (vxlan encapsulated with l2vni)
- DC-GW1 will lookup in mac-vrf table and forward it to DC-GW2
- DC-GW2 will look up in mac-vrf table and forward it to TOR2
- TOR2 will look up the mac in Local Ucast Table and forward it to VM2

2) Intra subnet communication between sriov vm and compute vm across datacenters.

The procedure is very much similar to the above use case except that the packet will be forwarded based on the openflow pipeline on the destination compute vm as defined in the RT2 spec https://git.opendaylight.org/gerrit/#/c/51693/.

Assume the following datapath and control path connectivity respectively.

VM1(MAC1) <-> TOR1 <-> DC-GW1(DATACENTER-GATEWAY) <-> DC-GW2 <-> COMPUTE2 <-> VM2(MAC2)

TOR1 <-> ODL1 <-> DC-GW1 <-> DC-GW2 <-> ODL2 <-> COMPUTE2

The following table programing is performed to achieve the datapath connectivity.

Remote Mcast Table of TOR1 unknown-mac Logicalswitch1 [DC-GW1-TEP, OTHER-COMPUTES-TEPS-ON-SAME-NETWORK]

Remote Ucast Table of TOR1 MAC2 Logicalswitch1 DC-GW1-TEP

Remote Broadcast Group buckets of Logicalswitch1(Network1) on Compute2 [DC-GW2-TEP, OTHER-COMPUTES-TEPS-ON-SAME-NETWORK]

DMac Table (51) of Compute2 MAC1 Logicalswitch1(Network1) [DC-GW2-TEP]

ODL1 will advertize RT2 route to DC-GW1 for MAC1 with TOR1 TEP as the the next hop

ODL2 will advertize RT2 route to DC-GW2 for MAC2 with COMPUTE2 TEP as the the next hop

The following is the sequence of steps an ARP packet will take.

- Initial ARP packet from VM1 will be looked up in Remote mcast table of TOR1 and will be flooded to DC-GW1
- DC-GW1 will flood it to DC-GW2
- DC-GW2 will flood it to COMPUTE2

- COMPUTE2 will flood it to VM2
- VM2 will respond with unicast ARP response.
- Unicast ARP response will be looked up in Dmac Table (51) of COMPUTE2 and will be forwarded to DC-GW2 (vxlan encapsulated with l2vni)
- DC-GW2 will lookup in mac-vrf table and forward it to DC-GW1
- DC-GW1 will lookup in mac-vrf table and forward it to TOR1
- TOR1 will look up the mac in Local Ucast Table and send it to VM1

The following is the sequence of steps a Unicast packet will take.

- Unicast IP packets will be simply looked up in Remote Ucast table of the TOR1 and forwarded to DC-GW1 (vxlan encapsulated with l2vni)
- DC-GW1 will lookup in mac-vrf table and forward it to DC-GW2
- DC-GW2 will look up in mac-vrf table and forward it to COMPUTE2
- COMPUTE2 will look up in Dmac Table and forward it to VM2

3) Intra subnet communication between baremetals behind l2gw across datacenters.

This is similar to case1.

4) Intra subnet communication between baremetal behind l2gw and compute vm across datacenters.

This is similar to case2.

Proposed change

Pipeline changes

No Pipeline changes are required.

Yang changes

There are no yang changes required.

Solution considerations

Proposed change in OpenDaylight-specific features

The following components within OpenDaylight Controller needs to be enhanced:

- ELAN Manager

The following actions are performed If the network is attached to evpn

Upon L2gw connection creation. Program remote mcast table of l2gw device with all the external teps of the elan instance.

Program remote ucast table of l2gw device with all the external macs from mac vrf table of the elan instance.

Advertize vtep of the l2gw device in RT3 route

Upon L2gw connection deletion. Delete remote mcast table entry for this elan instance of l2gw device.

Clear all the macs in mac vrf table of the elan instance from remote ucast table of l2gw device.

Withdraw RT3 route containing vtep of the l2gw device

Upon Local ucast add advertize the RT2 route for the added ucast mac

Upon Local ucast delete withdraw the RT2 route for the added ucast mac

Upon Receiving RT2 route Program remote ucast mac table of l2gw device with the mac received from RT2 route

Upon Receiving RT2 route withdraw Delete mac from remote ucast mac table of l2gw device

Upon Receiving RT3 route Program remote mcast mac table of l2gw device to include the tep

Upon Receiving RT3 route withdraw Program remote mcast mac table of l2gw device to exclude the tep

Upon attaching network to evpn Advertize local ucast macs of l2gw device as RT2 routes

Advertize vtep of the l2gw device in RT3 route

Upon detaching network from evpn Withdraw RT2 routes containing local ucast macs of l2gw device

Withdraw vtep of the l2gw device in RT3 route

Reboot Scenarios

This feature support all the following Reboot Scenarios for EVPN:

- Entire Cluster Reboot
- Leader PL reboot (PL : payload node: one of the nodes in the cluster)
- Candidate PL reboot
- OVS Datapath reboots
- Multiple PL reboots
- Multiple Cluster reboots
- Multiple reboots of the same OVS Datapath.
- Openstack Controller reboots

Configuration impact

N.A.

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

Fluorine.

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.

Implementation

Assignee(s)

Primary assignee: K.V Suneelu Verma <k.v.suneelu.verma@ericsson.com>

Vyshakh Krishnan C H <vyshakh.krishnan.c.h@ericsson.com>

Work Items

https://jira.opendaylight.org/browse/NETVIRT-1247

Dependencies

Requires a DC-GW that is supporting EVPN RT2 & RT3 on BGP Control plane. EVPN RT3 Support needs to be implemented in ODL before this.

Testing

Capture details of testing that will need to be added.

Unit Tests

Appropriate UTs will be added for the new code.

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.

1) create l2gw connection, verify that received routes from datacenter gateway are programmed in the l2gw device
2) create l2gw connection, verify that the l2gw device tep is advertized in RT3 route
3) delete l2gw connection, verify that received dcgw routes are deleted from the l2gw device
4) delete l2gw connection, verify that the RT3 route withdraw message is published
5) attach network to evpn, verify that l2gw local ucast macs are advertized as RT2 routes
6) detach network from evpn, verify that l2gw local ucast macs are withdrwan
7) receive RT2 route, verify that the received mac is programmed in l2gw device
8) upon receiving withdraw of RT2 route, verify that the received mac is deleted from l2gw device
9) receive RT3 route, verify that l2gw device mcast includes the dcgw nexthop ip
10) upon receiving withdraw of RT3 route, verify that l2gw device mcast excludes the dcgw nexthop ip

Documentation Impact

This will require changes to User Guide and Developer Guide.

References

[1] BGP MPLS-Based Ethernet VPN

Table of Contents

Support for RT3 route type in EVPN

https://git.opendaylight.org/gerrit/#/q/topic:EVPN_RT3

Enable support for RT3 route type in evpn deployment.

Problem description

OpenDaylight NetVirt service today supports EVPN over VXLAN tunnels. L2DCI communication was made possible with this.

This spec attempts to enhance EVPN service in NetVirt by adding support for RT3 route type.

Type 3 routes are required for Broadcast, Unknown Unicast and Multicast (BUM) traffic delivery across EVPN networks. Type 3 advertisements provide information that should be used to send BUM traffic. Without Type 3 advertisements, ingress router would not know how to deliver BUM traffic to other PE devices that comprise given EVPN instance.

Today BUM traffic is sent to all the datacenter gateways from which RT2 is received. Similarly datacenter gateway is excluded from BUM traffic when the last RT2 route is withdrawn from it.

Instead of relying on the RT2 routes to forward BUM traffic, RT3 route type is used to update the BUM traffic destination endpoints.

The format of Type 3 advertisement is as follows

Route Distinguisher (8 octets)
Ethernet Tag ID (4 octets)
IP Address Length (1 octet)
Originating Router's IP Address (4 or 16 octets)

Type 3 route also carries a Provider Multicast Service Interface (PMSI) Tunnel attribute.

Flags (1 octet)
Tunnel Type (1 octets)
MPLS Label (3 octets)
Tunnel Identifier (variable)

The following are the Tunnel Types:

- 0 - No tunnel information present
- 1 - RSVP-TE P2MP LSP
- 2 - mLDP P2MP LSP
- 3 - PIM-SSM Tree
- 4 - PIM-SM Tree

- 5 - BIDIR-PIM Tree
- 6 - Ingress Replication
- 7 - mLDP MP2MP LSP

In scope

Only Ingress Replication tunnel type will be supported for now. MPLS Label and Tunnel Identifier present in the received RT3 route will be ignored.

While advertizing MPLS Label and Tunnel Identifier will be set to a value of zero. While sending BUM traffic l2vni of the vpn instance is used in the vxlan packet.

Supports only ipv4 for Originating Router's IP Address field. Ipv6 support will be added later.

Out of scope

Tunnel types other than Ingress Replication will be ignored.

Use Cases

All the use cases supported by RT2 feauture will be supported. RT2 spec: http://docs.opendaylight.org/en/stable-nitrogen/submodules/netvirt/docs/specs/l2vpn-over-vxlan-with-evpn-rt2.html

Proposed change

Pipeline changes

No Pipeline changes are required.

Yang changes

There are no yang changes required.

The list `external-teps` is updated in elan container every time an RT3 route is received/withdrawn.

Listing 2: elan.yang

```
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;
            }
        }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

Solution considerations

Proposed change in BGP Quagga Stack

The following thrift apis would be added to communicate to quagga.

Listing 3: elan.yang

```
pushEvpnRT(EvpnConfigData evpnConfig)
onupdatePushEvpnRT(EvpnConfigData evpnConfig)

EvpnConfigData evpnConfig {
    1: required byte evpnConfigDataVersion = 1;
    2: required byte  routeType;
    3: required string rd;
    4: required long ethTag;
    5: required string esi;
    6: required byte tunnelType;
    7: required string tunnelId;
    8: required i32 label;
}
```

Proposed change in OpenDaylight-specific features

The following components within OpenDaylight Controller needs to be enhanced:

- ELAN Manager
- BGP Manager

Upon receiving the RT3 route, the elan instance associated to the evpn instance is identified. On that particular elan instance, external tep-ips field is added with the value of Originating Router's IP Address. This external tep-ips list is used for constructing the elan broadcast group.

The following actions will be performed against each step in the orchestration.

1) create evpn1, evpn2 instances.
2) associate network1 with evpn1 instance and network2 with evpn2.
3) spawn network1_vm1 on compute1. At this step RT3 route is advertized with the tep ip of compute1 (using rd of evpn1).
4) spawn network1_vm2 on compute1. No RT3 route needs to be advertized for compute1 as it is already done in step1.
5) spawn network1_vm3 on compute2. At this step RT3 route is advertized with the tep ip of compute2.
6) spawn network1_vm4 on compute2. No RT3 route needs to be advertized for compute2.
7) spawn network2_vm1 on compute1. At this step RT3 route is advertized with the tep ip of compute1 (using rd of evpn2).
8) spawn network2_vm2 on compute1. No RT3 route needs to be advertized for compute1.

9) spawn network2_vm3 on compute2. At this step RT3 route is advertized with the tep ip of compute2.
10) spawn network2_vm4 on compute2. No RT3 route needs to be advertized for compute2.
11) delete network1_vm1 from compute1. No action taken as compute1 still has a vm from network1 (network1_vm2)
12) delete network1_vm2 from compute1. Send withdraw RT3 route using rd of evpn1 and compute1 tep.
13) delete network2_vm1 from compute1. No action taken as compute1 still has a vm from network1 (network2_vm2)
14) delete network2_vm2 from compute1. Send withdraw RT3 route using rd of evpn2 and compute1 tep.
15) detach network1 from evpn1. Send withdraw RT2 for network1 (vm1, vm2, vm3 , vm4) . Send withdraw RT3 for compute1 and compute2 (using rd of evpn1).

Reboot Scenarios

This feature support all the following Reboot Scenarios for EVPN:

- Entire Cluster Reboot
- Leader PL reboot (PL : PayLoad Node : one of the cluster nodes where ODL is running in cluster)
- Candidate PL reboot
- OVS Datapath reboots
- Multiple PL reboots
- Multiple Cluster reboots
- Multiple reboots of the same OVS Datapath.
- Openstack Controller reboots
- Vm migration and evacuation

Configuration impact

N.A.

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

Fluorine

Alternatives

No Alternatives.

Usage

Features to Install

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

Implementation

Assignee(s)

Primary assignee: K.V Suneelu Verma <k.v.suneelu.verma@ericsson.com> Vyshakh Krishnan C H <vyshakh.krishnan.c.h@ericsson.com>

Work Items

https://jira.opendaylight.org/browse/NETVIRT-1243

Dependencies

Requires a DC-GW that is supporting EVPN RT3 on BGP Control plane.

Testing

Unit Tests

Appropriate UTs will be added for the new code.

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.

1) advertize RT3 route from datacenter gateway , verify that remote broad cast group on all computes which host that network is updated to include the datacenter gateway nexthop.
2) withdraw RT3 route from datacenter gateway , verify that remote broad cast group on all computes which host that network is updated to exclude the datacenter gateway nexthop.
3) bring up first elan vm on a compute, verify that RT3 route is advertized.
4) bring up second elan vm on the same compute, verify that RT3 route is not advertized again.
5) bring down first elan vm on a compute, verify that RT3 route is not withdrawn.
6) bring down last elan vm on a compute, verify that RT3 route is withdrawn.
7) Disassociate network from evpn, verify that all computes broadcast groups are updated to exclude the datacenter gateway nexthops.

Documentation Impact

This will require changes to User Guide and Developer Guide.

References

[1] BGP MPLS-Based Ethernet VPN

Table of Contents

IGMPv3 Snooping

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

IGMP Snooping is typically used to prevent flooding of multicast packets in a domain. In switches that do not support IGMP snooping, multicast packets would typically be flooded to all ports in the domain (VLAN, for example).

This flooding of multicast packets results in wasted bandwidth, as packets are sent on ports where there are no interested listeners of the particular multicast group address.

The main benefit of IGMP Snooping is to eliminate the flooding of packets by listening to IGMP messages between hosts and multicast routers, and build a database that will be used to forward multicast packets selectively to interested hosts and multicast routers.

This feature will implement IGMPv3 Snooping, as defined in RFC 4541.

Highlights of RFC 4541 that will be addressed by this feature:

- A snooping switch should forward IGMP Membership Reports only to those ports where multicast routers are attached.
- The switch supporting IGMP snooping must maintain a list of multicast routers and the ports on which they are attached.
- Packets with a destination IP address outside 224.0.0.X which are not IGMP should be forwarded according to group-based port membership tables and must also be forwarded on router ports.
- Packets with a destination IP (DIP) address in the 224.0.0.X range which are not IGMP must be forwarded on all ports.

This feature will not IPv6 MLD Snooping.

A global flag will be supported to enable/disable IGMP Snooping. Initially, this value will be read with ODL starts. Support for dynamic changes of this flag may not be initially supported.

IGMP Packet Processing and Forwarding

We need to keep track of where the multicast listeners and multicast routers are located.

The switch learns about interfaces by monitoring IGMP traffic. If an interface receives IGMP queries, this will indicate that a multicast router is attached to that interface, and the port will be added to the multicast forwarding table as a multicast-router interface (querier).

If an interface receives membership reports or igmp join for a multicast group, this will indicate that a multicast listener (host) is attached to that interface, and the port will be added to the multicast forwarding table as a group-member interface.

If an interface receives igmp leave for a multicast group, remove the port from the multicast forwarding table as a group-member interface.

Learning group members and multicast routers

IGMP Membership Report

- Add port to multicast Group Member BC group (210021)
- Add group address match criteria to multicast table (61)

IGMP Leave packet received (or timer expires):

- Remove port from multicast Group Member BC group (210021)
- Remove group address match rule from multicast table (61)

IGMP general query or group-specific query (any query)

- Add port to multicast Router BC group (220022)

Multicast Forwarding Rules

IGMP packets will be sent to ODL Controller learning, and then forwarded via sendIgmpPacketOut() as follows:

- Membership Report (leave) forwarded to all multicast router ports
- IGMP general query forwarded to all ports in domain
- IGMP group-specific query forwarded to group member ports

Multicast traffic that is not IGMP:

An unregistered packet is defined as an IPv4 multicast packet with a destination address which does not match any of the groups announced in earlier IGMP Membership Reports. A registered packet is defined as an IPv4 multicast packet with a destination address which matches one of the groups announced in earlier IGMP Membership Reports.

- destination address of 224.0.0.0/24 is flooded to all ports in domain
- Unregistered packet forwarded to all multicast router ports
- Registered packets forwarded to group member and multicast router ports

Problem description

The current behavior of IPv4 Multicast Packet Forwarding as of Oxygen Release:

IPv4 multicast

- Packets are flooded to all ports in domain when port-security is disabled.
- Packets are not forwarded when port-security is enabled, dropped by ACL rules.

IGMP

- Packets are flooded to all ports in domain when port-security is disabled.
- Packets are not forwarded when port-security is enabled, dropped by ACL rules.

As you can see above, when port security is disabled, multicast packets are flooded. When port security is enabled, multicast packets are dropped by ACL rules.

This IGMP Snooping feature, when enabled, will learn about multicast hosts and multicast routers from the IGMP conversation. These learned entries will be used to build a multicast forwarding database to forward IPv4 multicast packets as described in the Multicast Forwarding Rules section above.

Configuration to enable IGMP Snooping

From a user perspective, the following will need to be configured:

1. IGMP Snooping will need to be globally enabled in the config file. Default value is false.
2. IGMP protocol will need to be configured in security groups. Reference Security Group section above.

Use Cases

UC1 Multicast listener and sender on same compute node.

UC2 Mulicast listener and sender on different compute nodes. This will ensure IGMP Snooping works across internal tunnels.

UC3 Multicast listener on compute node and sender on vlan provider network.

UC4 Multicast listener on compute node and sender on flat provider network.

UC5 (Need to confirm if this is required) Multicast listener and sender on different compute nodes. One of the compute nodes is connected to L2GW. This will ensure IGMP Snooping works across external tunnels.

UC6 Multicast router on physical network (querier)

Proposed change

IGMP Snooping feature will send IGMP Packets to the ODL Controller. The IGMP messages will be parsed, and a multicast database will be built, consisting of multicast goup member ports and multicast router ports. This database will be used to populate Multicast Broadcast Groups, that will then be used to forward IPv4 multicast packets per the Multicast Forwarding Rules section above.

New Multicast Broadcast Groups (BC)

There will be a total of 3 broadcast groups/network needed for IGMP Snooping. These broadcast groups will be very similar to existing L2 BC groups. There would be a Local BC group per network (local ports only - packet ingress on tunnel port) and a Full BC group per network (local ports and tunnel ports - packet ingress on vm port).

- Multicast Router L/F - This group has the multicast router ports for the network.
- Multicast Group Member L/F - This group has the multicast group member ports for the network.
- All ports in domain L/F - This group already exists (Table 52, unknown DMACs).

Multicast Router Local (220021)

sudo ovs-ofctl add-group br-int -OOpenflow13 “group_id=220021,type=all,bucket=actions=set_field:0x0b->tun_id,resubmit(,55)”

Multicast Router Full (220022)

sudo ovs-ofctl add-group br-int -OOpenflow13 "group_id=220022,type=all,bucket=actions=group:220021,bucket=actions=set_field:0x5
>tun_id,load:0x600->NXM_NX_REG6[],resubmit(,220)

Multicast Group Local (210021)

sudo ovs-ofctl add-group br-int -OOpenflow13 "group_id=210021,type=all,bucket=actions=set_field:0x0a-
>tun_id,resubmit(,55)"

Multicast Group Full (210022)

sudo ovs-ofctl add-group br-int -OOpenflow13 "group_id=210022,type=all,bucket=actions=group:210021,bucket=actions=set_field:0x5
>tun_id,load:0x600->NXM_NX_REG6[],resubmit(,220)

Multicast Table (61)

Create a new IPv4 Multicast Table (61). This table will have rules that:

1. punt IGMP packets to the ODL Controller (learn and forward in ODL)
2. Match 224.0.0.0/24 and send to L2 Unknown DMACs table for L2 flooding to all ports in the domain (Table 48)
3. Match IPv4 multicast group address learned from IGMP Report/Join and send to Multicast Group BC Group and Multicast Router Group for forwarding
4. Send unmatched packets to Multicast Router Group for forwarding (unregistered multicast packet)

- sudo ovs-ofctl add-flow -OOpenflow13 br-int "table=61,priority=100,dl_type=0x0800,nw_proto=0x02 actions=CONTROLLER:65535"
- sudo ovs-ofctl add-flow -OOpenflow13 br-int "table=61,priority=100,dl_type=0x0800,nw_dst=224.0.0.0/24,actions=resubmit(,48
- sudo ovs-ofctl add-flow -OOpenflow13 br-int "table=61,priority=100,dl_type=0x0800,dl_type=0x0800,nw_dst=226.94.1.1,action
- sudo ovs-ofctl add-flow -OOpenflow13 br-int "table=61,actions=write_actions(group:220021)"

Multicast Group Table (62)

Need a way to send a packet to 2 BC groups. Thinking of using this table, and having something like this (better way to do this?):

- sudo ovs-ofctl add-flow -OOpenflow13 br-int "table=62,actions=write_actions(group:210021)"
- sudo ovs-ofctl add-flow -OOpenflow13 br-int "table=62,actions=write_actions(group:220021)"

Sample Flows

Flows to get Multicast packets to Multicast Table(61) from ARP Table (43)

- sudo ovs-ofctl add-flow -OOpenflow13 br-int table=43,priority=100,dl_type=0x0800,nw_proto=0x02,actions=goto_table:61
- sudo ovs-ofctl add-flow -OOpenflow13 br-int "table=43,priority=90,dl_type=0x0800,dl_dst=01:00:5e:00:00:00/ff:ff:ff:00:00:00,a

NOTE: The 2 rules above would also have to be added to Internal Tunnel Table (36) and External Tunnel Table (38).

Flows to get Multicast packets to Multicast Table(61) from Internal Tunnel Table (36)

- sudo ovs-ofctl add-flow -OOpenflow13 br-int table=36,priority=100,dl_type=0x0800,tun_id=0x5dc,nw_proto=0x02,actions=goto
- sudo ovs-ofctl add-flow -OOpenflow13 br-int "table=43,priority=90,dl_type=0x0800,dl_dst=01:00:5e:00:00:00/ff:ff:ff:00:00:00,tu

Flow to get Multicast packets to Multicast Table(61) from External Tunnel Table (38)

- sudo ovs-ofctl add-flow -OOpenflow13 br-int table=38,priority=100,dl_type=0x0800,tun_id=0x5dc,nw_proto=0x02,actions=goto
- sudo ovs-ofctl add-flow -OOpenflow13 br-int "table=36,priority=90,dl_type=0x0800,dl_dst=01:00:5e:00:00:00/ff:ff:ff:00:00:00,tu

Security Groups

Configure ACL to allow IGMP packet to MC Router port

In ODL, when an IGMP Query is received, update port config for which Query packet was received, and add allowed address pairs to multicast router port. Command line example here:

openstack port set –allowed-address ip-address=224.0.0.22,mac-address=01:00:5e:00:00:16 208b35fd-4c61-4d63-93f5-ab08e25a3560

Update port for IPv4 multicast address group, learned from IGMP Report/Join

When ODL receives IGMP Join/Membership Report, update the config for the port to allow the port to receive the IPv4 multicast packets as specified in the IGMP packet.

openstack port set –allowed-address ip-address=226.94.1.1,mac-address=01:00:5e:5e:01:01 74ab3b8e-1b95-4fef-a60d-295856b714b6

Configure security groups to allow IGMP packets

Adding support for IGMP protocol to security groups is required so that ACL tables will allow IGMP packets to egress the switch.

Here is an example of adding a rule to security group to allow igmp. This command adds rules to ACL tables to allow IGMP to egress.

- openstack security group rule create goPacketGo –ingress –ethertype IPv4 –protocol igmp
- openstack security group rule create goPacketGo –egress –ethertype IPv4 –protocol igmp

This adds a rule to table 240 that allows IGMP pkts to proceed through pipeline, going to table 241. Sample flow:

cookie=0x6900000, duration=82.942s, table=240, n_packets=8, n_bytes=432, priority=61010,ip,reg6=0xa00/0xfffff00,dl_dst=01:00:5e:00:00:16,nw_dst=224.0.0.22 actions=goto_table:241

ODL Security groups do not currently support IGMP. As such, some small code changes are required to support IGMP. For example, in ODL Oxygen, if you issue the command:

- openstack security group rule create goPacketGo –ingress –ethertype IPv4 –protocol igmp

an error is thrown from ODL neutron, saying protocol igmp is not supported. There is also a small change required in ACL to add support for igmp in security groups. I have the fix for this in my sandbox, and will be pushing this patch as part of this feature.

Pipeline changes

Add rules to ARP Table (43) to send IPv4 multicast packets to new IPv4 Multicast Table(61). Currently, ARP Table (43) sends packets to L2 Pipeline (48) if not ARP. We do not want IPv4 multicast packets to be processed in L2 Pipeline (and flooded to all ports in the network).

In table 43:

- arp check -> group 5000 (existing)
- igmp check -> table 61 (new)
- IPv4 MC check -> table 61 (new)
- goto table 48 (existing)

Add rules to Internal Table (36) to do the same as above:

In table 36:

- igmp check -> table 61 (new)
- IPv4 MC check -> table 61 (new)
- goto table 51 (existing)

Add rule to External Table (38) to do the same as above:

In table 38:

- igmp check -> table 61 (new)
- IPv4 MC check -> table 61 (new)
- goto table 51 (existing)

Yang changes

Add new yang to enable/disable igmp snooping.

module igmpsnooping-config { yang-version 1; namespace "urn:opendaylight:params:xml:ns:yang:igmpsnooping:config"; prefix "igmpsnooping-config";

description "Service definition for igmpsnooping module";

revision "2018-04-20" {

description "Initial revision";

}

```
    container igmpsnooping-config {
        leaf controller-igmpsnooping-enabled {  description "Enable igmp snooping on the controller";
            type boolean;
            default false;
        }
    }
}
```

Configuration impact

Adding new option to enable/disable igmp snooping for the controller.

Clustering considerations

TBD

Other Infra considerations

N/A

Security considerations

N/A

Scale and Performance Impact

Would be good to do some scale testing with large number of IGMP listeners/senders to determine if there is any negative impact on performance. Be sure to test with scale where there are lots of IGMP Report/Joins/Leaves to see if there are performance issues with IGMP punting to ODL Controller

Targeted Release

Flourine

Alternatives

N/A

Usage

User would have to enable IGMP Snooping in xml/rest before starting ODL.

User would have to configure Security Group for port and add IGMP protocol to Security Group.

Then, user should be able to spin up VMs on compute nodes, have some listeners, some senders, and the multicast listeners should be able to receive IPv4 Multicast packets from the senders.

Features to Install

odl-netvirt-openstack

REST API

CLI

Implementation

Assignee(s)

Primary assignee: <Victor Pickard>, <vpickard>, <vpickard@redhat.com>

Work Items

- Write blueprint.
- Update Pipeline for IGMP/IPv4 MC packet processing
- **Add code to:**
  - Listen for IGMP Packets
  - Create, manage and populate Multicast BC Groups learned from IGMP
  - Add rules to tables 43,36,38,61,62 for IGMP/IPv4 MC pkts
  - Test using IPerf
  - Add tests to CSIT

Dependencies

None

Testing

Setup Openstack/ODL deployment and test use cases as follows:

Unit Tests

TC 1

Start a multicast listener - sends IGMP Report/Join pkts iperf -s -u -B 226.94.1.1 -i 1

Start a multicast source. Sends stream of UDP 1Mbps to 226.94.1.1 iperf -c 226.94.1.1 -u -t 3600

Verify multicast listener receives packets from sender for all use cases. Verify listener ports are added to Multicast Group BC Group. Verify IPv4 multicast traffic is only sent to registered listeners (not flooded).

TC 2

Start (or simulate) a multicast router on Flat Provider network. We want to see IGMP Query messages arrive from provider network port.

Verify multicast router port is added to Multicast Router BC Group.

Send IPv4 multicast traffic on the network. Verify that registered and unregistered packets forwarded to multicast router port.

TC 3

Start multicast listeners and multicast routers on network.

Send IPv4 Traffic with DIP 224.0.0.X/24. Verify traffic is flooded to all ports in domain.

TC 4

Start multicast listeners and multicast routers on the network. Send IGMP Membership Report. Verify packet is forwarded to all multicast router ports on the domain.

TC 5

Start multicast listeners and multicast routers on the network. Send IGMP general query. Verify packet is forwarded to all ports in domain.

TC 6

Start multicast listeners and multicast routers on the network. Send IGMP group-specific query. Verify packet is forwarded to group member ports.

Integration Tests

CSIT

Add IGMP/IPv4 Multicast test cases to CSIT to address Unit Test cases above.

Documentation Impact

Vpickard to work with Doc team to add configuration/overview/operation of IGMP Snooping.

References

[1] OpenDaylight Documentation Guide

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

[3] IGMP Snooping Overview

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

Table of Contents

Enhance Robustness of L3VPN

https://git.opendaylight.org/gerrit/#/q/topic:l3vpn-robustness

Enhance current L3VPN to improve its robustness for specific use-cases and also enable it to consistently work with Neutron BGPVPN initiated orchestration lifecycles.

Problem description

Witnessing issues faced in L3VPN with production deployments of OpenDaylight, it was gradually occurring that piecemeal fixing issues in L3VPN (and all its related modules) unfortunately wouldn't scale.

It was ascertained during fixing such issues (ie., providing patches for production) that there was scope for improving robustness in L3VPN.

There were specific cases that may (or) may not succeed most of the time in L3VPN.

1. VM Migration
   - Specifically when VM being migrated is a parent of multiple IPAddresses (more than 4) where some are V4 and some V6.
2. VM Evacuation
   - Mutliple VMs on a host, each evacuated VM being a parent of multiple IPAddresses, and the host made down.
3. Speed up Router-association and Router-disassociation to a VPN lifecycle
4. Graceful ECMP handling (i.e, ExtraRoute with multiple nexthops)
   - Modify an existing extra-route with additional nexthops.
   - Modify an existing extra-route by removing a nexthop from a list of configured nexthops.
   - Boot a VM, which is a nexthop for a pre-configured extra-route
   - Delete a VM, which is a nexthop for an extra-route
5. Robust VIP/MIP Handling
   - MIP movement across VMs on different computes with VRRP configuration.
   - VM (holding MIP) migration.
   - MIP traffic impact on ODL cluster reboot.
   - MIP traffic impact on OVS disconnect/connect.
   - MIP Aliveness monitoring sessions are not persistent.

If we take a closer look, the first three points (points 1, 2 and 3) require vpn-interface-based serialization across the cluster for consistently driven control-plane + data-plane semantics.

Similarly the next two points above (points 4 and 5), we will notice that they require IP-driven serialization across the cluster for consistently driven control-plane + data-plane semantics.

However, today the entire VPN Engine unfortunately is vpn-interface-driven and not IP-driven. As a result, points 4 and 5 would never be realized with constant consistentcy.

Point 3 case deals with vpn-interface (and so IP Addresses within them) movement across two VPNs, one being router-vpn and other being the bgpvpn-with-that-router. And this is extremely slow today and will be unable to catch up with orchestration of any kind (be it via ODL RPCs) or via Openstack Neutron BGPVPN APIs in a scaled cloud.

The initiative of this specification is to improve intrinsic reliability and behavioural consistency within the L3VPN Engine by explicitly enforcing IP Serialization wherever required. In addition to that provide a design that can handle faster (but consistent) movement of IPAddresses between router-vpn and bgpvpn-with-that-router.

In scope

Out of scope

Use Case Summary

All the below use-cases have to be considered with a (3-node Openstack Controller + 3-node ODL cluster).

UC 1: VM Migration (cold migration)

This use-case is to ensure VM Cold Migrations are made robust within L3VPN Engine. If you notice this mimics a vpninterface moving to different designated location in the cloud. Has the following sub-cases: UC 1.1 - Migrate a single dualstack VM residing on a vpn UC 1.2 - Migrate multiple dualstack VMs residing on different vpns

UC 2: Evacuation of VM (voluntary / involuntary shutdown of the compute host)

This use-case is to ensure VM evacuation cycles are made robust within L3VPN Engine. This mimics a vpninterface moving to different location in the cloud, but triggered via failing a compute node. Has the following sub-cases: Has the following sub-cases: UC 2.1 - Evacuate a single dualstack VM from a single vpn from a compute host UC 2.2 - Evacuate multiple dualstack VMs across multiple vpns from the same compute host

UC 3: An Extra-route serviced by one or more VMs

This use-case is to ensure ECMP handling robustness within L3VPN Engine. This mimics an ipv4 address being reachable from multiple nexthops (or multiple vpninterfaces). Has the following sub-cases: UC 3.1 - Append a nexthop to an existing IPv4 extra-route UC 3.2 - Remove a nexthop from an existing IPv4 extra-route with multiple nexthops UC 3.3 - Clear away an IPv4 extra-route with multiple nexthops from a router, altogether UC 3.4 - Delete the VM holding the nexthop of an extra-route UC 3.5 - Delete a VM and add another new VM holding the same nexthop-ip of an existing extra-route

UC 4: Robust MIP Handling

This use-case is to ensure VIP/MIP handling robustness within L3VPN Engine. Has the following sub-cases: UC 4.1 - Move a MIP from one VM port to another VM port, wherein both the VMs are on the same subnet. UC 4.2 - When a MIP is shared by two VM ports (active / standby), delete the VM holding the MIP. UC 4.3 - Migrate a VM which is holding MIP. UC 4.4 - MIP traffic impact on ODL cluster reboot. UC 4.5 - MIP traffic impact on OVS disconnect/connect. UC 4.6 - MIP Aliveness monitoring sessions are not persistent.

UC 5: IP visibility across related VPNs

This use-case is to ensure that ip reachability across two related vpns is made robust within L3VPN Engine. Has the following sub-cases: UC 5.1 - Peering VPNs being configured and initiating migration of VMs on one of the peering VPNs UC 5.2 - Delete peering VPNs simultaneously

In general the above tests sufficiently trigger IP Serialization enforcement and this will enable us to remove the 2 seconds sleep() from within the ArpProcessingEngine (ArpNotificationHandler).

UC 6: Speed up Router-Association and Router-Disassociation

This use-case is about attempting to speed up the swap of Router into L3VPN and vice-versa. The idea is to eliminate the 2 seconds sleep present within swap logic, thereby increasing the rate of servicing vpn-interfaces in the router for the swap cases.

Proposed change

We brainstormed many proposals (or ways) to enforce IP Serialization (within the scope of router / vpn) and ended up with agreeing with the following proposal.

The proposal chosen was about enforcing IP Serialization by processing all the VPNs (and then all VPNInterfaces and IPAddresses within them) in the ODL system through a single node.

Please note that the L3VPN Service consists of the following engines: * VPNEngine (everything under VpnManager-Api and VpnManager-Impl) * FIBEngine (everything under FibManager-Api and FibManager-Impl) * VPN-TunnelEngine (TunnelInterfacexxxListener and TEPListener) * SubnetRouteEngine (VpnSubnetRouteHandler) * ARPLearningEngine (ArpNotificationHandler and AlivenessMonitor) All the above engines will be effected/affected as part of implementing following proposals.

Section 1

This section talks about enforcing IP Serialization for extra-routes. The main goal of the proposed implementation in this section is two-fold:

- To enforce IP Serialization for Extra-Route (or Static-Route) IP Addresses in a plain-router domain (or) a BGPVPN routing domain
- To eliminate the Thread.sleep enforced for extra-route hanlding in NeutronRouterChangeListener and also to remove the clusterLocks introduced in NextHopManager.
- The implementation enforces IP Serialization in the following way:

1. It introduces a new yang container, ‘extra-route-adjacency’ to hold all the configured extra-routes. The container model is defined in the Yang Changes section.
2. All the extra-routes configured on a router will now be written to ‘extra-route-adjacency’ by Neutronvpn. This will eliminate adding the extra-routes as adjacencies to their respective nexthop interfaces, i.e, Extra-Routes will never be represented as vpn-interface adjacencies going forward. All information about the configured extra-routes will only be present in ‘extra-route-adjacency’.
3. A new ‘extra-route-adjacency’ listener will be responsible for creating FIB, and updating other ECMP related datastores for the extra-routes. This listener will be a clustered one, that will use EOS to execute only on the owner node for L3VPN Entity. Translation of FIB to flows remains unchanged.

4. This section will also handle configuring/unconfiguring pre-created extra-routes on VMs that are booted at a later point in time. This is done by a new service for extra-route to port binding service, that will be executed on every VM addition/deletion.

Section 2

This section talks about enforcing IP Serialization for MIPs.

- The implementation enforces IP Serialization in the following way:

1. A new MIP-Engine will be implemented which will listen on 'mip-route-adjacency' config container. The container model is defined in the Yang Changes section.
2. When GARP/ARP Responses messages are punted to ODL controller, ArpNotificationHandler will write MIP info to 'mip-route-adjacency'.This listener will be a clustered one, that will use EOS to execute only on the owner node for L3VPN Entity.
3. MIP-Engine will be responsible for creating/deleting the FIB entry. It will also trigger the Aliveness Manager to start/stop the Monitoring session for each learnt IPs.
4. Upon ODL cluster reboot , MIP-Engine will trigger the Aliveness Manager to start the monitoring as Aliveness Manager does not persist the Monitoring sessions.

This section talks about making VPNEngine robust enough to handle following MIP related scenario:

1. Currently when OVS disconnects from ODL controller, AlivenessManager does not detect it. It tries to send Arp Request message to OVS hosting MIP and fails. Eventually AlivenessManager thinks MIP is dead and forces VPNEngine to withdraw the MIP route from DC-Gateway. It leads to MIP traffic loss eventhough MIP is active on OVS.
2. VPNEngine listens on 'Nodes' container to detect OVS disconnect/connect.VPNEngine will pause the Monitoring session for MIP when OVS disconnects and will resume the Monitoring session for MIP when OVS connects.

Section 3

Details about the proposal is given below to facilitate implementors.

1. The service name used will be 'L3VPN' and for that service name the VPN-Feature will choose a leader node from the 3 nodes in the ODL cluster. If there is only one node, then that node will be considered the leader.
2. How and which node is chosen as a leader will be decoupled from the VPN Engine. All the engines within the L3VPN Service will only be consumers of interface exposed by a new entity 'VpnLeadershipTracker' ,and this new entity will be responsible to tag the leader node by using 'L3VPN' service name as the key. While initially we will use cluster-singleton to choose a node as the leader for 'L3VPN' service, the tracker will also be expanded as a later review to choose that node which is holding the Default Operational Shard as the leader, as #ofreads and #ofwrites to the Default Operational Shard is higher by all the engines within the L3VPN Service.
3. A new *VPNEvent* POJO will be made available to store immutable information pertaining to an event of interest to the L3VPN service. This pojo is not a datastore entity and will be constructed dynamically only on the node which is the current leader of service name 'L3VPN'. Also the VPNEvent POJO will contain all the information required to process an event by all the engines within the L3VPN service.
4. All along in today's L3VPN Service, the FIBEngine is responsible for processing a VRFEntry creation/deletion/updation. Fundamentally, converting an action on a VRFEntry to flow additon/ group addition/ flow removal / group removal is driven by FIBEngine (aka VrfEntryListener). Going foward, while the FIBEngine will continue to hold this responsibility, the implementation will change the lifecycle for VRFEntry handling being triggered synchronously by the VPNEngine instead of asynchronously driven by the

FIBEngine. This will provide the benefit of passing all the required information synchronously to the methods in the FIBEngine for VRFEntry handling by the VPNEngine. The VRFEntry would become an artifact instead of a trigger source. This type of design also enables use of DJC to enforce IP Serialization within a given VPNInstance. This change additionally provides elimination of backpulls from the FIBEngine towards VPNEngine and other modules for non-BGP-imported-rows

5. Other than imported BGP routes, all other types of route processing (VM ports, exported VM ports, extra-routes, MIPs), will be done in the way quoted in point 4 above. The BGP imported routes from other neighbours will continue to be driven from within the VRFEntryListener.
6. Re-use as much existing handlers within the engines of L3VPN Service in order to contain the robustness effort
7. All the VPN Datastores will continued to be made available the same way for other external interfaces and consumers of external interfaces to remain unaffected. Most specifically implementing this area will not effect NATEngine, InterVPNLinksEngine and BGPEngine.
8. All BGP Advertisements and Withdrawals (for all routes managed by ODL itself - i.e., non-BGP-imported routes), will be done within the VRFEntryListener rather than by the VPN Engine.
9. The VPNInstance creation/deletion/updation handling will be driven on whichever node is the leader for 'L3VPN' service as per VpnLeadershipTracker. The handler for VPNInstance will continue to use JobCoordinator keyed on 'VPN-<vpn-name>' to process creation/updation/deletion.
10. The VpnInterface creation/deletion/updation handling will also be driven on whichever node is the leader for 'L3VPN' service as per VpnLeadershipTracker. The following jobKeys for JobCoordinator will be applied for VpnInterfaceEvents:

    a. The jobKey of 'vpnid-dpnid' will be used for populateFibOnNewDpn and cleanupFibOnNewDpn (and its related methods) Here vpnid is dataplane representation of the VPN (and not the control plane vpnuuid).

    b. b1. The jobKey of 'VPN-<vpn-interface-name>' will continued to be used to serialize all events for a specific vpn-interface.

    b2. Within the 'VPN-<vpn-interface-name>' run, a nested job will be fired with key of 'VPN-rd-prefix' to serialize h
    run for all IPAddresses within a given vpn (identified by rd). The 'prefix' here can be primary-ip, extra-route-prefix, or a discovered mip-prefix.

There are *Thread.sleep* in mulitiple places inside ODL Netvirt projects. This spec attempts to eliminate the sleep invocations in the following files: VpnInterfaceManager - sleep introduced to allow batch movement of vpn-interfaces from router-to-bgpvpn and vice-versa ArpNotificationHandler (arpcachetimeout) - The timeout here was actually added to facilitate vpn-lifecycle for a MIP. NexthopManager - ClusterLock with sleep used to safe management of l3nexthop (createLocalNextHop).

There are still sleeps in other services like Elan, L2Gateway etc and those sleep removals need to be pursued by respective components.

Pipeline changes

None

Yang changes

Changes will be needed in `l3vpn.yang` , `odl-l3vpn.yang` , `odl-fib.yang` and `neutronvpn.yang` to support the robustness improvements.

ODL-FIB YANG CHANGES

Listing 4: odl-fib.yang

```
--- a/fibmanager/api/src/main/yang/odl-fib.yang
+++ b/fibmanager/api/src/main/yang/odl-fib.yang
@@ -100,15 +100,19 @@ module odl-fib {

 container fibEntries {
     config true;
-        list vrfTables{
-            key "routeDistinguisher";
-            leaf routeDistinguisher {type string;}
-            uses vrfEntries;
-            uses macVrfEntries;
-        }
+        list vpnNames {
+            key vpnName;
+            leaf vpnName { type string; }
+            list vrfTables{
+                key "routeDistinguisher";
+                leaf routeDistinguisher {type string;}
+                uses vrfEntries;
+                uses macVrfEntries;
+            }

-        container ipv4Table{
-            uses ipv4Entries;
+            container ipv4Table{
+                uses ipv4Entries;
+            }
     }
 }
```

ODL-L3VPN YANG CHANGES

Listing 5: odl-l3vpn.yang

```
--- a/vpnmanager/api/src/main/yang/odl-l3vpn.yang
+++ b/vpnmanager/api/src/main/yang/odl-l3vpn.yang
@@ -184,10 +184,13 @@ module odl-l3vpn {
             }
     }

-    container evpn-rd-to-networks {
+    container evpn-to-networks {
         description "Holds the networks to which given evpn is attached";
-        list evpn-rd-to-network {
```

(continues on next page)

(continued from previous page)

```
+        list evpn-to-network {
-           key rd;
+           key vpn-name;
+           leaf vpn-name {
+             type string;
+           }
            leaf rd {
              type string;
            }
@@ -261,7 +264,7 @@ module odl-l3vpn {
     container vpn-instance-op-data {
         config false;
         list vpn-instance-op-data-entry {
-           key vrf-id;
+           key vpn-instance-name;
            leaf vpn-id { type uint32;}
            leaf vrf-id {
              description


@@ -699,4 +699,16 @@ module odl-l3vpn {
             leaf-list nexthop-key { type string; }
         }
     }
+
+    container extra-route-adjacency {
+        config false;
+        list vpn {
+            key "vpn-name";
+            leaf vpn-name { type string; }
+            list destination {
+                key "destination-ip";
+                leaf destination-ip { type string; }
+                list next-hop {
+                    key "next-hop-ip";
+                    leaf next-hop-ip { type string; }
+                    leaf interface-name { type string; }
+                }
+            }
+        }
+    }
+
+    container mip-route-adjacency {
+        config true;
+        list mip-ip {
+            key "vpn-name mip-ip";
+            leaf mip-ip { type string; }
+            leaf vpn-name { type string; }
+            leaf port-name { type string; }
+            leaf mac-address { type string; }
+            leaf creation-time { type string; }
+        }
+    }
```

Solution considerations

Configuration impact

Clustering considerations

The feature should operate in ODL Clustered (3-node cluster) environment reliably.

Other Infra considerations

N.A.

Security considerations

N.A.

Scale and Performance Impact

Not covered by this Design Document.

Targeted Release

Fluorine.

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

Implementation

Assignee(s)

Primary assignee:

Vivekanandan Narasimhan (n.vivekanandan@ericsson.com)

Work Items

The Trello cards have already been raised for this feature under the l3vpn-robustness.

#Here is the link for the Trello Card: #https://trello.com/c/Tfpr3ezF/33-evpn-evpn-rt5

Dependencies

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

New CSIT testcases will be added for this feature, as this starts to use Neutron BGPVPN APIs and makes it official for ODL platform.

Documentation Impact

References

Table of Contents

Neutron QoS (Quality of Service)

https://git.opendaylight.org/gerrit/#/q/topic:qos

Neutron provides QoS[26] feature which is partially supported in OpenDaylight today. This feature is to bring ODL implementation on parity with Neutron QoS API, esp OVS Agent implementation as detailed in[3]. This will require changes in networking-odl, Neutron Northbound and Netvirt.

Note:

- Ingress/Egress in Neutron is from VM's perspective.
- Ingress/Egress in OVS/ODL is from switch perspective.
- Egress from VM is Ingress on switch and vice versa

---

[2] Neutron QoS
[6] QoS Spec
[3] Neutron Configuration Guide - QoS

Problem description

Support for QoS was added to OpenDaylight in Boron release and it was later updated during Carbon release[1]. Neutron has updated QoS since then to support more features and they are already implemented in Neutron OVS Agent.

Following table captures current state of QoS support in Neutron and OpenDaylight as well as comparison with current support in OVS Agent.

Feature	N-ODL	ODL	OVS Agent
Bandwidth Limit Egress	✓	✓	✓
Bandwidth Limit Ingress			✓
Min Bandwidth Egress			SRIOV only
Min Bandwidth Ingress			
DSCP Marking Egress		✓	✓
DSCP Marking Ingress			
FIP QoS			✓

Use Cases

Neutron QoS API support for Bandwidth Limit Ingress and FIP QOS will require adding followiung changes to networking-odl driver and OpenDaylight:

- Bandwidth Limit Ingress
- FIP QoS

Following use cases will not be supported:

- Min Bandwidth Ingress/Egress
- DSCP Marking Ingress

Proposed change

This will require changes to multiple components.

Bandwidth Limit Ingress

Ingress Limit is not as simple as egress. This will require creation of a QoS Profile and adding the port to that profile[5]. Then we appply limiters on that queue. OVSDB Plugin already supports creating QoS Profiles. OpenFlow rules for that port will then direct traffic to that queue.

Creation of profile varies for DPDK vs Kernel datapath[9]

Listing 6: Sample OVS QoS configuration Kernel datapath

```
ovs-vsctl -- \
    add-br br0 -- \
    add-port br0 eth0 -- \
    add-port br0 vif1.0 -- set interface vif1.0 ofport_request=5 -- \
```

(continues on next page)

[1] Quality of Service - Oxygen spec
[5] OVS QoS FAQ
[9] http://docs.openvswitch.org/en/latest/topics/dpdk/qos/

(continued from previous page)

```
    add-port br0 vif2.0 -- set interface vif2.0 ofport_request=6 -- \
    set port eth0 qos=@newqos -- \
    --id=@newqos create qos type=linux-htb \
        other-config:max-rate=1000000000 \
        queues:123=@vif10queue \
        queues:234=@vif20queue -- \
    --id=@vif10queue create queue other-config:max-rate=10000000 -- \
    --id=@vif20queue create queue other-config:max-rate=20000000
```

Listing 7: Sample OVS QoS configuration DPDK

```
ovs-vsctl set port vhost-user0 qos=@newqos -- \
    --id=@newqos create qos type=egress-policer other-config:cir=46000000 \
    other-config:cbs=2048`
```

Networking-odl

With[7] nothing else should be needed in n-odl.

Neutron Northbound

Direction field is already present in neutron-qos.yang so no yang changes needed for this. However, SPI still needs to add code to populate mdsal with direction field. Corresponding changes to NeutronQos POJO will also be done.

OVSDB

Nothing extra needs to be done here, OVSDB already supports QoS queues.

Genius

To support Ingress limiting, QoS needs to bind as an Egress service in Genius. For this a new `QOS_EGRESS_TABLE [232]` will be added.

Note: TBD - QoS Egress Service priority

Netvirt

Existing QoS code in Netvirt will be enhanced to support Ingress rules, bind/unbind Egress service, create OVS QoS profiles and add port to the QoS queues.

[7] https://review.openstack.org/#/c/567626/

FIP QoS

Similar to Ingress Limit, QoS profiles will be used to create queue per FIP and then use OpenFlow flows to direct traffic to the specific queue[4].

Networking-odl

Nothing should be needed once[7] and[8] are merged.

NeutronNorthbound

qos-policy-id will be added to `L3-floatingip-attributes` in neutron-L3.yang

OVSDB

Nothing needs to be done here.

Genius

A new `QOS_FIP_INGRESS [91]` table will be added to Ingress L3 pipeline to add set queue for Egress Limit. For Ingress Limit, `QOS_FIP_EGRESS [233]` will be added to set queue.

Netvirt

FloatingIp listeners will be enhanced to track QoS configuration and invoke QoS API to configure flow rules for the FloatingIp. QoS API will create OVS profiles for QoS rules and when applied to FIP or port, will program appropriate flows.

Pipeline changes

A new QoS Egress table will be added to support for Ingress rules on port and another for FIP.

Table	Match	Action
QoS FIP Ingress [91]	Ethtype == IPv4 or IPv6 AND IP	SetQueue
QoS Port Egress [232]	Ethtype == IPv4 or IPv6 AND LPort tag	SetQueue
QoS FIP Egress [233]	Ethtype == IPv4 or IPv6 AND IP	SetQueue

[4] Floating IP Rate Limit
[8] https://review.openstack.org/#/c/504182/

Yang changes

No yang changes needed for Ingress support in QoS. Ingress is already supported in NeutronNorthbound and[7] will enable it as part of networking-odl.

Changes will be needed to neutron-qos-ext.yang to augment floating-ip.

Listing 8: neutron-qos-ext.yang

```
augment "/neutron:neutron/neutron:floating-ips/neutron:ip" {
    description "This module augments the floating-ips container
        in the neutron-floating-ips module with qos information";
    // ext:augment-identifier value needs to unique as name of the generated classes
    // is derived from this string
    ext:augment-identifier "qos-floating-ip-extension";
    leaf qos-policy-id {
        description "The floating-ips to which the Qos Policies can be applied";
        type yang:uuid;
    }
```

Configuration impact

This doesn't introduce any new configuration parameters.

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

None

Targeted Release

Flourine

Alternatives

N.A.

Usage

Nothing extra needs to be done to enable these features other than already captured in [1].

Features to Install

odl-netvirt-openstack

REST API

None in ODL other than yang changes. For Neutron API refer [2]__

CLI

34

Implementation

Assignee(s)

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

Primary assignee: Vishal Thapar, <#vthapar>, <vthapar@redhat.com>

Other contributors: TBD. (volunteers welcome)

Work Items

1. Changes in networking-odl (mostly testing)
2. Changes in NeutronNorthbound
3. Add listeners in Netvirt
4. Genius changes for modified pipeline
5. Netvirt implementation for Bandwidth Limit Ingress
6. Netvirt changes for FIP Egress
7. Netvirt changes for FIP Ingress
8. Add CSIT (depends on CSIT for existing QoS features)

9. Documentation

Dependencies

This has dependencies on other projects:

- Neutron - Rocky
- Networking-Odl - Rocky
- Neutron Northbound - Flourine
- OVSDB - Flourine
- Genius - Flourine

Testing

This will update existing tests for QoS for newer use cases

Unit Tests

Existing UTs, if any, will be updated to provide coverage for new code.

Integration Tests

N.A.

CSIT

QoS use cases are currently not covered by CSIT. Once CSIT is added for QoS, more test cases will be added to cover these use cases.

Once[7] and[8] are done, we can also enable QoS tempest tests in existing Netvirt CSIT.

Documentation Impact

Existing documenation will be updated to reflect the new changes.

References

Table of Contents

Support FIPs for Octavia VIPs

Problem description

An Octavia VIP is a neutron port that is not bound to any VM and is therefor not added to br-int. The VM containing the active haproxy sends gratuitous ARPs for the VIP's IP and ODL intercepts those and programs flows to forward VM traffic to the VMs port. Note that this is my understanding of how this all works, I have not validated it, the opener of this bug confirms that it works.

The ODL code responsible for configuring the FIP association flows on OVS currently relies on a southbound openflow port that corresponds to the neutron FIP port. The only real reason this is required is so that ODL can decide which switch should get the flows. See FloatingIPListener#createNATFlowEntries. In the case of the VIP port, there is no corresponding southbound port so the flows never get configured.

Use Cases

FIP assigned to an Octavia loadbalancer

Proposed change

The basic solution is to use the OF packet-in event to learn the dpn where the NAT flows need to be programmed.

Overview of code changes:

1. Refactor NatInterfaceStateChangeListener into two classes: (a) NatInterfaceStateChangeListener which just receives events. (b) NatSouthboundEventHandlers which contains the logic invoked for when an interface state changes (or a garp is received)
2. Implementation of NatArpNotificationHandler which receives the garps and invokes the correct methods in NatSouthboundEventHandlers
3. neutron-vip-state yang model which is used together with VipStateTracker (DataObjectCache) to manage state of the discovered VIPs. This is required in cases where the associated Ocatavia Amphora VM changes to a different compute node. In this case the existing flows must be removed from the odl compute node.
4. Tweak VIP learning code to accept neutron ports that are owned by "Octavia", previously the code assumed no neutron port ever needed to be learned.

Pipeline changes

None

Yang changes

```
:caption: odl-nat.yang

 container neutron-vip-states {
     config false;
     list vip-state {
         key ip;
         leaf ip {
             type string;
         }
         leaf dpn-id {
             type uint64;
         }
         leaf ifc-name {
             type string;
         }
     }
 }
```

Configuration impact

None

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

Unlikely to have any real impact

Targeted Release

Flourine with a backport as far as Oxygen

Alternatives

N/A

Usage

This feature should "just work" with Octavia. No special usage is required.

Features to Install

odl-netvirt-openstack

REST API

None

CLI

None

Implementation

Assignee(s)

Josh Hershberg, jhershbe, jhershbe@redhat.com

Work Items

https://git.opendaylight.org/gerrit/#/c/75281/

https://git.opendaylight.org/gerrit/#/c/75248/

Dependencies

None

Testing

Unit Tests

None

Integration Tests

As this is a bug and we are rushing to fix it for now testing will be done manually

CSIT

Yes, we really should in the near future.

Documentation Impact

None

References

None

Table of Contents

Support Neutron BGPVPN API on OpenDaylight L3VPN service

https://git.opendaylight.org/gerrit/#/q/topic:neutron-bgpvpn-api-support

Enhance current L3VPN service to realize VPNs consistently when VPNs are managed through the Openstack Neutron BGPVPN Plugin.

Problem description

As the current L3VPN service stands end-of-Oxygen release, it is able to realize a VPN orchestrated by Neutron BGPVPN API.

However, this orchestration is subjected heavily to timing and nature of how the orchestration was performed.

To explain, piecemeal creating a Neutron BGPVPN using Openstack BGPVPN REST API (or) CLI and making sure it is realized and then deleting the same Neutron BGPVPN (via RESTAPI or CLI) would succeed.

However, when the orchestration pattern on Openstack BGPVPN API is turned to be driven through an automation script (or) if the script uses Heat templates to automate and manage BGPVPNs, the runs will expose failure to properly realize such orchestrated VPNs in the OpenDaylight control plane (and thence data plane too).

The failures themselves will go unnoticed to the Openstack Orchestrator (or user) since the Openstack Neutron will provide a consistent view of available (or active) VPNs while the VPNs being realized in OpenDaylight would be either less than the active VPNs (or) might sometimes even be more (due to stale VPNs not cleaned up).

In scope

The following types of VPNs are scoped into this effort: a. Network-associated L3 BGPVPNs Networks can have either IPv4 (or) IPv6 (or) both IPv4 and IPv6 subnets in them. b. Router-associated L3 BGPVPNs Router can have either IPv4 (or) IPv6 (or) both IPv4 and IPv6 subnets attached to them. c. Plain Neutron Router (Internal VPN) Router can have either IPv4 (or) IPv6 (or) both IPv4 and IPv6 subnets attached to them. d. Plain External Network (Internal VPN) External network used as a VPN to provide external connectivity over FLAT / VLAN mechanisms. e. ExternalNetwork-associated L3 BGPVPNs L3BGPVPN representing connectivity to external networks via MPLSOverGRE tunnels.

Out of scope

a. Network-associated EVPNs

b. Router-associated EVPNs

Use Case Summary

This deals with making BGPVPN workflows inside ODL more robust when used with Neutron BGPVPN APIs. To add, when such Neutron BGPVPN APIs are applied through HOT templates, the VPNs have to be realized consistently in both the control-plane datastores and data-plane flows/routes.

RIB - Routing Information Base - Represents the routes held inside the ODL controller Quagga BGP. FIB - Forwarding Information Base - Represents the routes held inside ODL controller.

For all the router-driven BGPVPNs below, please assume that two subnets (one IPv4 and one IPv6) are associated to the router. For all the network-driven BGPVPNs below, please assume that only a single IPv4 subnet is present on the network.

For the template types and their contents, please see CSIT section.

UC 1: A net-driven BGPVPN creation with Network association

The steps for this use-case (single HOT template) template 1 - Create a VPN1 with RD1 Create Network1, Create Subnet1, Boot VMs on the Network1, Associate Network1 into VPN1

template 1 type - TYPE A

Create stack with template 1, make sure that VPN1 appears in the ODL controller. Also make sure all of the VM IPs (and their secondaries) must be realized into VPN1. Delete stack with template 1, make sure VPN1 disappears from the ODL controller.

UC 2: A net-driven BGPVPN cycled through multiple Networks association

The steps for this use-case (multiple HOT templates) template 1 - Create a VPN1 with RD1 template 2 - Create Network1, Create Subnet1, Boot VMs on the Network1, Associate Network1 into VPN1 template 3 - Create Network2, Create Subnet2, Boot VMs on the Network2, Associate Network2 into VPN1

template 1 type - TYPE B template 2 type - TYPE C template 3 type - TYPE C

Create stack with template 1, make sure VPN1 appears in the ODL controller.

Create stack with template 2 and template 3 concurrently. Make sure all of the VM IPs (and their secondaries) on Network1 and Network2 (and their secondaries) must be realized into VPN1.

Delete stack with template 2, template 3 concurrently. Make sure all of the VM IPs (and their secondaries) on Network1 and Network2 must be removed from VPN1.

Delete stack with template 1, VPN1 must disappear from the ODL controller.

UC 2.3: A net-driven BGPVPN cycled through concurrent Networks association and disassociation (no overlapping cidrs on networks)

The steps for this use-case (multiple HOT templates) template 1 - Create a VPN1 with RD1 template 2 - Create Network1, Create Subnet1, Boot VMs on the Network1, Associate Network1 into VPN1 template 3 - Create Network2, Create Subnet2, Boot VMs on the Network2, Associate Network2 into VPN1

template 1 type - TYPE B template 2 type - TYPE C template 3 type - TYPE C

Create stack with template 1, make sure VPN1 appears in the ODL controller. Create stack with template 2, all of the VM IPs (and their secondaries) on Network1 must be realized into VPN1.

Delete stack with template 2 and concurrently create stack with template 3. All of the VM IPs (and their secondaries) on Network1 must be removed from VPN1. All of the VM IPs (and their secondaries) on Network2 must be realized into VPN1.

Delete stack with template 3, make sure that Network2 must be removed from VPN1. Delete stack with template 1, VPN1 must disappear from the ODL controller.

UC 2.4: A net-driven BGPVPN cycled through concurrent Networks association and disassociation (overlapping cidrs on networks)

The steps for this use-case (multiple HOT templates) template 1 - Create a VPN1 with RD1 template 2 - Create Network1, Create Subnet1, Boot VMs on the Network1, Associate Network1 into VPN1 template 3 - Create Network2, Create Subnet2 (same CIDR as Subnet1), Boot VMs on the Network2, Associate Network2 into VPN1

template 1 type - TYPE B template 2 type - TYPE C template 3 type - TYPE C

Create stack with template 1, make sure VPN1 appears in the ODL controller. Create stack with template 2, all of the VM IPs (and their secondaries) on Network1 must be realized into VPN1.

Delete stack with template 2 All of the VM IPs (and their secondaries) on Network1 must be removed from VPN1.

Create stack with template 3 All of the VM IPs (and their secondaries) on Network2 must be realized into VPN1.

Delete stack with template 3, make sure that Network2 must be removed from VPN1. Delete stack with template 1, VPN1 must disappear from the ODL controller.

UC 2.5: Two net-driven BGPVPNs cycled through with same RouteDistinguisher and same Network

template 1 - Create Network1, Create Subnet1, Boot VMs on the Network1 template 2 - Create a VPN1 with RD1 and associate Network1 to VPN1 template 3 - Create a VPN2 with RD1 and associate Network1 to VPN2

template 1 type - TYPE D template 2 type - TYPE E template 3 type - TYPE E

Create stack with template 1, make sure all the VMs appears in the ODL controller. Create stack with template 2, all of the VM IPs (and their secondaries) on Network1 must be realized into VPN1.

Delete stack with template 2 and concurrently create stack with template 3. All of the VM IPs (and their secondaries) on Network1 must be removed from VPN1. All of the VM IPs (and their secondaries) on Network1 must be realized into VPN2.

Delete stack with template 3, make sure that Network1 must be removed from VPN2. Delete stack with template 1, make sure all the VMs disappear from the ODL controller.

UC 2.6: A net-driven BGPVPN ordering VPN deletion before network deletion

template 1 - Create a VPN1 with RD1 template 2 - Create Network1, Create Subnet1, Boot VMs on the Network1, Associate Network1 into VPN1

template 1 type - TYPE B template 2 type - TYPE C

Create stack with template 1, make sure VPN1 appears in the ODL controller. Create stack with template 2, all of the VM IPs (and their secondaries) on Network1 must be realized into VPN1.

Delete stack with template 1 All of the VM IPs (and their secondaries) on Network1 must be removed from VPN1. Make sure ELAN traffic continues to work.

Delete stack with template 2 Make sure all the VMs, Networks disappear from the ODL controller.

UC 2.7: A net-driven BGPVPN ordering network deletion before VPN deletion without disassociating the network

The steps for this use-case (multiple HOT templates) template 1 - Create a VPN1 with RD1 template 2 - Create Network1, Create IPv4 Subnet1, Boot VMs on the Network1 template 3 - Associate Network1 into VPN1

template 1 type - TYPE B template 2 type - TYPE D template 3 type - TYPE F

Create stack with template 1, make sure VPN1 appears in the ODL controller. Create stack with template 2, ensure all of the VM IPs (and their secondaries) on Router1 must be realized into VPN1. Create stack with template 3, and ensure all of the VM IPs (and their secondaries) on Router1 must be realized into VPN1.

Delete stack with template 2, make sure that Network1 must be removed from VPN1. Also make sure all of the VMs (and their secondaries) along with Network1 are not hanging around at all.

Delete stack with template 1, VPN1 must disappear from the ODL controller. Delete stack with template 3, the stack deletion must fail and this is normal.

UC 2.8: A router-driven BGPVPN creation with SingleStack Router association

The steps for this use-case (single HOT template) template 1 - Create a VPN1 with RD1. Create Network1, Create Subnet1, Boot VMs on the Network1, Create Router1, Add Subnet1 to Router1 and Associate Router1 into VPN1

template 1 type - TYPE M

Create stack with template 1, make sure that VPN1 appears in the ODL controller. Also make sure all of the VM IPs (and their secondaries) on the Router1 must be realized into VPN1. Delete stack with template 1, make sure VPN1 disappears from the ODL controller along with removal of VM IPs from the VPN1.

UC 2.9: A router-driven BGPVPN creation with DualStack Router association

The steps for this use-case (single HOT template) template 1 - Create a VPN1 with RD1. Create Network1, Create IPv4 Subnet11, Create IPv6 Subnet12 Create Network2, Create IPv4 Subnet21, Create IPv6 Subnet22, Boot VMs on the Network1 Boot VMs on the Network2, Create Router1, Add Subnet11 to Router1, Add Subnet12 to Router1, Add Subnet21 to Router1, Add Subnet22 to Router1 and Associate Router1 into VPN1

template 1 type - TYPE N

Create stack with template 1, make sure that VPN1 appears in the ODL controller. Also make sure all of the VM IPv4 and IPv6s (and their secondaries) on the Router1 must be realized into VPN1. Delete stack with template 1, make sure VPN1 disappears from the ODL controller.

UC 2.10: A router-driven BGPVPN cycled through with association and disassociation

The steps for this use-case (multiple HOT templates) template 1 - Create a VPN1 with RD1 template 2 - Create Network1, Create IPv4 Subnet1, Create IPv6 Subnet2, Boot VMs on the Network1 Create Router1, Add Subnet1 to Router1, Add Subnet2 to Router1 template 3 - Associate Router1 into VPN1

template 1 type - TYPE B template 2 type - TYPE O template 3 type - TYPE Q

Create stack with template 1, make sure VPN1 appears in the ODL controller. Create stack with template 2, all of the VM IPs (and their secondaries) on Network1 must be realized into Router1.

Create stack with template 3, and ensure all of the VM IPs (and their secondaries) on Router1 must be realized into VPN1.

Delete stack with template 3, make sure that Router1 must be removed from VPN1. Immediately create stack with template 3, make sure that Router1 is included back into VPN1. Also make sure all of the VM IPv4 and IPv6s (and their secondaries) on the Router1 must be realized into VPN1.

Delete stack with template 3, Router1 must disappear from VPN1. Delete stack with template 2, Router1 must disappear from the ODL controller. Delete stack with template 1, VPN1 must disappear from the ODL controller.

UC 2.10 variation 2 - A router-driven BGPVPN cycled through with association and disassociation

The steps for this use-case (multiple HOT templates) template 1 - Create a VPN1 with RD1. template 2 - Create Network1, Create IPv4 Subnet1, Create IPv6 Subnet2, Boot VMs on the Network1. Create Router1, Add Subnet1 to Router1, Add Subnet2 to Router1 template 3 - Associate Router1 into VPN1

template 1 type - TYPE B template 2 type - TYPE O template 3 type - TYPE Q

Create stack with template 1, make sure VPN1 appears in the ODL controller. Create stack with template 2, all of the VM IPs (and their secondaries) on Network1 must be realized into Router1.

Create stack with template 3, and ensure all of the VM IPs (and their secondaries) on Router1 must be realized into VPN1.

Delete stack with template 3, make sure that Router1 must be removed from VPN1. Immediately delete stack with template 2, make sure that Router1 disappears from ODL controller.

Delete stack with template 1, VPN1 must disappear from the ODL controller.

UC 2.11: A router-driven BGPVPN ordering router deletion before VPN deletion

The steps for this use-case (multiple HOT templates) template 1 - Create a VPN1 with RD1 template 2 - Create Network1, Create IPv4 Subnet1, Create IPv6 Subnet2, Boot VMs on the Network1 Create Router1, Add Subnet1 to Router1, Add Subnet2 to Router1, Associate Router1 into VPN1

template 1 type - TYPE B template 2 type - TYPE P

Create stack with template 1, make sure VPN1 appears in the ODL controller. Create stack with template 2, ensure all of the VM IPs (and their secondaries) on Router1 must be realized into VPN1.

Delete stack with template 2, make sure that Router2 must be removed from VPN1. Also make sure all of the VM IPv4 and IPv6s (and their secondaries) along with Router1 are not hanging around at all.

Delete stack with template 1, VPN1 must disappear from the ODL controller.

UC 2.11: variation 2 A router-driven BGPVPN ordering router deletion

before VPN deletion without disassociating the router

The steps for this use-case (multiple HOT templates) template 1 - Create a VPN1 with RD1 template 2 - Create Network1, Create IPv4 Subnet1, Create IPv6 Subnet2, Boot VMs on the Network1 Create Router1, Add Subnet1 to Router1, Add Subnet2 to Router1 template 3 - Associate Router1 into VPN1

template 1 type - TYPE B template 2 type - TYPE O template 3 type - TYPE Q

Create stack with template 1, make sure VPN1 appears in the ODL controller. Create stack with template 2, ensure all of the VM IPs (and their secondaries) on Router1 must be realized into VPN1. Create stack with template 3, and ensure all of the VM IPs (and their secondaries) on Router1 must be realized into VPN1.

Delete stack with template 2, make sure that Router2 must be removed from VPN1. Also make sure all of the VM IPv4 and IPv6s (and their secondaries) along with Router1 are not hanging around at all.

Delete stack with template 1, VPN1 must disappear from the ODL controller. Delete stack with template 3, the stack deletion must fail and this is normal.

UC 2.12: A router-driven BGPVPN ordering VPN deletion before router deletion

template 1 - Create a VPN1 with RD1 template 2 - Create Network1, Create IPv4 Subnet1, Create IPv6 Subnet2, Boot VMs on the Network1 Create Router1, Add Subnet1 to Router1, Add Subnet2 to Router1, Associate Router1 to VPN1

template 1 type - TYPE B template 2 type - TYPE P

Create stack with template 1, make sure VPN1 appears in the ODL controller. Create stack with template 2, all of the VM IPs (and their secondaries) on Network1 must be realized into VPN1.

Delete stack with template 1 All of the VM IPs (and their secondaries) on Router1 must be removed from VPN1. Make sure ELAN traffic and Router1 traffic continues to work.

Delete stack with template 2 Make sure all the VMs, Routers disappear from the ODL controller.

UC 2.13: Two router-driven BGPVPNs cycled through with same RouteDistinguisher and same Router

template 1 - Create Network1, Create IPv4 Subnet1, Create IPv6 Subnet2, Boot VMs on the Network1 Create Router1, Add Subnet1 to Router1, Add Subnet2 to Router1 template 2 - Create a VPN1 with RD1 and Associate Router1 into VPN1 template 3 - Create a VPN2 with RD1 and Associate Router1 into VPN2

template 1 type - TYPE O template 2 type - TYPE R template 3 type - TYPE R

Create stack with template 1, make sure all the VMs appears in the ODL controller. Create stack with template 2, all of the VM IPs (and their secondaries) on Router1 must be realized into VPN1.

Delete stack with template 2. Immediately create a stack with template 3. All of the VM IPs (and their secondaries) on Router1 must be removed from VPN1. All of the VM IPs (and their secondaries) on Router1 must be realized into VPN2.

Delete stack with template 3, make sure that Router1 must be removed from VPN2. Delete stack with template 1, make sure all the VMs disappear from the ODL controller.

Proposed change

In order to tighten the L3VPN service to make it consistently realize the Openstack Neutron BGPVPN orchestrated VPNs (and the networks/routers in them) in OpenDaylight controller, we need to address gaps in various components of NetVirt.

The following components will be enhanced to ensure a consistent and scaled-out realization of BGPVPNs onto OpenDaylight NetVirt: a. NeutronVPN (NeutronVpnManager) b. VPN Engine (VpnInstanceListener and VpnInterfaceManager) c. BGP Engine (BGPManager) d. VRF Engine (FIBManager) e. NAT Engine (NATService) f. EVPN Engine (EVPN in ELANManager)

All of the above engines enhancement is to enable safe parallel processing of multiple vpn-instances regardless of whether such vpninstances use the same route-distinguisher, same ip-address, same interfaces etc.

Module Changes

VPN Engine:

Currently Oper/VpnInstanceOpData is keyed by vrf-id (aka Route Distinguisher) and this kind of keying was all OK as we serviced only one VPNInstance at a time that would carry this vrf-id.

As we move to public cloud, the customers will start to use the neutron bgpvpn api (in lieu of ODL provided RESTful APIs). With the Neutron BGPVPN API interface, it is very possible to have multiple VPNInstance could be managed by tenants where in such instances carry the same vrf-id (aka Route Distinguisher). This is because route-distinguisher is not how an VPN is identified in Neutron BGPVPN, instead the vpn-uuid allocated by Neutron uniquely identifies a VPN.

However, L3VPN Service in ODL is using the route-distinguisher as the key for realizing a vpn (i.e, VpnInstanceOpData) and so the key for this yang will be changed to use the vpn-instance-uuid instead of route-distinguisher.

This specific VpnInstanceOpData datastore is referred by many services within ODL and so such services must be updated on their consumption/production of information into this datastore.

One more new model that maps a given RD (route-distinguisher) to an Active VpnInstance will be required for use by the BGPEngine.

VRF Engine:

The VRFEngine does not use vpn-instance-uuid at all and uses only vrfTables as the primary datastore for representing the FIB. This vrfTables has vrf-id (route-distinguisher) as the key, to maintain the FIB.

This engine will be enhanced such that vrfTables become a child to a vpn-instance-uuid keyed container thereby enabling multiple vpninstances to be managed regardless of whether such vpn-instances use the same RD or otherwise.

So the VrfEntry will be keyed by (VPNInstance + RD + Prefix) instead of being keyed only with (RD + prefix) as it stands today.

All BGP Advertisements and Withdrawals (for all routes managed by ODL itself - i.e., non-BGP-imported routes), will be done within the VRFEngine (i.e, VrfEntryListener) rather than by the VPN Engine. All such advertisements will be serialized within the VrfEntryListener so that route consistency is maintained between ODL and its BGP neighbor.

BGP Engine:

The BGP Engine within ODL is primarily responsible for: a. Pushing a route from its external neighbor into ODL b. Removing a route from its external neighbor from ODL c. Advertising an ODL managed route to its external neighbor d. Withdrawing an ODL managed route to its external neighbor

Since there can be more than one VpnInstance that can use a given route-distinguisher at a time within Opendaylight (a situation where previous vpn-instance is being deleted while a new vpn-instance with same RD being created), BGP Engine needs to know which of the vpn-instances out of the these to choose to import a route when it comes from an external neighbor (case a).

This is because, BGP Engine (and Quagga which has real BGP logic) within ODL operates only with route-distinguishers and not with vpn-instances. This is the same case with the other BGP neighbors too that talk to ODL BGP Engine.

As a result the BGP Engine will continue to work only with route-distinguishers but it will make a call to the VpnEngine to figure out to which all vpn-instances an incoming route must be written to (case a). The same principle applies when an external bgp route is to be removed from within OpenDaylight (case b).

Advertising (or) Withdrawing an ODL managed route will be driven by the VRFEngine towards the BGPEngine i.e, case c and case d. The BGPEngine will have no role to play here.

NeutronVPN Engine:

There are race conditions within the NeutronVPN that gets triggered when HOT templates are executed primarily in the Neutron BGPVPN management path. These race conditions are around the Neutron Router, Neutron BGPVPN, Neutron Subnet and Neutron Network resources access and their management within NeutronVPN. So this engine will be enhanced to close out these race-conditions.

Pipeline changes

This initiative does not introduce (or) mandate any pipeline changes.

Yang changes

Changes will be needed in `fib-rpc.yang` , `odl-fib.yang` to address this feature.

Changes will be needed in `odl-l3vpn.yang` and `odl-fib.yang` to support the robustness for Neutron BGPVPN API.

FIB-RPC YANG CHANGES

Listing 9: fib-rpc.yang

```
rpc populate-fib-on-dpn {
    description "Populates FIB table in specified DPN";
    input {
        leaf dpid {
            type uint64;
        }
        leaf vpn-id {
            type uint32;
        }
+       leaf vpn-instance-name {
+           type string;
+       }
        leaf rd {
            type string;
        }
    }
}

rpc cleanup-dpn-for-vpn {
    description "Removes the VPN Fib entries in a given DPN";
    input {
        leaf dpid {
            type uint64;
        }
        leaf vpn-id {
            type uint32;
        }
+       leaf vpn-instance-name {
+           type string;
+       }
        leaf rd {
            type string;
        }

    }
}
```

ODL-FIB YANG CHANGES

Listing 10: odl-fib.yang

```
--- a/fibmanager/api/src/main/yang/odl-fib.yang
+++ b/fibmanager/api/src/main/yang/odl-fib.yang
@@ -100,15 +100,19 @@ module odl-fib {

 container fibEntries {
     config true;
-        list vrfTables{
-            key "routeDistinguisher";
-            leaf routeDistinguisher {type string;}
-            uses vrfEntries;
-            uses macVrfEntries;
-        }
+        list vpnInstanceNames {
+            key vpnInstanceName;
+            leaf vpnInstanceName { type string; }
+            list vrfTables{
+                key "routeDistinguisher";
+                leaf routeDistinguisher {type string;}
+                uses vrfEntries;
+                uses macVrfEntries;
+            }

-        container ipv4Table{
-            uses ipv4Entries;
+            container ipv4Table{
+                uses ipv4Entries;
+            }
     }
 }
```

ODL-L3VPN YANG CHANGES

Listing 11: odl-l3vpn.yang

```
--- a/vpnmanager/api/src/main/yang/odl-l3vpn.yang
+++ b/vpnmanager/api/src/main/yang/odl-l3vpn.yang
@@ -184,10 +184,13 @@ module odl-l3vpn {
             }
     }

-    container evpn-rd-to-networks {
+    container evpn-to-networks {
         description "Holds the networks to which given evpn is attached";
-        list evpn-rd-to-network {
+        list evpn-to-network {
-           key rd;
+           key vpn-instance-name;
+           leaf vpn-instance-name {
+             type string;
+           }
            leaf rd {
```

(continues on next page)

(continued from previous page)

```
              type string;
            }
@@ -261,7 +264,7 @@ module odl-l3vpn {
     container vpn-instance-op-data {
         config false;
         list vpn-instance-op-data-entry {
-           key vrf-id;
+           key vpn-instance-name;
            leaf vpn-id { type uint32;}
            leaf vrf-id {
              description

@@ +706,7 +717,7 @@ module odl-l3vpn {
+ container rd-to-vpns {
+    config false;
+    list rd-to-vpn {
+        key "rd";
+        leaf rd { type string; }
+        list vpn-instances {
+            key "vpn-instance-name";
+            leaf vpn-instance-name { type string; }
+            leaf active { type boolean; }
+        }
+    }
+}
```

Solution considerations

Configuration impact

Clustering considerations

The feature should operate in ODL Clustered (3-node cluster) environment reliably.

Other Infra considerations

N.A.

Security considerations

N.A.

Scale and Performance Impact

Not covered by this Design Document.

Targeted Release

Fluorine.

Alternatives

There were no alternative proposals considered for this initiative.

Usage

The feature can be excited by Openstack Neutron BGPVPN REST APIs (or) by Openstack Neutron BGPVPN CLIs.

As part of this spec implementation, we will actually be exciting Neutron BGPVPN APIs via the Heat Templates. Please see more details of the templates and the types of templates we will use in the CSIT section.

Features to Install

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

REST API

No REST APIs are introduced in this feature.

Implementation

Assignee(s)

Primary assignee:

Vivekanandan Narasimhan (n.vivekanandan@ericsson.com)

Work Items

The Trello cards have already been raised for this feature under the neutron-bgpvpn-api-support.

#Here is the link for the Trello Card: #https://trello.com/c/Tfpr3ezF/33-evpn-evpn-rt5

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

A New CSIT suite will be added for this feature, as this starts to use Neutron BGPVPN APIs in couple with HOT templates and makes it official for ODL platform.

CSIT will use the HOT Templates referred here.

TYPE A

Listing 12: TYPE A

```
description: BGPVPN networking example TYPE A (admin)
heat_template_version: '2013-05-23'

resources:

    BGPVPN1:
        type: OS::Neutron::BGPVPN
        properties:
            import_targets: [ "100:1001"]
            export_targets: [ "100:1002"]
            route_targets: [ "100:1000" ]
            name: "default VPN"

    Net1:
        type: OS::Neutron::Net
        properties:
            name: "default Net1"

    SubNet1:
        type: OS::Neutron::Subnet
        properties:
            network: { get_resource: Net1 }
            cidr: 20.1.1.0/24

    BGPVPN_NET_assoc1:
        type: OS::Neutron::BGPVPN-NET-ASSOCIATION
        properties:
            bgpvpn_id: { get_resource: BGPVPN1 }
            network_id: { get_resource: Net1 }
```

TYPE B

Listing 13: TYPE B

```
description: BGPVPN networking example TYPE B (admin)
heat_template_version: '2013-05-23'

resources:

    BGPVPN1:
        type: OS::Neutron::BGPVPN
        properties:
            import_targets: [ "100:1001"]
            export_targets: [ "100:1002"]
            route_targets: [ "100:1000" ]
            name: "default VPN"
```

TYPE C

Listing 14: TYPE C

```
description: BGPVPN networking example TYPE C (admin)
heat_template_version: '2013-05-23'

resources:

    Net1:
    type: OS::Neutron::Net
    properties:
        name: "default Net1"

    SubNet1:
        type: OS::Neutron::Subnet
        properties:
            network: { get_resource: Net1 }
            cidr: 20.1.1.0/24

    BGPVPN_NET_assoc1:
        type: OS::Neutron::BGPVPN-NET-ASSOCIATION
        properties:
            bgpvpn_id: "default VPN"
            network_id: { get_resource: Net1 }
```

TYPE D

Listing 15: TYPE D

```
description: BGPVPN networking example TYPE D (admin)
heat_template_version: '2013-05-23'

resources:

    Net1:
```

(continues on next page)

(continued from previous page)

```
        type: OS::Neutron::Net
        properties:
            name: "default Net1"

    SubNet1:
    type: OS::Neutron::Subnet
    properties:
        network: { get_resource: Net1 }
        cidr: 20.1.1.0/24
```

TYPE E

Listing 16: TYPE E

```
description: BGPVPN networking example (admin)
heat_template_version: '2013-05-23'

resources:

BGPVPN1:
    type: OS::Neutron::BGPVPN
    properties:
        import_targets: [ "100:1001"]
        export_targets: [ "100:1002"]
        route_targets: [ "100:1000" ]
        name: "default VPN"

BGPVPN_NET_assoc1:
    type: OS::Neutron::BGPVPN-NET-ASSOCIATION
    properties:
        bgpvpn_id: "default VPN"
        network_id: "default Net1"
```

TYPE F

Listing 17: TYPE F

```
description: BGPVPN networking example (admin)
heat_template_version: '2013-05-23'

resources:

    BGPVPN_NET_assoc1:
        type: OS::Neutron::BGPVPN-NET-ASSOCIATION
        properties:
            bgpvpn_id: "default VPN"
            network_id: "default Net1"
```

TYPE M

Listing 18: TYPE M

```
description: BGPVPN networking example (admin)
heat_template_version: '2013-05-23'

resources:

    BGPVPN1:
        type: OS::Neutron::BGPVPN
        properties:
            import_targets: [ "100:1001"]
            export_targets: [ "100:1002"]
            route_targets: [ "100:1000" ]
            name: "default VPN"


    Net1:
        type: OS::Neutron::Net
        properties:
            name: "default Net1"

 SubNet1:
     type: OS::Neutron::Subnet
     properties:
        network: { get_resource: Net1 }
        cidr: 20.1.10.0/24

 Router:
     type: OS::Neutron::Router
     properties:
         name: "default Router"


 router_interface:
     type: OS::Neutron::RouterInterface
     properties:
        router_id: { get_resource: Router }
        subnet_id: { get_resource: SubNet1 }

 BGPVPN_router_assoc1:
```

(continues on next page)

(continued from previous page)

```
    type: OS::Neutron::BGPVPN-ROUTER-ASSOCIATION
    properties:
        bgpvpn_id: { get_resource: BGPVPN1 }
        router_id: { get_resource: Router }
```

TYPE N

Listing 19: TYPE M

```
description: BGPVPN networking example (admin)
heat_template_version: '2013-05-23'

resources:

    BGPVPN1:
    type: OS::Neutron::BGPVPN
    properties:
        import_targets: [ "100:1001"]
        export_targets: [ "100:1002"]
        route_targets: [ "100:1000" ]
        name: "default VPN"

    Net1:
        type: OS::Neutron::Net
        properties:
            name: "default Net1"

    SubNet11:
        type: OS::Neutron::Subnet
        properties:
            network: { get_resource: Net1 }
            cidr: 20.1.1.0/24

    SubNet12:
        type: OS::Neutron::Subnet
        properties:
            network: { get_resource: Net1 }
            cidr: 2001:db8:cafe:1e::/64

    Net2:
        type: OS::Neutron::Net
        properties:
            name: "default Net2"

    SubNet21:
        type: OS::Neutron::Subnet
        properties:
            network: { get_resource: Net2 }
            cidr: 30.1.1.0/24

    SubNet22:
        type: OS::Neutron::Subnet
        properties:
            network: { get_resource: Net2 }
```

(continues on next page)

(continued from previous page)

```
            cidr: 2002:db8:cafe:1e::/64

    Router:
        type: OS::Neutron::Router
        properties:
            name: "default Router"

    router_interface1:
        type: OS::Neutron::RouterInterface
        properties:
            router_id: { get_resource: Router }
            subnet_id: { get_resource: SubNet11 }

    router_interface2:
        type: OS::Neutron::RouterInterface
        properties:
            router_id: { get_resource: Router }
            subnet_id: { get_resource: SubNet12 }

    router_interface3:
        type: OS::Neutron::RouterInterface
        properties:
            router_id: { get_resource: Router }
            subnet_id: { get_resource: SubNet21 }

    router_interface4:
        type: OS::Neutron::RouterInterface
        properties:
            router_id: { get_resource: Router }
            subnet_id: { get_resource: SubNet22 }

    BGPVPN_router_assoc1:
        type: OS::Neutron::BGPVPN-ROUTER-ASSOCIATION
        properties:
             bgpvpn_id: { get_resource: BGPVPN1 }
             router_id: { get_resource: Router }
```

TYPE O

Listing 20: TYPE 0

```
description: BGPVPN networking example (admin)
heat_template_version: '2013-05-23'

resources:

    BGPVPN1:
        type: OS::Neutron::BGPVPN
        properties:
            import_targets: [ "100:1001"]
            export_targets: [ "100:1002"]
            route_targets: [ "100:1000" ]
            name: "default VPN"
```

(continues on next page)

(continued from previous page)

```
    Net1:
        type: OS::Neutron::Net
        properties:
            name: "default Net1"

    SubNet11:
        type: OS::Neutron::Subnet
        properties:
            network: { get_resource: Net1 }
            cidr: 20.1.1.0/24

    SubNet12:
        type: OS::Neutron::Subnet
        properties:
           network: { get_resource: Net1 }
           cidr: 2001:db8:cafe:1e::/64

    Router:
        type: OS::Neutron::Router
        properties:
            name: "default Router"

    router_interface1:
        type: OS::Neutron::RouterInterface
        properties:
            router_id: { get_resource: Router }
            subnet_id: { get_resource: SubNet11 }

    router_interface2:
        type: OS::Neutron::RouterInterface
        properties:
            router_id: { get_resource: Router }
           subnet_id: { get_resource: SubNet12 }
```

TYPE P

Listing 21: TYPE P

```
description: BGPVPN networking example TYPE P (tenant)
heat_template_version: '2013-05-23'

resources:

    BGPVPN1:
    type: OS::Neutron::BGPVPN
    properties:
        import_targets: [ "100:1001"]
        export_targets: [ "100:1002"]
        route_targets: [ "100:1000" ]
        name: "default VPN"

    Net1:
        type: OS::Neutron::Net
        properties:
```

(continues on next page)

(continued from previous page)

```
            name: "default Net1"

    SubNet11:
        type: OS::Neutron::Subnet
        properties:
            network: { get_resource: Net1 }
            cidr: 20.1.1.0/24

SubNet12:
    type: OS::Neutron::Subnet
    properties:
       network: { get_resource: Net1 }
       cidr: 2001:db8:cafe:1e::/64

Router:
    type: OS::Neutron::Router
    properties:
        name: "default Router"

router_interface1:
    type: OS::Neutron::RouterInterface
    properties:
       router_id: { get_resource: Router }
       subnet_id: { get_resource: SubNet11 }

router_interface2:
    type: OS::Neutron::RouterInterface
    properties:
       router_id: { get_resource: Router }
       subnet_id: { get_resource: SubNet12 }

BGPVPN_router_assoc1:
    type: OS::Neutron::BGPVPN-ROUTER-ASSOCIATION
    properties:
        bgpvpn_id: { get_resource: BGPVPN1 }
        router_id: { get_resource: Router }
```

TYPE Q

Listing 22: TYPE Q

```
description: BGPVPN networking example TYPE P (tenant)
heat_template_version: '2013-05-23'

resources:

BGPVPN_router_assoc1:
    type: OS::Neutron::BGPVPN-ROUTER-ASSOCIATION
    properties:
        bgpvpn_id: "default VPN"
        router_id: "default Router"
```

TYPE R

Listing 23: TYPE R

```
description: BGPVPN networking example TYPE P (tenant)
heat_template_version: '2013-05-23'

resources:

    BGPVPN1:
        type: OS::Neutron::BGPVPN
        properties:
            import_targets: [ "100:1001"]
            export_targets: [ "100:1002"]
            route_targets: [ "100:1000" ]
            name: "default VPN"

    BGPVPN_router_assoc1:
        type: OS::Neutron::BGPVPN-ROUTER-ASSOCIATION
        properties:
            bgpvpn_id: "default VPN"
            router_id: "default Router"
```

Documentation Impact

References

[1] https://docs.openstack.org/networking-bgpvpn/latest/user/heat.html#examples

Table of Contents

Tap-as-a-Service

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

Tap-as-a-Service provides port mirroring for tenant virtual networks.

Problem description

Debugging a complex virtual network can be a difficult task for a cloud administrator. The administrator does not have any visibility into the network. In order to analyze the network behavior we need to mirror the traffic and do a packet inspection.

Use Cases

TaaS will provide visibility into the network, by monitoring the network traffic generated or received by the VM. Port mirroring involves sending a copy of packets entering or leaving one port to another designated port. TaaS can also provide data for network analytics and security applications.

Port mirroring happens from Tap Flow Port to Tap Service Port. TapFlow represents the port on a VM from which the traffic needs to be mirrored and TapService represents the port on a different VM to which the mirrored traffic is delivered. The first phase of the feature will support the following use cases.

- Use case 1: Local port mirroring where the Tap Service Port(TSP) and Tap Flow Port(TFP) are on the same switch. Both the TFP and TSP are untagged Neutron Ports.
- Use case 2: Local Port mirroring where TFP is untagged and TSP is tagged sub-port.
- Use case 3: Local Port mirroring where TSP and TFP are both transparent.
- Use case 4: Local Port mirroring where TFP is untagged and TSP is transparent.
- Use case 5: Local Port mirroring where TFP is tagged sub-port and TSP is transparent. Remote port mirroring where the Tap Service Port and Tap Flow Port are on different switches.
- Use case 6: Remote port mirroring where the Tap Service Port and Tap Flow Port are on different switches. Both the TSP and TFP are untagged Neutron Ports
- Use case 7: Remote Port mirroring where TFP is untagged and TSP is tagged sub-port.
- Use case 8: Remote Port mirroring where TSP and TFP are both transparent.
- Use case 9: Remote Port mirroring where TFP is untagged and TSP is transparent.
- Use case 10: Remote Port mirroring where TFP is tagged sub-port and TSP is transparent.

Proposed change

To realize tap services in OpenDaylight, a new tapservice module will be implemented. This module listens to neutron-tapaas configuration DS and fetches tap service and flow port data. This modules also listens to Interface State operational DS and retrieves the service and flow port related operational information. Based on the data, pipeline to realize the tap service is programmed. The tap service details are persisted so that if the VM that hosts the flow or service port migrates, the old flows can be detected and removed.

Pipeline changes

This features introduce the following pipeline:

Two new tables will be added as a part of this feature. OUTBOUND_TAP_CLASSIFIER_TABLE = 170; INBOUND_TAP_CLASSIFIER_TABLE = 171;

Service priority for this feature is as follows :- * For traffic on tap flow port bound to outbound traffic (TaaS API with direction == OUT) , then TAP MUST be highest priority ingress service (w.r.t switch) for that lport INGRESS_TAP_SERVICE_INDEX = 1 * For a tap flow port bound to inbound traffic (TaaS API with direction == IN) , then TAP MUST be lowest priority egress service (w.r.t to switch) for that port EGRESS_TAP_SERVICE_INDEX = 9

A block of IDs from Id Manager will be allocated to get the tunnel Id corresponding to the Tap Service TAP_SERVICE_LOW_ID = 350000L TAP_SERVICE_HIGH_ID = 375000L;

Tap Flow Port Egress Traffic

Mirroring traffic egressing from Tap Flow port will be of highest priority. So the ingress traffic (w.r.t switch) will match the INGRESS_TAP_SERVICE_INDEX at the INGRESS_LPORT_DISPATCHER_TABLE and go to the OUTBOUND_TAP_CLASSIFIER_TABLE where the packet matching the Lport Tag of the Flow Tap port will be * copied and sent based on the Egress Actions for the Tap Service Port either * to EGRESS_LPORT_DISPATCHER_TABLE to the VM port if the Tap Service port is in the same switch as Tap Flow port or * embed the label corresponding to Tap Service Id in the VNI field and sent onto the tunnel. * original packet is ReSubmitted to the INGRESS_LPORT_DISPATCHER_TABLE.

Tap Flow Port Ingress Traffic

Mirroring traffic ingressing into the Tap Flow Port will be of the lowest priority. So the egress traffic (w.r.t switch) will match the EGRESS_TAP_SERVICE_INDEX at the EGRESS_LPORT_DISPATCHER_TABLE and go to Inbound_Tap_CLASSIFIER_TABLE where the packet matching the Lport Tag of the Flow Tap port will be * copied and sent based on the Egress Actions for the Tap Service Port either * to EGRESS_LPORT_DISPATCHER_TABLE to the VM port if the Tap Service port is in the same switch as Tap Flow port or * embed the label corresponding to Tap Service Id in the VNI field and sent onto the tunnel. * ReSubmit the original packet to EGRESS_LPORT_DISPATCHER_TABLE

Tap Service Port Ingress Traffic

If the Tap Service port and Tap Flow port are on different switches then, the copied packet will egress from the tunnel and will match on the tunnel id corresponding to the Tap Service Id in the INTERNAL_TUNNEL_TABLE and go to EGRESS_LPORT_DISPATCHER_TABLE and from there it will be output onto the VM of the Service Tap port.

TABLE	MATCH	ACTION
LPORT_DISPATCHER_TABLE	metadata=service priority && lport-tag	goto OUTBOUND_TAP_CLASSIFIER _TABLE
OUTBOUND_TAP_CLASSIFIER _TABLE	**lport-tag=tap flow** port	**Action 1: GoTo EGRESS_LPORT** _DISPATCHER_TABLE, if same switch ONTO Tunnel port, if different **Action 2:** ReSubmit to LPORT_DISPATCHER _TABLE
EGRESS_LPORT_DISPATCHER _TABLE	Reg6==service Priority && lport-tag	**Go to INBOUND_TAP_CLASSIFIER** _TABLE
INBOUND_TAP_CLASSIFIER | lport-tag=tap flow port _TABLE |		Action 1: Output on the VM Service Port if same switch ONTO Tunnel port, if different Action 2: ReSubmit to EGRESS_LPORT _DISPATCHER_TABLE
INTERNAL_TUNNEL_TABLE	tunnel_id=tap service id	go to EGRESS_LPORT_DISPATCHER TABLE

Tap Service with VLAN Tags

TFP TYPE | TSP TYPE | Packet entering TSP | Pipeline
UnTagged | UnTagged | UnTagged | Normal
UnTagged | Subport -Tag Y | Tagged with Tag Y | Match on Lport Tag of | | subport
Transparent | Transparent | Tag retained | Normal
UnTagged | Transparent | UnTagged | Normal
Subport- Tag X | Transparent | Tagged with Tag X | Lport Tag of the | | Subport

Yang changes

New YANG model will be defined in a new module called "tapservice". This yang is to support the tap service realization in opendaylight.

```
 :caption: tapservice.yang
 grouping tap-port-attributes {
   description
       "Attributes for the service and flow port";
   leaf dpid {
       type uint64;
   }
   leaf port-number {
       type uint32;
   }
   leaf if-index {
       type int32;
   }
}
container tap-services-lookup {
   description "Container to store the list of tap services configured from␣
↪openstack along
   with its service and flow port attributes. This is used to program the␣
↪openflow rules
   on the switches corresponding to tap service and flow ports";

   list tap-services {
       key "tap-service-id";
       leaf tap-service-id {
           type yang:uuid;
           description "UUID of the Tap Service Instance";
       }
       leaf port-id {
           type yang:uuid;
           description "Destination port for traffic";
       }
       uses tap-port-attributes;
       list tap-flows {
           key "tap-flow-id";
           description "List of tap flows associated with the tap Service";
           leaf tap-flow-id {
               type yang:uuid;
               description "Tap flow Instance";
```

(continues on next page)

(continued from previous page)

```
            }
            uses neutron-taas:tap-flow-attributes;
            uses tap-port-attributes;
        }
    }
}
```

Configuration impact

There is no change to any existing configuration.

Clustering considerations

Clustering support is already taken care in the infrastructure. There is no new requirement for this feature.

Other Infra considerations

None.

Security considerations

Tap Service Port should be configured with the Openstack "port_security_enabled" set to "false" to enable tap traffic to ingress it.

Scale and Performance Impact

The performance impact of mirroring on the switches needs to be tested and documented

Targeted Release

Fluorine.

Alternatives

None.

Usage

Features to Install

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

Workflow

Following are the steps to be followed for mirroring a Neutron port.

1. Create a Tap Service Neutron port with "port_security_enabled" set to "false".
2. Launch a VM for receiving mirrored data. Associate the Neutron port in step 1 while creating the VM.
3. Create a Tap Service instance using the TaaS CLI "neutron tap-service-create" and associate with the port created in Step 1. This can also be configured via REST APIs.
4. Create a Tap Flow Port using Neutron Client command for TaaS, "neutron tap-flow-create" and associate with the Tap Service instance create in Step 3 and the target Neutron port whose traffic needs to be mirrored. Mirroring can be done for both incoming and/or outgoing traffic from the target Neutron port.

REST API

Tap Service and Flow port can also be created using the following REST API.

Create TapService

URL: /POST /v2.0/taas/tap_services

Sample JSON data

```
{
  "tap_service": {
      "description": "Test_Tap",
      "name": "Test",
      "port_id": "c9beb5a1-21f5-4225-9eaf-02ddccdd50a9",
      "tenant_id": "97e1586d580745d7b311406697aaf097"
  }
}
```

Create TapFlow

URL: POST /v2.0/taas/tap_flows

Sample JSON data

```
{
 "tap_flow": {
     "description": "Test_flow1",
     "direction": "BOTH",
     "name": "flow1",
     "source_port": "775a58bb-e2c6-4529-a918-2f019169b5b1",
     "tap_service_id": "69bd12b2-0e13-45ec-9045-b674fd9f0468",
```

(continues on next page)

(continued from previous page)

```
    "tenant_id": "97e1586d580745d7b311406697aaf097"
  }
}
```

Delete TapService

URL: DELETE /v2.0/taas/tap_services/{tap_service_uuid}

Delete TapFlow

URL: DELETE /v2.0/taas/tap_flows/{tap_flow_uuid}

CLI

None.

Implementation

Assignee(s)

Primary assignee: <Hema Gopalakrishnan> (hema.gopalkrishnan@ericsson.com)

Work Items

1. Add a new bundle
2. Define a new Yang
3. Add listener to neutron-tapaas configuration DS and do the processing.
4. Add listener to Interface State Operational DS.
5. Support Tap Service add for each of the use case.
6. Support Tap Service delete scenario.
7. Support VM migration
8. Add UTs.
9. Add ITs.
10. Add CSIT.
11. Add Documentation

Dependencies

Taap driver in networking-odl needs to be implemented.

Testing

Unit Tests

Relevant Unit Test cases will be added.

Integration Tests

N/A

CSIT

Following test cases will be added to Netvirt CSIT.

1. Configure Tap Service and Flow ports in the same switch and verify the traffic from the flow ports are mirrored to the tap service port.
2. Configure Tap Service and Flow ports in different switches and verify that traffic flows through tunnel to reach the tap service port.
3. Configure the Tap Service and flow ports with VLAN tags as untagged, tagged and transparent and verify each use case.
4. Configure the Tap Flow port with different mirroring direction and verify the appropriate behavior.

Documentation Impact

This will require changes to User Guide and Developer Guide.

User Guide needs to be updated with information on how to configure Tap Services.

References

[1] Netvirt Florine Release Plan - https://docs.google.com/spreadsheets/d/1bDygyIwNOGFEEFDTQJN2LqoTyfmaxwtka-AlwkPcvz
edit#gid=1799274276

[2] Pipeline Changes - https://git.opendaylight.org/gerrit/#/c/71782/

[3] Netvirt Trello Card

[4] Openstack API Reference - https://github.com/openstack/tap-as-a-service/blob/master/API_REFERENCE.rst

1.1.3 NetVirt Design Specifications

Contents:

Table of Contents

Controller Punt Protection

https://git.opendaylight.org/gerrit/#/q/topic:controller-punt-protection

This feature aims to avoid potential overloading of the controller due to excessive punting of packets done via different flow tables. It will use OVS learn actions to rate limit packets punted to the controller similar to[1]. This spec will cover use cases pertaining to IPv4 forwarding only.

Problem description

The current behavior of modules that punt the packet via their flow tables is to continue punting the packets to the controller until controller processes the packet and installs certain flows that prevent further punting of such packets. During the time it takes the controller to process the packet which might require data store accesses, same packets need not be punted again since it puts unnecessary load on the controller as it does the same processing which it did for the first packet.

Use Cases

1. Subnet Route - Unknown IP packets belonging to particular subnet are punted
2. ARP - ARP requests/responses are punted to controller
3. SNAT - Controller based SNAT requires punting of packets for establishing new nat sessions

Proposed change

Packets punting to the controller can be rate limited by installing a temporary flow with the use of OVS learn actions. The first packet that gets punted will install this temporary flow and subsequent packets matching this flow will not be punted. The hard timeout for the learnt flows will be configurable and the configuration changes will take effect on ODL restart. To disable the rate limiting and installation of learnt flows, the timeout values can be set to 0.

Packets that get punted to the controller can be classified in the following way depending on the openflow actions associated with them.

1. punt the packet to controller and stop pipeline processing. This is generally done for unicast packets eg punts done by subnet route and SNAT.

2. punt a copy of the packet to controller but continue the pipeline processing of the packet through other flow tables. This is usually done for broadcast/multicast packets eg ARP punts.

For rate limiting punts of type (1), a temporary learnt flow with higher priority than the flow that punts the packet to controller will be installed. Subsequent packets will hit the learnt flow and punting will not occur till the learnt flow time outs. This approach will be used for subnet route and SNAT punt use cases.

For punts of type (2), new tables need to be added which will have the temporary learnt flows and the result of the match done by the learnt flows will be used by the next tables in the pipeline to determine whether or not to punt this packet. This approach will be used for ARP punts. Two new tables will be introduced for ARP punts which will rate limit arp requests and responses. GARP packets will not be rate limited. The reason for not rate limiting the GARP packets is because they are used by VNFs to advertise themselves and are sent unsolicited. The rate limiting of these can lead to ODL not knowing the presence of the VNF and in cases like VRRP can lead to traffic drops.

[1] http://docs.opendaylight.org/en/stable-nitrogen/submodules/netvirt/docs/specs/temporary-smac-learning.html

Pipeline changes

1. Subnet route pipeline change

   Existing flow for subnet route punt will be modified to add learn action which will install a higher prirority flow to match on vpn id and destination IP.

   Example flows after change:

```
cookie=0x8000004, duration=2167.659s, table=22, n_packets=0, n_bytes=0,␣
↪priority=42,ip,metadata=0x30d40/0xfffffe,nw_dst=10.1.1.255 actions=drop
cookie=0x8000004, duration=1706.113s, table=22, n_packets=0, n_bytes=0,␣
↪priority=42,ip,metadata=0x30d40/0xfffffe,nw_dst=20.1.1.255 actions=drop
cookie=0x0, duration=4.651s, table=22, n_packets=4, n_bytes=392, hard_timeout=10,␣
↪priority=10,ip,metadata=0x30d40/0xfffffe,nw_dst=10.1.1.6 actions=drop

cookie=0x0, duration=4456.957s, table=22, n_packets=1, n_bytes=98, priority=0,ip␣
↪actions=CONTROLLER:65535,
          learn(table=22,hard_timeout=10,priority=10,eth_type=0x800,NXM_OF_IP_
↪DST[],OXM_OF_METADATA[1..23])
```

2. ARP pipeline changes

   Two new tables(195, 196) will be introduced for ARP processing:

   Table 43 for resubmitting to tables 195, 196 and 48 (similar to [0])

   Table 195 for punting the packet as well as installing learnt flows in table 195 and 196. Table 195 will have flow with higher priority than punt flow and will match on packet's actual ARP SPA, TPA and elan id and set a register value on match.

   Table 196 will do a reverse match on packet's SPA, TPA and elan id. This match coupled with match from table 195 will be used to identify GARP packets.

   Table 48 will be modified, with a new flow, which will use the match results from table 195 and 196. A match in table 195 will indicate that this arp packet is already punted to controller, and only rest of the pipeline processing is required. A match both in table 195 and 196 will identify garp packet and this will be processed as done currently.

   Group for arp request processing will be modified to only contain resubmit to table 81.

   Example of flows after change:

```
group_id=5000,type=all,bucket=actions=resubmit(,81)

cookie=0x822002d, duration=6495.082s, table=43, n_packets=55, n_bytes=2310,␣
↪priority=100,
    arp,arp_op=1 actions=group:5000,resubmit(,195),resubmit(,196),resubmit(,48)
cookie=0x822002e, duration=6495.082s, table=43, n_packets=0, n_bytes=0,␣
↪priority=100,
    arp,arp_op=2 actions=resubmit(,195),resubmit(,196),resubmit(,48)
cookie=0x8220000, duration=6495.533s, table=43, n_packets=8, n_bytes=576,␣
↪priority=0 actions=goto_table:48

cookie=0xdeadbeef, duration=34.360s, table=195, n_packets=9, n_bytes=378, hard_
↪timeout=120, priority=20,
    arp,metadata=0x138b000000/0xffff000000,arp_spa=20.1.1.7,arp_tpa=20.1.1.10␣
↪actions=load:0x1->NXM_NX_REG4[0..7]
cookie=0xdeadbeef, duration=19.142s, table=195, n_packets=9, n_bytes=378, hard_
↪timeout=120, priority=20,
```

(continues on next page)

(continued from previous page)

```
    arp,metadata=0x138b000000/0xffff000000,arp_spa=20.1.1.7,arp_tpa=20.1.1.7␣
↪actions=load:0x1->NXM_NX_REG4[0..7]
cookie=0xdeadbeef, duration=50.609s, table=195, n_packets=2, n_bytes=84,␣
↪priority=10,arp actions=CONTROLLER:65535,
    learn(table=195,hard_timeout=120,priority=20,cookie=0xdeadbeef,eth_type=0x806,
    NXM_OF_ARP_SPA[],NXM_OF_ARP_TPA[],OXM_OF_METADATA[24..39],load:0x1->NXM_NX_
↪REG4[0..7]),
    learn(table=196,hard_timeout=120,priority=20,cookie=0xdeadbeef,eth_type=0x806,
    NXM_OF_ARP_SPA[]=NXM_OF_ARP_TPA[],NXM_OF_ARP_TPA[]=NXM_OF_ARP_SPA[],OXM_OF_
↪METADATA[24..39],load:0x1->NXM_NX_REG4[8..15])

cookie=0xdeadbeef, duration=46.760s, table=196, n_packets=0, n_bytes=0, hard_
↪timeout=120, priority=20,
    arp,metadata=0x138b000000/0xffff000000,arp_spa=20.1.1.10,arp_tpa=20.1.1.7␣
↪actions=load:0x1->NXM_NX_REG4[8..15]
cookie=0xdeadbeef, duration=31.542s, table=196, n_packets=9, n_bytes=378, hard_
↪timeout=120, priority=20,
    arp,metadata=0x138b000000/0xffff000000,arp_spa=20.1.1.7,arp_tpa=20.1.1.7␣
↪actions=load:0x1->NXM_NX_REG4[8..15]

cookie=0x8500000, duration=6518.105s, table=48, n_packets=27, n_bytes=1134,␣
↪priority=100,
    arp,reg4=0x1/0xffff actions=load:0->NXM_NX_REG4[],resubmit(,49),resubmit(,50)
cookie=0x8500000, duration=6518.105s, table=48, n_packets=18, n_bytes=756,␣
↪priority=100,
    arp,reg4=0x101/0xffff actions=load:0->NXM_NX_REG4[],CONTROLLER:65535,
↪resubmit(,49),resubmit(,50)
cookie=0x8500000, duration=6519.105s, table=48, n_packets=18, n_bytes=996,␣
↪priority=0 actions=resubmit(,49),resubmit(,50)
```

3. SNAT pipeline change

   Similar to subnet route punt, existing flow for controller based SNAT will be modified with learn action which will put a higher priority flow to match on packet's src ip, dst ip, protocol, layer 4 src port and layer 4 dst port along with vpn id.

   Example flows after change:

```
cookie=0x0, duration=95.890s, table=46, n_packets=0, n_bytes=0, priority=5,tcp,
↪metadata=0x30d40/0xfffffe
            actions=CONTROLLER:65535,learn(table=46,priority=7,eth_type=0x800,nw_
↪proto=6,hard_timeout=5,
            NXM_OF_IP_SRC[],NXM_OF_IP_DST[], NXM_OF_TCP_DST[],NXM_OF_TCP_SRC[],
↪OXM_OF_METADATA[1..23]),
            write_metadata:0x30d40/0xfffffe

cookie=0x0, duration=17.385s, table=46, n_packets=0, n_bytes=0, priority=5,udp,
↪metadata=0x30d40/0xfffffe
            actions=CONTROLLER:65535,learn(table=46,priority=7,eth_type=0x800,nw_
↪proto=17,hard_timeout=5,
            NXM_OF_IP_SRC[],NXM_OF_IP_DST[],NXM_OF_UDP_DST[],NXM_OF_UDP_SRC[],OXM_
↪OF_METADATA[1..23]),
            write_metadata:0x30d40/0xfffffe
```

Yang changes

To support the configuration of timeouts specific to each punt, following yang changes will be done

vpn-config yang changes

`vpnmanager-config:vpn-config` container will be enhanced with two configuration variables to reflect the hard timeout values in learnt flows for arp and subnet route punts.

Listing 24: vpnmanager-config.yang

```
container vpn-config {
     config true;
     leaf arp-cache-size {
         description "arp cache size";
         type uint64;
         default 10000;
     }
     ...

     leaf subnet-route-punt-timeout {
         description "hard timeout value for learnt flows for subnet route punts␣
↪(unit - seconds).
             To turn off the rate limiting and installation of learnt flows, it␣
↪should be set to 0";
         type uint32;
         default 10;
     }
 }
```

elanmanger-config yang changes

`elan-config:elan-config` container will be modified wtih the configuration variable for hard timeout values for ARP learnt flows

Listing 25: elanmanager-config.yang

```
container elan-config {
    config true;
    leaf auto-create-bridge {
        description "If true, auto-create default bridge";
        type boolean;
        default true;
    }
    ...

    leaf arp-punt-timeout {
         description "hard timeout value for learnt flows for arp punts (unit -␣
↪seconds).
             To turn off the rate limiting and installation of learnt flows, it␣
↪should be set to 0";
         type uint32;
         default 5;
```

(continues on next page)

(continued from previous page)

```
    }
}
```

natservice-config yang changes

`natservice-config:natservice-config` container will be enhanced with configuration variable to reflect the hard timeout values in learnt flows for SNAT punts.

Listing 26: natservice-config.yang

```
container natservice-config {
     config true;
     leaf nat-mode {
         type enumeration {
             enum "controller";
             enum "conntrack";
         }
         default "controller";
     }
     leaf snat-punt-timeout {
         description "hard timeout value for learnt flows for snat punts in seconds.
             To turn off the rate limiting and installation of learnt flows, it␣
↪should be set to 0,";
         type uint32;
         default 5;
     }
}
```

Configuration impact

Following configuration files will be modified to provide the default values to the configuration parameters.

netvirt-vpnmanager-config.xml

```
<vpnmanager-config xmlns="urn:opendaylight:netvirt:vpn:config">
   <arp-cache-size>10000</arp-cache-size>
   <arp-learn-timeout>2000</arp-learn-timeout>
   <subnet-route-punt-timeout>10</subnet-route-punt-timeout>
</vpnmanager-config>
```

netvirt-elanmanger-config.xml

```
<elanmanager-config xmlns="urn:opendaylight:netvirt:elan:config">
   ...
   <temp-smac-learn-timeout>10</temp-smac-learn-timeout>
   <arp-punt-timeout>5</arp-punt-timeout>
   ...
</elanmanager-config>
```

netvirt-natservice-config.xml

```
<natservice-config xmlns="urn:opendaylight:netvirt:natservice:config">
    <nat-mode>controller</nat-mode>
    <snat-punt-timeout>5</snat-punt-timeout>
</natservice-config>
```

Clustering considerations

N.A.

Other Infra considerations

None.

Security considerations

None.

Scale and Performance Impact

This change should reduce the packet in load on the controller from subnet route, ARP and SNAT punts. This will result in overall higher performance on the controller side.

Targeted Release

Fluorine

Alternatives

None.

Usage

N/A.

Features to Install

odl-netvirt-openstack

REST API

N/A.

CLI

N/A.

Implementation

Assignee(s)

Primary assignee: Ravindra Nath Thakur (ravindra.nath.thakur@ericsson.com)

Other contributors: Vinayak Joshi (vinayak.joshi@ericsson.com)

Work Items

N/A.

Dependencies

None

Testing

Unit Tests

Existing ARP/Subnet Route and SNAT functionality will be tested.

Integration Tests

N/A.

CSIT

CSIT testcases will be added for all the punt scenarios covered in the spec which will check learnt flows are getting created and packet counter for learnt flows. Test cases will also be added to check whether learnt flows are getting deleted after the configured hard timeout value.

Documentation Impact

Pipeline documentation should be updated accordingly to reflect the changes to the different services.

References

Table of Contents

OVS Based NA Responder for IPv6 default Gateway

https://git.opendaylight.org/gerrit/#/q/topic:ovs_based_na_responder_for_gw

This spec addresses OVS Based Neighbor Advertisement(NA) responder for IPv6 default Gateway. Neighbor Solicitation(NS) request which has initiated from IPv6 configured/enabled Data Center(DC) VMs are served by OVS based NA responder.

Problem description

In IPv6, whenever a VM wants to communicate to a VM in a different subnet, the MAC address of the IPv6 subnet gateway must be resolved. For that VM generates a Neighbor Solicitation(NS) message to resolve the IPv6 subnet gateway MAC address, which is currently being served by ODL controller which responds with Neighbor Advertisement(NA) message.

Having ODL controller dependency for resolving IPv6 subnet gateway MAC address would result in L3 forwarding failures whenever control plane is down (or if that OVS is disconnected from the Controller). To resolve this issue, it must be possible to resolve the gateway MAC in OVS itself.

This feature targets to provide OVS based NA responder to respond to NS message with GW MAC address during ODL controller downtime to achieve IPv6 L3 forwarding function to be continued.

Use Cases

1. OVS based NA responder for gateway MAC address resolution.
2. OVS based NA responder for gateway MAC address for reachability detection.
3. OVS based NA responder for hidden IPv6 address and MAC address.

Proposed change

OVS based NA responder for gateway MAC address resolution

Background: OVS based ARP Responder for IPv4 gateway MAC resolution

The current VPNv4 implementation ensures that OVS responds to ARP requests for resolving IPv4 default gateway MAC address. This will ensure that the L3 services continue to function even if ODL controller is down or OVS is disconnected from the ODL controller.

Proposal:

OVS based IPv6 NA responder needs to be implemented to resolve the default gateway MAC address which will be similar to IPv4 OVS based ARP responder.

Configuration Variable to enable/disable OVS based NA responder:

Following configuration variable will be added to ipv6service module so that ODL controller must continue to support both controller based and OVS switch based NA responder.

```
<ipv6service-config xmlns="urn:opendaylight:netvirt:ipv6service-config">
  <na-responder-mode>switch</na-responder-mode> or <na-responder-mode>controller</na-
↪responder-mode>
</ipv6service-config>
```

Default NA responder mode will be set it as switch mode.

Openflow Plugin Changes:

The OF plugin in ODL will be enhanced to support below OVS extension in OF plugin master branch.

- **OFPXMT_OFB_ICMPV6_ND_RESERVED**
  - Options-(Router[R], Solicitation[S], Override[O])
- **OFPXMT_OFB_ICMPV6_NS_OPTION_TYPE**
  - Options-(1->SLL, 2->TLL)

OVS based NA responder for gateway MAC address resolution:

When a VM is booted in a network containing IPv6 subnet and the subnet is associated with a neutron router, the ODL controller will do the following match criteria and will install the appropriate open flow rules in the ARP_RESPONDER_TABLE (table 81) when responding to the NS request which has initiated from the IPv6 configured/enabled VMs.

Currently, NS packets for resolving gateway MAC address are punted to the ODL controller from IPV6_TABLE(table 45).

The Neutron Router port has two IPs. One from the Subnet CIDR and the other which is the Link Local Address(LLA)

- Neutron router port having IPv6 subnet CIDR IP.

```
cookie=0x4000000, duration=3053.224s, table=45, n_packets=0, n_bytes=0,
priority=50,icmp6,metadata=0x900004138a000000/0xfffffffffffffffe,icmp_
↪type=135,icmp_code=0,
nd_target=2001:db8:0:2:0:0:0:1 actions=CONTROLLER:65535
```

- Neutron router port having IPv6 Link Local Address(LLA).

```
cookie=0x4000000, duration=3053.224s, table=45, n_packets=0, n_bytes=0,
priority=50,icmp6,metadata=0x900004138a000000/0xfffffffffffffffe,icmp_
↪type=135,icmp_code=0,
nd_target=fe80::f816:3eff:fecc:9e83 actions=CONTROLLER:65535
```

The action for the above flow needs to be changed to forward the NS packets to ARP_RESPONDER_TABLE(table 81) which will respond to the NS request for resolving gateway MAC address. For doing this NS to NA translation at ARP_RESPONDER_TABLE(table 81), it is required to change icmpv6_type from 135(NS) to 136(NA) and icmpv6_options_type to 2 as Target Link Layer Address (TLL)

```
cookie=0x4000000, duration=3053.224s, table=45, n_packets=0, n_bytes=0,
priority=50,icmp6,metadata=0x4138a000000/0xffffffffff000000,icmp_type=135,
↪icmp_code=0,
nd_target=2001:db8:0:2:0:0:0:1, nd_sll=fa:16:3e:55:ad:df
actions=set_field:136->icmpv6_type,set_field:0->icmpv6_code,set_field:2->
↪icmpv6_options_type,goto_table:81

cookie=0x4000000, duration=3053.224s, table=45, n_packets=0, n_bytes=0,
priority=50,icmp6,metadata=0x4138a000000/0xffffffffff000000,icmp_type=135,
↪icmp_code=0,
nd_target=fe80::f816:3eff:fecc:9e83, nd_sll=fa:16:3e:55:ad:df
actions=set_field:136->icmpv6_type,set_field:0->icmpv6_code,set_field:2->
↪icmpv6_options_type,goto_table:81
```

For each VM port (Also for hidden IPs), OVS based NA responder flow will be programmed in ARP_RESPONDER_TABLE(table 81) as mentioned below.

Neighbor Solicitation(NS) messages can be classified into two types

- NS message having valid source IPv6 address (e.g., 2001:db8:0:2:f816:3eff:feef:c47a) and source MAC address (e.g., 00:11:22:33:44:55)

  In this case ODL controller will program the NA responder flow with Unicast destination IPv6 address (Which is NS source IPv6 address). In this case NS request will contain the VMs vNIC MAC address information in the ICMPv6 option field Source Link Layer Address(SLL).

  Example:

```
cookie=0x12220d57, duration=0.0s, table=81, n_packets=0, n_bytes=0,␣
↪priority=80, icmp6,
icmp_type=136,icmp_code=0, metadata=0x4138a000000/0xffffffffff000000,nd_
↪target=2001:db8:0:2:0:0:0:1
actions= move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_
↪field:00:23:15:d3:22:01->eth_src,
move:NXM_NX_IPV6_SRC[]->NXM_NX_IPV6_DST[],set_field:2001:db8:0:2:0:0:0:1->
↪ipv6_src,
set_field:00:23:15:d3:22:01->nd_tll,set_field:0xE000->OFPXMT_OFB_ICMPV6_
↪ND_RESERVED,
load:0->NXM_OF_IN_PORT[],output:2

cookie=0x12220d57, duration=0.0s, table=81, n_packets=0, n_bytes=0,␣
↪priority=80, icmp6,
```

(continues on next page)

(continued from previous page)

```
icmp_type=136,icmp_code=0, metadata=0x4138a000000/0xfffffffff000000,nd_
↪target=fe80::f816:3eff:fecc:9e83
actions= move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_
↪field:00:23:15:d3:22:01->eth_src,
move:NXM_NX_IPV6_SRC[]->NXM_NX_IPV6_DST[],set_
↪field:fe80::f816:3eff:fecc:9e83->ipv6_src,
set_field:00:23:15:d3:22:01->nd-tll,set_field:0xE000->OFPXMT_OFB_ICMPV6_
↪ND_RESERVED,
load:0->NXM_OF_IN_PORT[],output:2
```

Note: In this case following NA flags will be set Router -> 1 Solicitation -> 1 Override -> 1

- NS message having unspecified (::) source IPv6 address

  In this case NS request needs to be redirecting the packets to the ODL controller for responding to the NS request. Since without SLL option from the NS request OVS switch may not be set TLL filed in NA response packet.

  Example:

```
cookie=0x4000000, duration=3053.224s, table=45, n_packets=0, n_bytes=0,
priority=50,icmp6,metadata=0x900004138a000000/0xfffffffffffffffe,icmp_
↪type=135,icmp_code=0,
nd_target=2001:db8:0:2:0:0:0:1 actions=CONTROLLER:65535

cookie=0x4000000, duration=3053.224s, table=45, n_packets=0, n_bytes=0,
priority=50,icmp6,metadata=0x900004138a000000/0xfffffffffffffffe,icmp_
↪type=135,icmp_code=0,
nd_target=fe80::f816:3eff:fecc:9e83 actions=CONTROLLER:65535
```

Note: In this case if there is no specific match found in IPV6_TABLE(table 45) for NS packet, it will be redirecting to the ODL controller matching with elan tag value in metadata field.

All the mentioned example flows in the spec will require changes in the OVS to support new attributes(OFPXMT_OFB_ICMPV6_ND_RESERVED and OFPXMT_OFB_ICMPV6_NS_OPTION_TYPE) and we will be working on getting those changes into OVS community.

OVS based NA responder for gateway MAC address for reachability analysis

After the MAC address for a particular gateway is resolved, the IPv6 VM periodically generates NS requests to ensure the neighbor is reachable.

- **This message can arrive as a Unicast message addressed to the Gateway MAC**
  - NS can be sent from both Neutron ports and hidden IPs.
- **The message format can be different than the broadcast/multicast NS message**
  - The option field MAY/MAY NOT contain source link layer address.
- **For such messages, a response must be generated. However, the response NEED NOT include the MAC address**
  - With proposal, gateway MAC address is not included in the NA response.

Programming NA responder flows in OVS for IPv6 subnet gateway:

The following cases needs to be handled for programming/un-programming the OVS based NA responder flows.

1) Router Association to subnet
2) Router disassociation from subnet
3) VM boot-up on a OVS
4) VM shutdown
5) VM Migration
6) VM Port Update
7) OVS disconnections

Pipeline changes

Flow needs to be programmed in IPV6_TABLE(table 45) for redirecting the Neighbor Solicitation(NS) packets to ARP_RESPONDER_TABLE(table 81) matching with ND target address as IPv6 subnet GW IP.

```
cookie=0x4000000, duration=506.885s, table=17, n_packets=0, n_bytes=49916,␣
↪priority=10,
metadata=0xc60000000000/0xffffff0000000000 actions=write_
↪metadata:0x900004138a000000/0xfffffffffffffffe,
goto_table:45

cookie=0x4000000, duration=506.974s, table=45, n_packets=0, n_bytes=0,␣
↪priority=50, icmp6,
metadata=0x4138a000000/0xffffffffff000000, icmp_type=135, icmp_code=0, nd_
↪target=<Subnet_CIDR_GW_IP>,
nd_sll=fa:16:3e:55:ad:df
actions=set_field:136->icmpv6_type,set_field:0->icmpv6_code,set_field:2->
↪icmpv6_options_type,goto_table:81

cookie=0x4000000, duration=506.974s, table=45, n_packets=0, n_bytes=0,␣
↪priority=50, icmp6,
metadata=0x4138a000000/0xffffffffff000000, icmp_type=135, icmp_code=0, nd_
↪target=<Router_port_LLA>,
nd_sll=fa:16:3e:55:ad:df
actions=set_field:136->icmpv6_type,set_field:0->icmpv6_code,set_field:2->
↪icmpv6_options_type,goto_table:81
```

OVS NA responder flow for GW MAC resolution for NS packet which contains SLL option field and valid IPv6 source address:

```
cookie=0x12220d57, duration=0.0s, table=81, n_packets=0, n_bytes=0,␣
↪priority=80,icmp6,
icmp_type=136, metadata=<matches elan + lport tag>, nd_target=<Subnet_CIDR_
↪GW_IP>
actions= move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
set_field:<GW-Mac-Address>->eth_src,move:NXM_NX_IPV6_SRC[]->NXM_NX_IPV6_
↪DST[],
set_field:<Subnet_CIDR_GW_IP>->ipv6_src,set_field:<GW-mac-Address>->nd-tll,
set_field:0xE000->OFPXMT_OFB_ICMPV6_ND_RESERVED,load:0->NXM_OF_IN_PORT[],
↪output:<VM port>
```

(continues on next page)

(continued from previous page)

```
cookie=0x12220d57, duration=0.0s, table=81, n_packets=0, n_bytes=0,␣
↪priority=80,icmp6,
icmp_type=136, metadata=<matches elan + lport tag>, nd_target=<Router_port_
↪LLA>
actions= move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
set_field:<GW-Mac-Address>->eth_src,move:NXM_NX_IPV6_SRC[]->NXM_NX_IPV6_
↪DST[],
set_field:<Router_port_LLA>->ipv6_src,set_field:<GW-mac-Address>->nd-tll,
set_field:0xE000->OFPXMT_OFB_ICMPV6_ND_RESERVED,load:0->NXM_OF_IN_PORT[],
↪output:<VM port>
```

OVS NA responder flow for GW MAC address reachability checking for NS packet without containing Option SLL field and valid IPv6 source address:

```
cookie=0x12220d57, duration=0.0s, table=81, n_packets=0, n_bytes=0,␣
↪priority=80, icmp6, icmp_type=136,
metadata=<matches elan + lport tag>,nd_target=<Subnet_CIDR_GW_IP>
actions= move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
set_field:<GW-Mac-Address>->eth_src,move:NXM_NX_IPV6_SRC[]->NXM_NX_IPV6_
↪DST[],
set_field:<Subnet_CIDR_GW_IP>->ipv6_src,
set_field:0xE000->OFPXMT_OFB_ICMPV6_ND_RESERVED,load:0->NXM_OF_IN_PORT[],
↪output:<VM port>

cookie=0x12220d57, duration=0.0s, table=81, n_packets=0, n_bytes=0,␣
↪priority=80, icmp6, icmp_type=136,
metadata=<matches elan + lport tag>,nd_target=<Router_port_LLA>
actions= move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
set_field:<GW-Mac-Address>->eth_src,move:NXM_NX_IPV6_SRC[]->NXM_NX_IPV6_
↪DST[],
set_field:<Router_port_LLA>->ipv6_src,
set_field:0xE000->OFPXMT_OFB_ICMPV6_ND_RESERVED,load:0->NXM_OF_IN_PORT[],
↪output:<VM port>
```

OVS NA responder flow for GW MAC resolution for NS packet without containing Option SLL field and unspecified IPv6 source address:

In this case NS request needs to be redirecting the packets to the ODL controller for responding to the NS request. Since without SLL option field from the NS request OVS switch may not be able to set TLL filed in NA response packet.

```
cookie=0x4000000, duration=3053.224s, table=45, n_packets=0, n_bytes=0,
priority=50,icmp6,metadata=0x900004138a000000/0xfffffffffffffffe,icmp_
↪type=135,icmp_code=0,
nd_target=2001:db8:0:2:0:0:0:1 actions=CONTROLLER:65535

cookie=0x4000000, duration=3053.224s, table=45, n_packets=0, n_bytes=0,
priority=50,icmp6,metadata=0x900004138a000000/0xfffffffffffffffe,icmp_
↪type=135,icmp_code=0,
nd_target=fe80::f816:3eff:fecc:9e83 actions=CONTROLLER:65535
```

Yang changes

For the new configuration knob a new yang ipv6service-config shall be added in IPv6 service, with the container for holding the IPv6 NA responder mode configured. It will have two options controller and switch, with switch being the default.

```
container ipv6service-config {
  config true;
  leaf na-responder-mode {
      type enumeration {
          enum "controller";
          enum "switch";
      }
      default "switch";
  }
}
```

Limitations

ODL controller dependency is still required for one of the corner UC as below.

- NS packet without containing Option SLL field and unspecified IPv6 source address (::)

Configuration impact

The proposed change requires the IPv6 service to provide a configuration knob to switch between the controller based/switch based implementation. A new configuration file netvirt-ipv6service-config.xml shall be added with default value switch.

```
<ipv6service-config xmlns="urn:opendaylight:netvirt:ipv6service-config">
  <na-responder-mode>switch</na-responder-mode>
</ipv6service-config>
```

The dynamic update of na-responder-mode will not be supported. To change the na-responder-mode the controller cluster needs to be restarted after changing the na-responder-mode. On restart the IPv6 NA responder for gateway MAC address lifecycle will be reset and after the controller comes up in the updated na-responder-mode, a new set of ovs flows will be installed on the openvswitch and it can be different from the ones that were forwarding traffic earlier.

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

The new OVS based NA responder implementation is expected to improve the performance when compared to the existing one and will reduce the overhead of the ODL controller.

Targeted Release

Fluorine

Alternatives

None

Usage

Create Internal Networks and Subnets

```
openstack network create vpn6_net_1
openstack network create vpn6_net_2

openstack subnet create --network vpn6_net_1 --subnet-range 2001:db8:0:2::/64 vpn6_
↪sub_1 --ip-version=6 --ipv6-address-mode=slaac --ipv6-ra-mode=slaac

openstack subnet create --network vpn6_net_2 --subnet-range 2001:db8:0:3::/64 vpn6_
↪sub_2 --ip-version=6 --ipv6-address-mode=slaac --ipv6-ra-mode=slaac
```

Create router

```
openstack router create vpn6_router
```

Attach IPv6 Subnets to Router

```
openstack router add subnet vpn6_router vpn6_sub_1
openstack router add subnet vpn6_router vpn6_sub_2
```

Create VPNv6 Security Group

```
openstack security group create vpn6_sg
openstack security group rule create vpn6_sg --ingress --ethertype IPv6 --dst-port␣
↪1:65535 --protocol tcp
openstack security group rule create vpn6_sg --egress --ethertype IPv6 --dst-port␣
↪1:65535 --protocol tcp
openstack security group rule create vpn6_sg --ingress --ethertype IPv6 --protocol␣
↪icmp
openstack security group rule create vpn6_sg --egress --ethertype IPv6 --protocol icmp
openstack security group rule create vpn6_sg --ingress --ethertype IPv6 --dst-port␣
↪1:65535 --protocol udp
openstack security group rule create vpn6_sg --egress --ethertype IPv6 --dst-port␣
↪1:65535 --protocol udp
```

Create VM ports:

```
openstack port create --network vpn6_net_1 vpn6_net_1_port_1 --security-group vpn6_sg
openstack port create --network vpn6_net_2 vpn6_net_2_port_1 --security-group vpn6_sg
```

Boot VMs:

```
openstack server create --image <VM-Image> --flavor <VM-Flavor> --nic port-id=vpn6_
↪net_1_port_1 --availability-zone nova:<Hypervisor-Name> <VM-Name>
openstack server create --image <VM-Image> --flavor <VM-Flavor> --nic port-id=vpn6_
↪net_2_port_1 --availability-zone nova:<Hypervisor-Name> <VM-Name>
```

Features to Install

odl-netvirt-openstack

REST API

No new REST API being added.

CLI

No new CLI being added.

Implementation

Assignee(s)

Primary assignee: Karthikeyan Krishnan <karthikeyan.k@altencalsoftlabs.com/karthikeyangceb007@gmail.com>

Other contributors: Somashekar Byrappa <somashekar.b@altencalsoftlabs.com>

Nithi Thomas <nithi.t@altencalsoftlabs.com>

Work Items

- Write a framework which can support multiple modes of NA responder implementation.
- Add support in openflow plugin for OVS based NA responder actions.
- Add support in genius for OVS based NA responder actions.
- Add a config parameter to select between controller based and ovs based NA responder.
- Add the flow programming for OVS based NA responder in netvirt.
- Write Unit tests for OVS based NA responder.

Dependencies

The following OVS extensions are required to support this feature on ODL controller.

- The OVS must implement the OF extensions to support match and set field actions for the RESERVED field(OFPXMT_OFB_ICMPV6_ND_RESERVED) of NA message.
- The OVS must implement the OF extension to modify to the type field of the NS Option from SLL to TLL(OFPXMT_OFB_ICMPV6_NS_OPTION_TYPE).

Testing

The test cases for this feature must cover dual-stack and single-stack VMs and test the OVS based NA responder for both switch and controller mode. This feature should not break any functionality of the existing controller based NA responder.

Test cases below:

1. Verify the OVS Responder Flows for Gateway MAC resolution.
2. Verify the OVS Responder Flows for Reachability analysis.
3. Verify the L2 Data Traffic(ELAN) with Single OVS.
4. Verify the L2 Data Traffic(ELAN) with Multiple OVS.
5. Verify the L3 Data Traffic(L3VPN) with Router Associated with BGP-VPN.
6. Verify the L3 Data Traffic with IPv6 Subnet added to Router.

7. Verify the OVS Responder Flows when OVS is Disconnected.
8. Verify the L3 Data Traffic(L3VPN) when ODL is Disconnected from OVS.

Unit Tests

Unit test needs to be added for the new OVS based NA responder mode. It shall use the component tests framework

Integration Tests

Integration tests needs to be added for the OVS based NA responder flows.

CSIT

The following new CSIT test cases will be added for this feature.

1. Verify the data plane traffic between VM1 and VM2 on same network when ODL controller is down.
2. Verify the data plane traffic between VM1 and VM2 on different network when ODL controller is down.
3. Verify the data plane traffic between VM1 and VM2 on L3 BGP-VPN Scenario when ODL controller is down.
4. Verify the data plane traffic between VM1 and VM2 on same network when ODL controller is Up.
5. Verify the data plane traffic between VM1 and VM2 on different network when ODL controller is Up.
6. Verify the data plane traffic between VM1 and VM2 on same network with single router dual stack network configured VMs.
7. Verify the data plane traffic between VM1 and VM2 on different network with single router dual stack network configured VMs.
8. Verify the data plane traffic between hidden IPv6 configured on VM1 and neutron configured IPv6 on VM2 on same network.
9. Verify the data plane traffic between hidden IPv6 configured on VM1 and neutron configured IPv6 on VM2 on different network.

Documentation Impact

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

References

[1] OpenDaylight Documentation Guide

[2] Neighbor Discovery for IP version 6 (IPv6)

Table of Contents

Retain Elected Napt Switch After Upgrade for SNAT

https://git.opendaylight.org/gerrit/#/q/topic:retain-napt-switch-after-upgrade-for-snat

Important: All gerrit links raised for this feature will have topic name as **retain-napt-switch-after-upgrade-for-snat**

This feature attempts to retain the earlier elected Napt-Switch for a given Router(VRF) before upgrade as Napt-Switch even after the upgrade of `OpenDaylight Controller` in order to minimize dataplane churn for existing SNAT sessions during and after upgrade.

Problem description

For each of the router's in controller-Based SNAT, one of the OVS will be elected as Napt-Switch from the list of candidate OVS(in which Router has it's presence). New Napt-Switch for this router gets re-elected under following scenarios.

- When the elected Napt Switch goes down or rebooted.
- When the last VM on the elected Napt Switch is deleted/moved.
- Cluster Reboot scenario.
- When VM booted on Non-Napt Switch and if the Napt-Switch is down at that time.

With current implementation, whenever a VM is booted on any OVS for a given router, then first we check if there already elected Napt-Switch and in connected state currently. If not, then the current OVS gets itself elected as Napt Switch and all NAT flows for the on-going SNAT session will be installed into the newly elected Napt Switch.

As part of upgrade, the configuration DataStore back-up will be taken and the same will be restored after upgrade. When the OVS are connected back to ODL-controller, if the Non-Napt Switch gets connected first before Napt-Switch, its gets re-elected as Napt-Switch as its finds earlier elected Napt-Switch is in disconnected state. As a result, the other Non-Napt Switches(which are yet to be connected) continue to send the traffic to older Napt-switch resulting in failure for the ongoing SNAT sessions until these Non-napt Switches are connected and updated with flow pointing to newly elected Napt Switch.

Also, when we elect a newNAPT as part of upgrade for routers, there is this possibility of a same OVS getting elected as Napt-Switch for many of the routers (where OVS has the VM presence for these routers) after upgrade.

The workflow will be changed to prevent re-election of Napt Switch as part of upgrade for controller-based SNAT.

Use Cases

- Existing SNAT sessions should continue to work seamlessly during and after upgrade.

Proposed change

Currently, we have a utility flag `upgradeInProgress` to track whether or not upgrade in progress. This flag will be set to true using suitable REST call before start any OVS's are getting connected back ODL-controller.

When OVS gets connnected which is a Non-Napt Switch and if the corresponding Napt-switch is not yet connected and if the `upgradeInProgress` is set true, it will not try to re-elect itself as Napt-Switch and continue as Non-Napt Switch.

After all the OVS are connected back, this flag will be set to false during which NAT will again go through the list of earlier elected Napt Switches and validate there connectivity status. If any of the earlier elected Napt Switch is not connected back, then NAT will trigger re-election to find and re-elect a new Napt-Switch.

Pipeline changes

None

YANG changes

None

Configuration impact

None

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)

Fluorine.

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: Chetan Arakere Gowdru <chetan.arakere@altencalsoftlabs.com>

Other contributors:

Work Items

1. Add Check to prevent re-election of Napt Switch if upgradeInProgress is set.
2. Re-check the connectivity status of earlier elected Napt-Switch after upgrade is completed.
3. Re-elect new Switch if earlier elected Napt Switch is down after upgrade.

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

Test case will be added to validate this feature to the upgrade test suite.

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

- Upgrade in Progress flag

Table of Contents

Subnet Routing for hidden IPv6 addresses

https://git.opendaylight.org/gerrit/#/q/topic:subnet-routing-for-hidden-ipv6

This spec addresses Subnet Routing feature for hidden IPv6 traffic wherein ODL would learn the hidden/invisible IPv6 addresses and set up forwarding rules for both intra-DC and inter-DC communication.

Problem description

Many Virtual network functions (VNFs) use IP addresses that are configured out of band (i.e., outside of Neutron) within neutron subnet (i.e., it cannot be a non-neutron subnet). To discover such IPs and provide L3 routing and L3VPN based forwarding for these hidden IPs, Subnet Route feature is required. Currently, ODL supports subnet routing feature for hidden IPv4 only. This feature extends the solution for hidden IPv6 addresses. Only the IP Addresses that fall in neutron subnet CIDR range will be discovered.

Use Cases

a. A hidden IPv6 address in a valid Neutron IPv6 subnet must be discoverable when traffic is initiated to that address from a source external to the DC (i.e, from an IPv6 address behind the DC-GW). The traffic must succeed to that discovered IPv6 address from the same external source after initial loss of few packets (~10).

b. A hidden IPv6 address in a valid Neutron IPv6 subnet must be discoverable when traffic is initiated to that address from a source internal to the DC (i.e., some other VM on a subnet routable towards this IPv6 subnet). The traffic must succeed to that discovered IPv6 address from the same internal routable source after initial loss of few packets (~10).

c. If the hidden IPv6 address is a globally unique IPv6 address and its in a valid Neutron IPv6 subnet which is on an external network, then such address must be discoverable when traffic is initiated to that address from internet to the DC (from internet IPv6 address towards this IPv6 subnet). The traffic must succeed to that discovered IPv6 address from the internet after initial loss of few packets (~10).

Proposed change

Subnet routing for hidden IPv6 addresses

Background: Subnet routing for hidden IPv4 addresses

Hidden v4 IPs were learnt via two ways:

- The IPv4 addresses were learnt when the VMs sent out GARP messages.
- **When a user within or outside the subnet or DC-GW attempts to communicate with a Hidden IP,**
  - The message was received by the ODL.
  - The ODL would identify the host hosting the hidden IP.
  - Forwarding rules were setup to establish communication between the user and hidden IP.

Proposal: Subnet routing for hidden IPv6 addresses

In general, all the use cases like L2, L3 routing, L3VPN forwarding applicable to IPv4 is also applicable to IPv6 (except NAT) and hence needs to be supported. This spec highlights only the difference which will be handled for IPv6 specifically.

Similar to IPv4, even IPv6 addresses will be learnt via two ways:

- The IPv6 addresses will be learnt when the VMs sends out unsolicited Neighbor Advertisement (NA) messages.
- **When a user within or outside the subnet or DC-GW attempts to communicate with a Hidden IP,**
  - The message will be received by the ODL.
  - The ODL would identify the host hosting the hidden IP by initiating Neighbor Solicitation (NS) message. Learn IPv6 addresses via NA response.
  - Forwarding rules (FIB entries) will be configured to establish communication between the user and hidden IP.

Currently, subnet routing for hidden IPv4 addresses works only when router is associated with BGPVPN. As part of this spec, it will be enhanced (for both IPv4 and IPv6) to support even plain router use cases (i.e., with just subnets associated to router) without BGPVPN configurations.

It will be implemented in two phases:

Phase-I: Adhering to the current design of IPv4, subnet routing for hidden IPv6 addresses will be supported only when router is associated to BGPVPN.

Phase-II: Subnet routing for both IPv4 and IPv6 will be supported even for plain router use cases (i.e., with just subnets associated to router) without BGPVPN configurations.

Details:

It is possible that some hidden IPs may not be discovered by Neighbor Advertisements.

Example:

- VM may not send unsolicited neighbor advertisements.
- There may not be intra-Subnet traffic to the hidden IP.
- **The very first packet arriving to the hidden IP might be from.**
  - A different Subnet attached to the Same router OR.
  - A user outside the DC.

The ODL will not know where to direct the packet since it has not learnt the IP.

For each Router and for each subnet attached to the Router, the L3VPN service

- Will identify a **Designated DPN** to attract traffic for that Subnet. A DPN is eligible for becoming a **Designated DPN** only if there is atleast one active VM IPv6 port on the subnet on that DPN (wherein this subnet also hosts the hidden IPv6 address).
- The subnet route will be matched only when there are no /128 routes matching the dest IP of the packet.
- Traffic matching the **subnet route** entry will be punted to the controller.
- Controller sends out a NS message to the DestIP in the corresponding Subnet.
- The NS message is broadcasted in the ELAN and eventually reaches the VM hosting the IP.
- The VM responds back with a neighbor advertisement message, which is punted to the controller.
- Controller learns location of the IPv6 address by reading the metadata which contains lport tag of neutron port. Forms FIB entry with the NH corresponding to neutron port, programs it in the FIB table.
- FIB programming triggers DC-GW advertisement wherein it advertises routes to BGP neighbor.
- FIB programming also results in programming the remote flow rules in all the other DPNs that have the same VPN footprint.

Criteria for learning Hidden IPv6 addresses

- The hidden IPs will be learnt by ODL ONLY when the subnet is attached to a router. It MUST NOT be learning hidden IPs belonging to a subnet that is NOT associated with any router.
- Hidden IPs will be in the same subnet as the Neutron Subnet configured by OpenStack.
- It must be possible to learn the hidden IPs from both IPv4 and IPv6 subnets to which the same port can be associated (dual stack VMs).
- It must be possible to communicate to the Hidden IP from the same subnet and from another subnet attached to the same router.
- A single VM could have multiple vNICs configured and each vNIC could be associated with a different router (or VPN). The Hidden IPs will be configured on the interface of the VM and the VM can have multiple Hidden IPs in each subnet. It must be possible for ODL to learn the hidden IPs such that Hidden IPs are learnt in the corresponding subnet. There are no leaks from one VPN into another.
- It must be possible for the Hidden IPs to move across VNF instances within the same subnet. ODL must be capable of determining the correct owner of the hidden IP and forward the frames accordingly.

High Availability of Subnet Routing Service

OVS Failure: DPN DISCONNECT

When the DPN (OVS) disconnects, the VPN service

- Must identify whether there are any Subnet routes with the TEP IP of the OVS as the next hop.
- **For each such subnet route**
  - The VPN service will withdraw the route from the DC-GW.
  - **The VPN service will find an alternate designated DPN for the Subnet**
    - If no such OVS exists, then the action is DEFERRED until such an OVS becomes available.
  - The subnet route is re-advertised with the Next-Hop (NH) set to the TEP IP of the alternate OVS.
  - The subnet route is reprogrammed on ALL OVSes (with the VPN footprint) to direct the traffic to the alternate OVS.
  - The discovered hidden IP routes are NOT withdrawn.

OVS Failure: DPN CONNECT

When the DPN (OVS) connects, the VPN service

- Must identify whether there are any DEFERRED Subnet routes.
- **For each such subnet route**
  - The VPN service will check if the connected OVS can become the designated DPN.
  - This is possible if the connected OVS is a OVS that has at least one VM in the Subnet.
  - If the OVS could be a designated DPN for the subnet, then a Subnet route is advertised to the DC-GW with the NH set to the TEP IP of the connected OVS.
  - The OVS is programmed with a Flow rule matching the Subnet with an action to punt the packets to the controller.

TEP (Tunnel End-Point) Failures: TEP DELETE

Whenever a TEP is deleted,

- **The VPN service will identify**
  - The set of subnets for which the OVS was a designated DPN.
  - The set of Hidden IPs hosted in the VMs connected to the affected OVS.
  - The set of Neutron Port IPs attached to the affected OVS.
- The VPN service will immediately withdraw the Neutron Port IPs and Hidden IPs identified.
- For each subnet identified, the actions in *OVS Failure: DPN DISCONNECT* are triggered.

TEP Failures: TEP ADD

Whenever a TEP is added,

- **The VPN service will identify**
  - The set of deferred Subnet Routes.
  - The set of Neutron Port IPs attached to the affected OVS.
- The VPN service will immediately advertise the Neutron Port IPs to the DC-GW.
- For each subnet identified, the actions described in *OVS Failure: DPN CONNECT* are triggered.

Movement of Hidden IP

Learning IPv6 addresses

When the IPv6 hidden IP moves between hosts, the information with the ODL becomes invalid. To recover from this error, the ODL makes use of Unsolicited NA (UNA) message

- When the Hidden IP moves, it is possible that the VM sends out UNA message.
- Punting the NA message to the controller, the ODL will identify that the location of the hidden IP.
- IPv6 address has changed and ODL can inform the DC-GW accordingly.

Last VM on a VPN removed from the designated DPN

When the last VM on a VPN removed from the designated DPN, a new DPN having VM presence for the subnet needs to be elected as the designated DPN which will anchor the subnet routing.

Limiting Flow Cache

For every Hidden IP discovered, the VPN Service will maintain a FLOW VALID timer

- The timer value will be global.
- The timer value is configurable via configuration files.
- The default value of the timer should be 2 minutes.

When the timer expires, the VPN Service

- Sends out a Unicast NS message to the VM that is hosting the Hidden IP.
- Starts a ND_MESSAGE_SENT timer.
- The ND_MESSAGE_SENT timer value will be global and configurable via configuration files.
- The default value of the timer should be 30 sec.

If the VPN Service receives a NA message as response before ND_MESSAGE_SENT expires

- The VPN Service restarts the FLOW_VALID timer.

If the ND_MESSAGE_SENT timer expires

- The NS Message is sent again.

If the response is NOT received for the second message as well,

- The VPN Service withdraws the affected Hidden IP from the DC-GW.
- The VPN Service removes the affected Hidden IP from the FIB.
- The VPN Service removes the flow entries that correspond to the affected Hidden IP from all OVSes.

Pipeline changes

- When a user outside the subnet or DC-GW attempts to communicate with a Hidden IP. If there is no match in FIB for this hidden IP (i.e., the hidden IP is unknown so far), then the packets needs to be punted to the controller. So that the controller could identify the host hosting the hidden IP by initiating Neighbor Solicitation (NS) message then learn IPv6 addresses via NA response.

  Currently, VPN service programs FIB entries in L3 FIB table (21) for both IPv4 and IPv6 subnets (e.g., match on nw_dst=10.0.0.0/24 or ipv6_dst=1001:db8:0:2::/64) only when router is associated with BGPVPN. But actually it needs be programmed even when just subnet is associated with a router to support intra-DC traffic for hidden IPs across subnets. These flows matching on subnet forward packets from FIB table (21) to subnet route table (22).

  e.g.:

```
cookie=0x8000003, duration=350.898s, table=21, n_packets=0, n_bytes=0,␣
↪priority=74,ipv6,metadata=0x30d70/0xfffffe,ipv6_dst=1001:db8:0:2::/64␣
↪actions=write_metadata:0x138c030d70/0xfffffffffe,goto_table:22
cookie=0x8000003, duration=350.898s, table=21, n_packets=0, n_bytes=0,␣
↪priority=74,ipv6,metadata=0x30d70/0xfffffe,ipv6_dst=2001:db8:0:2::/64␣
↪actions=write_metadata:0x138d030d70/0xfffffffffe,goto_table:22
```

  In order to punt packets to the controller, there is no need of additional flow as it already exists in L3 subnet route table (22) as below.

```
cookie=0x8000004, duration=12731.641s, table=22, n_packets=0, n_bytes=0,␣
↪priority=0 actions=CONTROLLER:65535
```

- Flow needs to be programmed in IPv6 table (45) for punting Neighbor Advertisements (NA) to the controller and forward the packet further in the pipeline as well. These NA packets are used for learning the hidden IPs.

  Only NAs from Global Unicast Address (GUA) IPv6 addresses excluding from neutron port Fixed IPs and Link Local Address (LLA)'s will be punted to controller.

  In order to exclude NAs from neutron port Fixed IPs being punted to controller, one flow per fixed GUA IPv6 address will be programmed in IPv6 table (45) which resubmits to dispatcher table (17). e.g.:

```
cookie=0x4000000, duration=382.556s, table=45, n_packets=1, n_bytes=70,␣
↪priority=50,icmp6,metadata=0x138b000000/0xffff000000,icmp_type=136,icmp_code=0,
↪ipv6_src=1001:db8:0:2:f816:3eff:feb4:aaaa actions=resubmit(,17)
cookie=0x4000000, duration=382.556s, table=45, n_packets=1, n_bytes=70,␣
↪priority=50,icmp6,metadata=0x138b000000/0xffff000000,icmp_type=136,icmp_code=0,
↪ipv6_src=1001:db8:0:2:f816:3eff:feb4:bbbb actions=resubmit(,17)
```

  Lower priority flows (e.g., priority=40) matching on subnet CIDR will be programmed to punt NA packets to controller. e.g.:

```
cookie=0x4000000, duration=382.556s, table=45, n_packets=1, n_bytes=70,␣
↪priority=40,icmp6,metadata=0x138a000000/0xffff000000,icmp_type=136,icmp_code=0,
↪ipv6_src=1001:db8:0:2::/64 actions=CONTROLLER:65535,resubmit(,17)
cookie=0x4000000, duration=382.556s, table=45, n_packets=1, n_bytes=70,␣
↪priority=40,icmp6,metadata=0x138b000000/0xffff000000,icmp_type=136,icmp_code=0,
↪ipv6_src=2001:db8:0:2::/64 actions=CONTROLLER:65535,resubmit(,17)
```

(continues on next page)

(continued from previous page)

- The learnt hidden IPv6 addresses will be programmed in FIB table. e.g.:

```
cookie=0x8000003, duration=20.092s, table=21, n_packets=0, n_bytes=0,␣
↪priority=138,ipv6,metadata=0x30d52/0xffffffe,ipv6_
↪dst=1001:db8:0:2:f816:3eff:feb4:deff actions=group:150003
cookie=0x8000003, duration=5.313s, table=21, n_packets=0, n_bytes=0, priority=138,
↪ipv6,metadata=0x30d52/0xffffffe,ipv6_dst=2001:db8:0:2:f816:3eff:fe13:d202␣
↪actions=group:150005
```

Out-of-scope

Subnet Routing feature was made to work for FLAT/VLAN external networks for IPv4 addresses via the PNF (Physical Network Functions) feature. This spec doesn't claim to support IPv6 Subnet Routing feature for FLAT/VLAN external networks and is out-of-scope.

Yang changes

ipv6-ndutil.yang needs to be updated with new RPC to support sending Neighbor Solicitation packet to an OpenFlow group.

```
rpc send-neighbor-solicitation-to-of-group {
    input {
        leaf source-ipv6 {
            type inet:ipv6-address;
            mandatory "true";
        }
        leaf target-ip-address {
            type inet:ipv6-address;
            mandatory "true";
        }
        leaf source-ll-address {
            type yang:mac-address;
            mandatory "true";
        }
        leaf dp-id {
            type uint64;
            mandatory "true";
        }
        leaf of-group-id {
            type uint32;
            mandatory "true";
            description "NS will be sent to the specified OpenFlow group ID.";
        }
    }
}
```

Limitations

Since the Hidden IPs and Neutron IPs are from the same subnet, there would be coordination required to ensure that the IP spaces do not clash.

- This coordination is assumed to be manual and is out of scope of this spec.
- Specifically, ODL will not build/deploy any intelligence to identify IP address clash or recover from it.

Configuration impact

None

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

None

Targeted Release

Fluorine

Alternatives

The solution is about auto-discovery of hidden v6 IPs and provide L3 routing and L3VPN based forwarding for hidden v6 IPs. Alternatively, L3 routing and L3VPN based forwarding for hidden IPs can be achieved by manual configuration of extra/static routes.

Usage

Features to Install

odl-netvirt-openstack

REST API

No new REST API being added.

CLI

No new CLI being added.

Implementation

Assignee(s)

Primary assignee: Somashekar Byrappa <somashekar.b@altencalsoftlabs.com>

Other contributors: Karthikeyan K <karthikeyan.k@altencalsoftlabs.com>

Nithi Thomas <nithi.t@altencalsoftlabs.com>

Work Items

1. Program IPv6 table (45) with flows to punt NA packets to controller.
2. Handle punting IPv6 traffic to controller for unknown hidden IPv6 addresses if subnet is associated to router.
3. Learning hidden IPv6 addresses, program FIB and advertise routes if external VPN is configured.
4. Subnet routing un-programming for hidden IPv6 addresses.
5. Handle OVS disconnect/connect impact on subnet routing for hidden IPv6 addresses.
6. Handle TEP add/delete impact on subnet routing for hidden IPv6 addresses.
7. Discover movement of hidden IPv6 addresses.
8. Limiting flow cache with a flow valid timer.

Dependencies

Testing

This feature builds on the Subnet routing feature for IPv4. It must be ensured that the feature will not break any IPv4 subnet features. The test cases for this feature must cover dual-stack and single-stack VMs and test the subnet route feature for both IPv4 and IPv6 subnets.

- Some VMs must have only IPv4 addresses
- Some VMs must have only IPv6 addresses

- Some VMs must have both IPv4 and IPv6 addresses

Test cases below:

1. Verify traffic between DC-GW and hidden IPv6 address in a L3VPN.
2. Verify traffic to ensure there is no leak between DC-GW and hidden IPv6 address across L3VPNs.
3. Verify traffic between hidden IPv6 address on network1 and VM2 on network2.
4. Verify traffic between hidden IPv6 address to hidden IPv6 address across different networks.
5. Verify traffic between hidden IPv6 address and VM1 on the same network.
6. Verify traffic between hidden IPv6 address to hidden IPv6 address in same network.
7. Verify traffic when hidden IPv6 address is moved between VMs on same DPN.
8. Verify traffic when hidden IPv6 address is moved between VMs on different DPN.
9. Verify traffic when designated DPN for a subnet is disconnected/connected.
10. Verify traffic when TEPs in designated DPN for a subnet is deleted/added.

Unit Tests

Integration Tests

CSIT

A subset (which can be done using CSIT framework) of test cases mentioned in *Testing* needs to be added.

Documentation Impact

References

1.1.4 NetVirt Design Specifications

Contents:

Table of Contents

ACL Statistics

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

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

Drop statistics will be provided for below cases:

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

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

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

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

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

This feature will only be supported with Stateful ACL mode.

Problem description

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

Use Cases

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

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

Proposed change

ACL Changes

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

With the current implementation, drop flows are as below:

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

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

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

```
cookie=0x6900001, duration=938.964s, table=252, n_packets=0, n_bytes=0,␣
↪priority=62015,
        metadata=0x10000000000/0xffffff0000000000, ct_state=+inv+trk actions=drop

cookie=0x6900001, duration=938.969s, table=252, n_packets=0, n_bytes=0, priority=50,
        metadata=0x10000000000/0xffffff0000000000, ct_state=+new+trk actions=drop
```

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

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

Drop packets statistics support

ODL Controller will be updated to provide a new RPC/NB REST API `<get-acl-port-statistics>` in ACL module with `ACL Flow Stats Request` and `ACL Flow Stats Response` messages. This RPC/API will retrieve details of dropped packets by Security Group rules for all the neutron ports specified as part of `ACL Flow Stats Request`. The retrieved information (instantaneous) received in the OF reply message is formatted as `ACL Flow Stats Response` message before sending it as a response towards the NB.

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

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

Pipeline changes

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

Yang changes

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

Listing 27: 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;
```

(continues on next page)

(continued from previous page)

```
        }
    }

    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 {
```

(continues on next page)

(continued from previous page)

```
            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"
            }
```

(continues on next page)

(continued from previous page)


```
        },
        {
            "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": {
```


(continues on next page)

(continued from previous page)

```
                    "invalid-drop-count": "0",
                    "drop-count": "4"
                }
            },
            {
                "direction": "egress",
                "bytes": {
                    "invalid-drop-count": "168",
                    "drop-count": "378"
                },
                "packets": {
                    "invalid-drop-count": "2",
                    "drop-count": "9"
                }
            },
            {
                "interface-name": "6c53df3a-3456-11e5-a151-feff819cdc9f",
                "error": {
                    "error-message": "Interface not found in datastore."
                }
            }]
        }]
    }
}
```

Note: Below are error messages for the interface:

(a) "Interface not found in datastore."

(b) "Failed to find device for the interface."

(c) "Unable to retrieve drop counts due to error: <<error message>>"

(d) "Unable to retrieve drop counts as interface is not configured for statistics collection."

(e) "Operation not supported for ACL <<Stateless/Transparent/Learn>> mode"

CLI

No CLI being added for this feature

Implementation

Assignee(s)

Primary assignee: <Somashekar Byrappa>

Other contributors: <Shashidhar R>

Work Items

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

Dependencies

This doesn't add any new dependencies.

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

Bug 7232 - Problem observed with "get-flow-statistics" RPC call

Testing

Unit Tests

Following test cases will need to be added/expanded

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

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

Integration Tests

Integration tests will be added, once IT framework is ready

CSIT

Following test cases will need to be added/expanded

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

Documentation Impact

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

References

N.A.

Table of Contents

ACL Remote ACL - Indirection Table to Improve Scale

ACL Remote ACL Indirection patches: https://git.opendaylight.org/gerrit/#/q/topic:remote_acl_indirection

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

Problem description

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

For example, consider we have:

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

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

Use Cases

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

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

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

Proposed change

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

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

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

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

ACL1 (211/241):

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

ACL2 (212/242):

- For each port with a single SG - we will match on the IPs and the ELAN ID (for tenant awareness) here, and set the SG ID in the metadata, before going to the ACL3 table.
- For any port with multiple SGs (or with no SG) - an empty value (0x0) will be set as the SG ID in the metadata, to avoid potential garbage in the SG ID, and goto ACL3 table.

ACL3 (213/243):

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

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

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

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

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

Pipeline changes

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

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

(X) These are the regular rules, not configured with any remote SG.
(XX) These are the proposed rules with the optimization - assuming the lport is using a single ACL.
(XXX) These are the remote SG rules in the current implementation, which we will fall back to if the lport has multiple ACLs.

Table Numbering:

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

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

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

ACL1: INGRESS/EGRESS_ACL_TABLE ACL2: INGRESS/EGRESS_ACL_REMOTE_ACL_TABLE ACL3: INGRESS/EGRESS_ACL_FILTER_TABLE

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

Yang changes

None.

Configuration impact

None.

Clustering considerations

None.

Other Infra considerations

None.

Security considerations

None.

Scale and Performance Impact

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

Targeted Release

Carbon

Alternatives

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

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

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

Usage

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

Functionality should remain as before in any case.

Features to Install

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

- odl-netvirt-openstack

REST API

None.

CLI

Refer to the Neutron CLI Reference[1] for the Neutron CLI command syntax for managing Security Rules with Remote Security Groups.

Implementation

Assignee(s)

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

Primary assignee:

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

Other contributors:

- ?

Work Items

Task list in Carbon Trello

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

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

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

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

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

Problem description

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

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

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

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

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

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

Use Cases

The new ACL implementation will affect the below use cases,

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

Proposed change

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

In this approach, we will use the "ct_mark" flag of connection tracker. The default value of ct_mark is zero.

- ct_mark=0 matches the packet in new state
- ct_mark=1 matches the packet in established state

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

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

In every SG flow, we will have below changes, "table=213/243, priority=3902, **ct_state=+trk** ,icmp,reg6=0x200/0xfffff00 actions=ct(commit,zone=6001, **exec(set_field:0x1->ct_mark)**),resubmit(,17/220)"

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

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

In conntrack flow, "table=213/243, priority=62020,ct_state=-new+est-rel-inv+trk, **ct_mark=0x1** actions=resubmit(,17/220)" "table=213/243, priority=62020,ct_state=-new-est+rel-inv+trk, **ct_mark=0x1** actions=resubmit(,17/220)"

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

In default drop flow of table 213/243, "table=213, n_packets=0, n_bytes=0, priority=50, **ct_state=+trk** ,metadata=0x20000000000/0xfffff0000000000 actions=drop" "table=243, n_packets=6, n_bytes=588, priority=50, **ct_state=+trk** ,reg6=0x300/0xfffff00 actions=drop"

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

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

[1] "table=212/242, n_packets=73, n_bytes=7154, priority=40,icmp,reg6=0x200/0xfffff00,ct_mark=1 actions=ct(commit, zone=5500, **exec(set_field:0x0->ct_mark)**),goto_table:ACL4"

- It will match the ct_mark value with the one and change the ct_mark to zero.

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

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

Please refer the Pipeline Changes for table information.

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

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

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

"table=17, priority=10,metadata=0x20000000000/0xffffff0000000000 actions=write_metadata:0x900002157f000000/0xfffffffffffffffe, **goto_table:210**"

"table=220, priority=6,reg6=0x200 actions=load:0x90000200->NXM_NX_REG6[],write_metadata:0x157f000000/0xfffffffffe, **goto_table:240**"

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

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

Sample flows to be installed in each scenario,

SG rule addition

SG flow: [ADD] "table=213/243, n_packets=33, n_bytes=3234, priority=62021, **ct_state=+trk**, icmp, reg6=0x200/0xfffff00 actions=ct(commit,zone=6001, **exec(set_field:0x1->ct_mark)**),resubmit(,17/220)"

Conntrack flow: [DEFAULT] "table=213/243, n_packets=105, n_bytes=10290, priority=62020,ct_state=-new+est-rel-inv+trk, **ct_mark=0x1** actions=resubmit(,17/220)"

SG Rule deletion

SG flow: [DELETE] "table=213/243, n_packets=33, n_bytes=3234, priority=62021, ct_state=+trk,icmp,reg6=0x200/0xfffff00 actions=ct(commit,zone=6001,exec(set_field:0x1->ct_mark)),resubmit(,17/220)"

New flow: [ADD] "table=212/242, n_packets=73, n_bytes=7154, priority=62021, **ct_mark=0x1**,icmp,reg6=0x200/0xfffff00 actions=ct(commit, **exec(set_field:0x0->ct_mark)**),goto_table:213/243"

Rule Restore

SG flow: [ADD] "table=213/243, n_packets=33, n_bytes=3234, priority=62021, ct_state=+trk, icmp,reg6=0x200/0xfffff00 actions=ct(commit,zone=6001,exec(set_field:0x1->ct_mark)),resubmit(,17/220)"

New flow: [DELETE] "table=212/242, n_packets=73, n_bytes=7154, priority=62021,ct_mark=0x1,icmp,reg6=0x200/0xfffff00 actions=ct(commit,exec(set_field:0x0->ct_mark)),goto_table:213/243"

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

"table=210/240, n_packets=1, n_bytes=98, priority=0 actions=drop"

Flow Sample:

Egress flows before the changes,

cookie=0x6900000, duration=30.590s, table=17, n_packets=108, n_bytes=10624, priority=10,metadata=0x20000000000/0xffffff0000000000 actions=write_metadata:0x9000021389000000/0xfffffffffffffffe,goto_table:211
cookie=0x6900000, duration=30.247s, table=211, n_packets=0, n_bytes=0, priority=61010,ipv6,dl_src=fa:16:3e:93:dc:92,ipv6_src=fe80::f816:3eff:fe93:dc92 actions=ct(table=212,zone=5001) cookie=0x6900000, duration=30.236s, table=211, n_packets=96, n_bytes=9312, priority=61010,ip,dl_src=fa:16:3e:93:dc:92,nw_src=10.100.5.3 actions=ct(table=212,zone=5001) cookie=0x6900000, duration=486.527s, table=211, n_packets=2, n_bytes=180, priority=0 actions=drop
cookie=0x6900000, duration=30.157s, table=212, n_packets=0, n_bytes=0, priority=50,ipv6,metadata=0x1389000000/0xffff000000,ipv6_dst=fe80::f816:3eff:fe93:dc92 actions=write_metadata:0x2/0xfffffe,goto_table:212 cookie=0x6900000, duration=30.152s, table=212, n_packets=0, n_bytes=0, priority=50,ip,metadata=0x1389000000/0xffff000000,nw_dst=10.100.5.3 actions=write_metadata:0x2/0xfffffe,goto_table:212 cookie=0x6900000, duration=486.527s, table=212, n_packets=96, n_bytes=9312, priority=0 actions=goto_table:212 cookie=0x6900000, duration=486.056s, table=213, n_packets=80, n_bytes=8128, priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,17) cookie=0x6900000, duration=485.948s, table=213, n_packets=0, n_bytes=0, priority=62020,ct_state=-new-est+rel-inv+trk actions=resubmit(,17) cookie=0x6900001, duration=30.184s, table=213, n_packets=0, n_bytes=0, priority=62015,ct_state=+inv+trk,metadata=0x20000000000/0xfffff0000000000 actions=drop cookie=0x6900000, duration=30.177s, table=213, n_packets=16, n_bytes=1184, priority=1000,ct_state=+new+trk,ip,metadata=0x20000000000/0xfffff0000000000 actions=ct(commit,zone=5001),resubmit(,17) cookie=0x6900000, duration=30.168s, table=213, n_packets=0, n_bytes=0, priority=1001,ct_state=+new+trk,ipv6,metadata=0x20000000000/0xfffff0000000000 actions=ct(commit,zone=5001),resubmit(,17) cookie=0x6900001, duration=30.207s, table=213, n_packets=0, n_bytes=0, priority=50,ct_state=+new+trk,metadata=0x20000000000/0xfffff0000000000 actions=dro

After the changes, flows will be,

cookie=0x6900000, duration=30.590s, table=17, n_packets=108, n_bytes=10624, priority=10,metadata=0x20000000000/0xffffff0000000000 actions=write_metadata:0x9000021389000000/0xfffffffffffffffe,goto_table:210
cookie=0x6900000, duration=30.247s, table=210, n_packets=0, n_bytes=0, priority=61010,ipv6,dl_src=fa:16:3e:93:dc:92,ipv6_src=fe80::f816:3eff:fe93:dc92 actions=ct(table=211,zone=5001) cookie=0x6900000, duration=30.236s, table=210, n_packets=96, n_bytes=9312, pri-

ority=61010,ip,dl_src=fa:16:3e:93:dc:92,nw_src=10.100.5.3 actions=ct(table=211,zone=5001) cookie=0x6900000, duration=486.527s, table=210, n_packets=2, n_bytes=180, priority=0 actions=drop cookie=0x6900000, duration=30.157s, table=211, n_packets=0, n_bytes=0, priority=50,ipv6,metadata=0x1389000000/0xffff000000,ipv6_dst=fe80::f816:3eff:fe93:dc92 actions=write_metadata:0x2/0xffffffe,goto_table:212 cookie=0x6900000, duration=30.152s, table=211, n_packets=0, n_bytes=0, priority=50,ip,metadata=0x1389000000/0xffff000000,nw_dst=10.100.5.3 actions=write_metadata:0x2/0xffffffe,goto_table:212 cookie=0x6900000, duration=486.527s, table=211, n_packets=96, n_bytes=9312, priority=0 actions=goto_table:212 cookie=0x6900000, duration=486.527s, table=212, n_packets=96, n_bytes=9312, priority=0 actions=goto_table:213 cookie=0x6900000, duration=486.056s, table=213, n_packets=80, n_bytes=8128, priority=62020,ct_state=-new+est-rel-inv+trk,ct_mark=0x1 actions=resubmit(,17) cookie=0x6900000, duration=485.948s, table=213, n_packets=0, n_bytes=0, priority=62020,ct_state=-new-est+rel-inv+trk,ct_mark=0x1 actions=resubmit(,17) cookie=0x6900001, duration=30.184s, table=213, n_packets=0, n_bytes=0, priority=62015,ct_state=+inv+trk,metadata=0x20000000000/0xfffff0000000000 actions=drop cookie=0x6900000, duration=30.177s, table=213, n_packets=16, n_bytes=1184, priority=1000,ct_state=+trk,ip,metadata=0x20000000000/0xfffff0000000000 actions=ct(commit,zone=5001,exec(set_field:0x1->ct_mark)),resubmit(,17) cookie=0x6900000, duration=30.168s, table=213, n_packets=0, n_bytes=0, priority=1001,ct_state=+new+trk,ipv6,metadata=0x20000000000/0xfffff0000000000 actions=ct(commit,zone=5001),resubmit(,17) cookie=0x6900001, duration=30.207s, table=213, n_packets=0, n_bytes=0, priority=50,ct_state=+trk,metadata=0x20000000000/0xfffff0000000000 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=0x20000000000/0xfffff0000000000,ct_mark=1,idle_timeout=1800 actions=ct(commit,zone=5001,exec(set_field:0x0->ct_mark)),goto_table:213"

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

Pipeline changes

The propose changes includes:

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

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

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

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

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

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

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

Table	Match	Action
Dispatcher	metadata=service_id:ACL	write_metadata:(elan_id=ELAN, service_id=NEXT), goto_table:210/240 (ACL1)
ACL1 (210/240)		goto_table:ACL2
...		
ACL2 (211/241)		goto_table:ACL3
ACL3 (212/242)	ip,ct_mark=0x1,reg6=0x200/0xffffff00	(set_field:0x0->ct_mark), goto_table:ACL4
ACL3 (212/242)		goto_table:ACL4
ACL4 (213/243)	ct_state=-new+est-rel-inv+trk,ct_mark=0x1	resubmit(,DISPATCHER)
ACL4 (213/243)	ct_state=+trk,priority=3902,ip,reg6=0x200/0xffffff00	set_field:0x1->ct_mark, resubmit(,DISPATCHER)
ACL4 (213/243)	ct_state=+trk, reg6=0x200/0xffffff00	drop
...		

Yang changes

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

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

The nicira-match.yang and the openflowplugin-extension-nicira-match.yang needs to be updated with the ct_mark match.

```
grouping ofj-nxm-nx-match-ct-mark-grouping{
       container ct-mark-values {
```

(continues on next page)

(continued from previous page)

```
            leaf ct-mark {
               type uint32;
            }
             leaf mask {
               type uint32;
            }
        }
    }
```

Configuration impact

None.

Clustering considerations

None.

Other Infra considerations

None.

Security considerations

None.

Scale and Performance Impact

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

Targeted Release

Carbon

Alternatives

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

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

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

Usage

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

Features to Install

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

- odl-netvirt-openstack

REST API

None.

CLI

Refer to the Neutron CLI Reference[1] for the Neutron CLI command syntax for managing Security Rules.

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

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

Dependencies

None.

Testing

Unit Tests

Integration Tests

CSIT

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

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

Documentation Impact

None.

References

Table of Contents

ACL: Support for protocols that are not supported by conntrack

https://git.opendaylight.org/gerrit/#/q/topic:acl-non-conntrack

This spec addresses following issues in ACL module:

a. Enhance ACL to support protocols like OSPF, VRRP etc that are not supported by conntrack in stateful mode.

b. Handle overlapping IP addresses while processing remote ACLs.

c. Optimization for Remote ACL by reducing number of flows even for ports having multiple ACLs.

Problem description

a. Enhance ACL to support protocols like OSPF, VRRP etc that are not supported by conntrack in stateful mode.

   Problem: With current stateful ACL implementation, data packets of IP protocols like OSPF, VRRP, etc are classified as invalid and dropped in ACL tables. This is because conntrack used with stateful mode, does not recognize these IP protocols.

   In the current ACL implementation, all IP traffic (except DHCP and ARP) are passed through the conntrack framework for stateful tracking. Conntrack supports (and hence recognizes) only the following protocols:

   - ICMP
   - TCP
   - UDP

- ICMPv6

Stateful tracking is done only for the above protocols and all other traffic is classified as INVALID by conntrack and hence being dropped by the ACL tables in the OF pipeline.

b. Handle overlapping IP addresses while processing remote ACLs.

Problem: In the current implementation if two or more ports with different ACL's have same IP address (configured via allowed-address-pair), ACL remote table (212/242) will have only one flow per ELAN for that IP address pointing to a remote ACL. Flow for other ACL will be missing. Hence, packet may not hit the right flow and end up dropping the packets.

c. Optimization for Remote ACL by reducing number of flows even for ports having multiple ACLs.

Problem: In the current implementation, optimization for Remote ACL by reducing number of flows is supported only for ports having single ACL but the simlar optimization is not available for ports having multiple ACLs.

Use Cases

a. SDN infrastructure must be able to recognize any IP protocol and depending on the ACL rule configured must ALLOW or DROP the corresponding traffic.

b. Multiple ports can have same IP/MAC in many scenarios. Few instances include VRRP or OSPF cases where multicast IP/MAC (eg: 224.0.0.5/01:00:5e:00:00:05) is configured via allowed-address-pair.

c. Improves scalability and performance.

Proposed change

a. Selectively allow only certain IP protocols to transit through the conntrack framework. The rest of the traffic should bypass the conntrack framework and be governed only by the configured ACL filter rules (except DHCP and ARP). So, the behavior would be as follows:

- DHCP and ARP are not governed by ACL rules and are always allowed.
- A selective set of IP protocols which carry point-to-point datapath information are allowed to pass through conntrack for stateful connection tracking.
- The rest of the protocols are NOT passed through conntrack and are directly checked against the OF translated ACL filter rules.

b. New yang container (acl-ports-lookup) is introduced to handle overlapping IP addresses while processing remote ACLs. This would remove dependency on ID manager for generating flow priorities used in ACL filter tables. With the yang approach, having a unique priority for each flow in ACL filter table is not required anymore. Removal of ID manager dependency for generation of flow priorities would provide better scalability and performance.

c. ACL pipeline is re-designed to reduce number of flows in ACL tables.

ACL0 (239):

- This table is untouched.

ACL1 (210/240):

- Anti-spoofing filter table.
- An additional match on lport tag is added for anti-spoofing flows to support overlapping IP addresses where multiple VMs are configured with same IP/MAC addresses via allowed-address-pair.

```
cookie=0x6900000,table=240,priority=61010,**reg6=0x200/0xfffff00**,ip,dl_
↪dst=fa:16:3e:5c:42:d5,nw_dst=10.10.10.12 actions=goto_table:242
```

ACL2 (211/241):

- Classifies conntrack supported traffic and sends the traffic to conntrack table (ACL3).
- Traffic not supported by conntrack is directly sent to ACL filter table (ACL5).
- Metadata is written to classify conntrack (CONST_0) or non-conntrack (CONST_1) traffic.

ACL3 (212/242):

- Sends traffic to conntrack.

ACL4 (213/243):

- This table number is re-aligned but the functionality remains same as explained in spec #acl-reflection-on-existing-traffic which supports the ACL changes reflecting on existing traffic.

ACL5 (214/244):

- ACL filter cum dispatcher table which is common to both conntrack supported and non-conntrack supported traffic.
- For conntrack supported traffic:
  - If session is already established (ct_state=+est+trk | +rel+trk), packet would get returned to dispatcher table from here itself. It doesn't go through next subsequent ACL tables.
  - INVALID packets (ct_state=+inv+trk) are dropped.
- Other flows are common to both conntrack and non-conntrack supported traffic.
- Flows related to ACL rules are classified into three categories as explained below:

  (i) Flows for ACL rules which are configured with **remote_ip_prefix**. This is straight forward case where packets matching these flows would be directly sent to table ACL8.

```
e.g:
sg1 -> ALLOW IPv4 22/tcp from 0.0.0.0/0

cookie=0x6900000,table=244,priority=62040,tcp,reg6=0x600/0xfffff00,tp_dst=22␣
↪actions=goto_table:247
```

  (ii) Ports having single or multiple SGs but all the rules with a common (single) remote SG. In this case, flows **doesn't match** on ACL rules in this table but instead will match ACL rules in ACL6 table. To support this usecase, dispatcher kind of mechanism is performed to loop/iterate through all the rules having remote ACLs.

```
sg1 -> ALLOW IPv4 icmp from sg1
sg1 -> ALLOW IPv4 22/tcp from sg1
sg2 -> ALLOW IPv4 100/udp from sg1

cookie=0x6900000,table=244,priority=62030,reg6=0x200/0xfffff00,metadata=0x100/
↪0xffffc actions=drop
cookie=0x6900000,table=244,priority=62010,reg6=0x200/0xfffff00 actions=write_
↪metadata:0x100/0xffffc,goto_table:245
```

  (iii) Ports having single or multiple SGs with collective ACL rules having different remote SGs. Approach followed is same as (ii), flows **doesn't match** on ACL rules in this table but instead will match ACL rules in ACL6 table. To support this usecase, dispatcher kind of mechanism is performed to loop/iterate through all the rules having remote ACLs.

Example-1: Port having single SG (sg1).

```
sg1 -> ALLOW IPv4 22/tcp from sg1
sg1 -> ALLOW IPv4 icmp from sg2

cookie=0x6900000,table=244,priority=62030,reg6=0x200/0xffffff00,metadata=0x200/
↪0xffffffc actions=drop
cookie=0x6900000,table=244,priority=62020,reg6=0x200/0xffffff00,metadata=0x100/
↪0xffffffc actions=write_metadata:0x200/0xffffffc,goto_table:245
cookie=0x6900000,table=244,priority=62010,reg6=0x200/0xffffff00 actions=write_
↪metadata:0x100/0xffffffc,goto_table:245
cookie=0x6900000,table=244,priority=0 actions=drop
```

Example-2: Port having multiple SGs (sg1, sg2 and sg3).

```
sg1 -> ALLOW IPv4 22/tcp from sg1
sg1 -> ALLOW IPv4 icmp from sg2
sg2 -> ALLOW IPv4 80/tcp from sg2
sg3 -> ALLOW IPv4 100/udp from sg3

cookie=0x6900000,table=244,priority=62030,reg6=0x200/0xffffff00,metadata=0x300/
↪0xffffffc actions=drop
cookie=0x6900000,table=244,priority=62020,reg6=0x200/0xffffff00,metadata=0x200/
↪0xffffffc actions=write_metadata:0x300/0xffffffc,goto_table:245
cookie=0x6900000,table=244,priority=62020,reg6=0x200/0xffffff00,metadata=0x100/
↪0xffffffc actions=write_metadata:0x200/0xffffffc,goto_table:245
cookie=0x6900000,table=244,priority=62010,reg6=0x200/0xffffff00 actions=write_
↪metadata:0x100/0xffffffc,goto_table:245
cookie=0x6900000,table=244,priority=0 actions=drop
```

- To handle rules having remote SG, flows in this table are grouped based on remote SG. Flows for rules having common remote ACL are grouped together and matched based on remote SG ID.

ACL6 (215/245):

- ACL filter table for ports having single or multiple remote SGs.
- ACL filters are matched per remote SG in this table.
- Table miss would resubmit back to ACL5 table to iterate for the next remote SG.

Below are some of the cases where ports associated to single or multiple SGs collectively having rules with different remote SGs.

(i) Port having single SG (sg1).

```
sg1 -> ALLOW IPv4 22/tcp from sg1
sg1 -> ALLOW IPv4 icmp from sg2

cookie=0x6900000,table=245,priority=61010,tcp,reg6=0x200/0xffffff00,metadata=0x100/
↪0xffffffc,tp_dst=22 actions=goto_table:246
cookie=0x6900000,table=245,priority=61010,icmp,reg6=0x200/0xffffff00,
↪metadata=0x200/0xffffffc actions=goto_table:246
cookie=0x6900000,table=245,priority=0 actions=resubmit(,244)
```

(ii) Port having multiple SGs (sg1, sg2 and sg3).

```
sg1 -> ALLOW IPv4 22/tcp from sg1
sg1 -> ALLOW IPv4 icmp from sg2
sg2 -> ALLOW IPv4 80/tcp from sg2
```

(continues on next page)

(continued from previous page)

```
sg3 -> ALLOW IPv4 100/udp from sg3

cookie=0x6900000,table=245,priority=61010,tcp,reg6=0x200/0xfffff00,metadata=0x100/
↪0xffffc,tp_dst=22 actions=goto_table:246
cookie=0x6900000,table=245,priority=61010,icmp,reg6=0x200/0xfffff00,
↪metadata=0x200/0xffffc actions=goto_table:246
cookie=0x6900000,table=245,priority=61010,tcp,reg6=0x200/0xfffff00,metadata=0x200/
↪0xffffc,tp_dst=80 actions=goto_table:246
cookie=0x6900000,table=245,priority=61010,udp,reg6=0x200/0xfffff00,metadata=0x300/
↪0xffffc,tp_dst=100 actions=goto_table:246
cookie=0x6900000,table=245,priority=0 actions=resubmit(,244)
```

ACL7 (216/246):

- Remote ACL filter table.
- Flows match on remote ACL and corresponding IP addresses.
- This table is independent of ports i.e., no match on lport tag.
- During IP delete scenarios (port delete/update), look-up to yang container (acl-ports-lookup) is performed, flows are deleted only when IP address is not used by any other ports within that ACL.
- Below usecase gives the reason for having looping/iteration based approach for this table.

  Usecase: Packets matching multiple rules having different remote SGs. This is a case where packets can match both rules (ip and icmp filters). But let's say it matches src IP (nw_src=10.10.10.4) from remote SG (sg2). So in this case, ACL6 and ACL7 tables needs to be iterated twice to find the right match.

```
sg1 -> ALLOW IPv4 from sg1
sg1 -> ALLOW IPv4 icmp from sg2

cookie=0x6900000,table=244,priority=62030,reg6=0x200/0xfffff00,metadata=0x200/
↪0xffffc actions=drop
cookie=0x6900000,table=244,priority=62020,reg6=0x200/0xfffff00,metadata=0x100/
↪0xffffc actions=write_metadata:0x200/0xffffc,goto_table:245
cookie=0x6900000,table=244,priority=62010,reg6=0x200/0xfffff00 actions=write_
↪metadata:0x100/0xffffc,goto_table:245
cookie=0x6900000,table=244,priority=0 actions=drop

cookie=0x6900000,table=245,priority=61010,ip,reg6=0x200/0xfffff00,metadata=0x100/
↪0xffffc actions=goto_table:246
cookie=0x6900000,table=245,priority=61010,icmp,reg6=0x200/0xfffff00,
↪metadata=0x200/0xffffc actions=goto_table:246
cookie=0x6900000,table=245,priority=0 actions=resubmit(,244)

cookie=0x6900000,table=246,priority=61010,ip,metadata=0x100/0xffffc,nw_src=10.10.
↪10.6 actions=goto_table:247
cookie=0x6900000,table=246,priority=61010,ip,metadata=0x100/0xffffc,nw_src=10.10.
↪10.12 actions=goto_table:247
cookie=0x6900000,table=246,priority=61010,ip,metadata=0x200/0xffffc,nw_src=10.10.
↪10.4 actions=goto_table:247
cookie=0x6900000,table=246,priority=0 actions=resubmit(,244)
```

ACL8 (217/247):

- Packets reaching this table would have passed all the ACL filters. Traffic could be of both conntrack and non-conntrack supported.
- In case of conntrack traffic, commits the session in conntrack and resubmits to dispatcher.

```
cookie=0x6900000,table=247,priority=61010,ip,reg6=0x200/0xfffff00,metadata=0x0/
↪0x2 actions=ct(commit,zone=5002,exec(set_field:0x1->ct_mark)),resubmit(,220)
cookie=0x6900000,table=247,priority=61010,ipv6,reg6=0x200/0xfffff00,metadata=0x0/
↪0x2 actions=ct(commit,zone=5002,exec(set_field:0x1->ct_mark)),resubmit(,220)
```

- In case of non-conntrack traffic, resubmits to dispatcher.

```
cookie=0x6900000,table=247,priority=61010,reg6=0x200/0xfffff00,metadata=0x2/0x2␣
↪actions=resubmit(,220)
```

Pipeline changes

Current ACL pipeline:

Table	Match	Action
Dispatcher	metadata=service_id:ACL	write_metadata:(elan_id=ELAN|VPN_ID, service_id=NEXT), goto_table:ACL0|ACL1
.		
ACL0 (239)	ct_state=+trk	ct(table=ACL1)
ACL0 (239)	(TABLE-MISS)	goto_table:ACL1
.		
ACL1 (210/240)	(anti-spoofing filters)	goto_table:ACL2
ACL1 (210/240)	(TABLE-MISS)	drop
.		
ACL2 (211/241)	metadata=ELAN|VPN_ID, ip_src/dst=VM1_IP	write_metadata:(remote_acl=id), goto_table:ACL3
ACL2 (211/241)	metadata=ELAN|VPN_ID, ip_src/dst=VM2_IP	write_metadata:(remote_acl=id), goto_table:ACL3
. . .		
ACL2 (211/241)	(TABLE-MISS)	goto_table:ACL3
.		
ACL3 (212/242)	reg6=lport, ip|ipv6, ct_mark=0x1	(set_field:0x0->ct_mark), goto_table:ACL4
ACL3 (212/242)	(TABLE-MISS)	goto_table:ACL4
.		
ACL4 (213/243)	ct_state=-new+est-rel-inv+trk, ct_mark=0x1	resubmit(,DISPATCHER)
ACL4 (213/243)	ct_state=-new-est+rel-inv+trk, ct_mark=0x1	resubmit(,DISPATCHER)
ACL4 (213/243)	reg6=lport, ct_state=+inv+trk	drop
ACL4 (213/243)	reg6=lport, ct_state=+trk, <acl_rule>	set_field:0x1>ct_mark, resubmit(,DISPATCHER) (X)
ACL4 (213/243)	reg6=lport+remote_acl, ct_state=+trk, <acl_rule>	set_field:0x1>ct_mark, resubmit(,DISPATCHER) (XX)
ACL4 (213/243)	reg6=lport, ct_state=+trk, ip_src/dst=VM1_IP, <acl_rule>	set_field:0x1>ct_mark, resubmit(,DISPATCHER) (XXX)
ACL4 (213/243)	reg6=lport, ct_state=+trk, ip_src/dst=VM2_IP, <acl_rule>	set_field:0x1>ct_mark, resubmit(,DISPATCHER) (XXX)
ACL4 (213/243)	reg6=lport, ct_state=+trk	drop
. . .		
ACL4 (213/243)	(TABLE-MISS)	drop

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

(XX) These are the 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.

Proposed ACL pipeline:

Table	Match	Action
Dispatcher	metadata=service_id:ACL	write_metadata:(service_id=NEXT), goto_table:ACL0|ACL
.		
ACL0 (239)	ct_state=+trk	ct(table=ACL1)
ACL0 (239)	(TABLE-MISS)	goto_table:ACL1
.		
ACL1 (210/240)	(anti-spoofing filters)	goto_table:ACL2
ACL1 (210/240)	(TABLE-MISS)	drop
.		
ACL2 (211/241)	UDP	write_metadata:CONST_0, goto_table:ACL3 (X)
ACL2 (211/241)	TCP	write_metadata:CONST_0, goto_table:ACL3 (X)
ACL2 (211/241)	ICMP	write_metadata:CONST_0, goto_table:ACL3 (X)
ACL2 (211/241)	ICMPv6	write_metadata:CONST_0, goto_table:ACL3 (X)
ACL2 (211/241)	(TABLE-MISS)	write_metadata:CONST_1, goto_table:ACL5 (XX)
.		
ACL3 (212/242)	metadata=lport1, ip|ipv6	ct(table=ACL4,zone=ELAN_ID)
ACL3 (212/242)	metadata=lport2, ip|ipv6	ct(table=ACL4,zone=ELAN_ID)
. . .		
ACL3 (212/242)	(TABLE-MISS)	drop
.		
ACL4 (213/243)	reg6=lport, ip|ipv6, ct_mark=0x1	(set_field:0x0->ct_mark), goto_table:ACL5
ACL4 (213/243)	(TABLE-MISS)	goto_table:ACL5
.		
ACL5 (214/244)	ct_state=-new+est-rel-inv+trk, ct_mark=0x1	resubmit(,DISPATCHER)
ACL5 (214/244)	ct_state=-new-est+rel-inv+trk, ct_mark=0x1	resubmit(,DISPATCHER)
ACL5 (214/244)	reg6=lport, ct_state=+inv+trk	drop
ACL5 (214/244)	reg6=lport1, pri=40, <acl_rule>	goto_table:ACL8 (XXX)
ACL5 (214/244)	reg6=lport1, pri=30, metadata=remote_acl1	drop (XXXX)
ACL5 (214/244)	reg6=lport1, pri=10,	write_metadata:(remote_acl1), goto_table:ACL6 (XXXX)
ACL5 (214/244)	reg6=lport2, pri=30, metadata=remote_acl3	drop (XXXXX)
ACL5 (214/244)	reg6=lport2, pri=20, metadata=remote_acl2	write_metadata:(remote_acl3), goto_table:ACL6 (XXXXX)
ACL5 (214/244)	reg6=lport2, pri=20, metadata=remote_acl1	write_metadata:(remote_acl2), goto_table:ACL6 (XXXXX)
ACL5 (214/244)	reg6=lport2, pri=10	write_metadata:(remote_acl1), goto_table:ACL6 (XXXXX)
ACL5 (214/244)	reg6=lport1, pri=5	drop
ACL5 (214/244)	reg6=lport2, pri=5	drop
. . .		
ACL5 (214/244)	(TABLE-MISS)	drop
.		
ACL6 (215/245)	reg6=lport, pri=20, metadata=remote_acl1, <rule1>	goto_table:ACL7
ACL6 (215/245)	reg6=lport, pri=20, metadata=remote_acl1, <rule2>	goto_table:ACL7
ACL6 (215/245)	reg6=lport, pri=20, metadata=remote_acl2, <rule1>	goto_table:ACL7
ACL6 (215/245)	reg6=lport, pri=20, metadata=remote_acl3, <rule1>	goto_table:ACL7
. . .		
ACL6 (215/245)	(TABLE-MISS)	resubmit(,ACL5)
.		
ACL7 (216/246)	metadata=remote_acl1, ip_src/dst=VM1_IP	goto_table:ACL8
ACL7 (216/246)	metadata=remote_acl1, ip_src/dst=VM2_IP	goto_table:ACL8
ACL7 (216/246)	metadata=remote_acl2, ip_src/dst=VM3_IP	goto_table:ACL8
ACL7 (216/246)	metadata=remote_acl3, ip_src/dst=VM4_IP	goto_table:ACL8

Table 1 – continued from previous page

Table	Match	Action
...		
ACL7 (216/246)	(TABLE-MISS)	resubmit(,ACL5)
.		
ACL8 (217/247)	reg6=lport, metadata=CONST_0	ct(commit,zone=ELAN_ID,exec(set_field:0x1->ct_mark)),
ACL8 (217/247)	reg6=lport, metadata=CONST_1	resubmit(,DISPATCHER) (XX)
...		
ACL8 (217/247)	(TABLE-MISS)	drop

CONST_0 Constant referring to conntrack supported traffic. eg: 0x0/0x2
CONST_1 Constant referring to non-conntrack supported traffic. eg: 0x2/0x2

(X) These are conntrack supported traffic.
(XX) These are non-conntrack supported traffic.
(XXX) These are the rules not configured with any remote SG.
(XXXX) These are the cases where all the rules have common (single) remote SG.
(XXXXX) These are rules having different remote SG.

Note: Observe the sample priorities in ACL5 table for different cases.

Sample flows:

```
cookie=0x6900000,table=239,priority=62020,ct_state=+trk,ip actions=ct(table=240)
cookie=0x6900000,table=239,priority=62020,ct_state=+trk,ipv6 actions=ct(table=240)
cookie=0x6900000,table=239,priority=61010 actions=goto_table:240

cookie=0x6900000,table=240,priority=63010,arp,reg6=0x200/0xffffff00 actions=resubmit(,
↪220)
cookie=0x6900000,table=240,priority=61010,reg6=0x200/0xffffff00,ip,dl_
↪dst=fa:16:3e:5c:42:d5,nw_dst=10.10.10.12 actions=goto_table:242

cookie=0x6900000,table=241,priority=61010,tcp6 actions=write_metadata:0x0/0x2,goto_
↪table:242
cookie=0x6900000,table=241,priority=61010,udp6 actions=write_metadata:0x0/0x2,goto_
↪table:242
cookie=0x6900000,table=241,priority=61010,tcp actions=write_metadata:0x0/0x2,goto_
↪table:242
cookie=0x6900000,table=241,priority=61010,udp actions=write_metadata:0x0/0x2,goto_
↪table:242
cookie=0x6900000,table=241,priority=61010,icmp6 actions=write_metadata:0x0/0x2,goto_
↪table:242
cookie=0x6900000,table=241,priority=61010,icmp actions=write_metadata:0x0/0x2,goto_
↪table:242
cookie=0x6900000,table=241,priority=0 actions=write_metadata:0x2/0x2,goto_table:244

cookie=0x6900000,table=242,priority=61010,ip,reg6=0x200/0xffffff00␣
↪actions=ct(table=243,zone=5002)
cookie=0x6900000,table=242,priority=0 actions=drop

cookie=0x6900000,table=243,priority=0 actions=goto_table:244
```

(continues on next page)

(continued from previous page)

```
cookie=0x6900000,table=244,priority=62060,ct_state=-new+est-rel-inv+trk,ct_mark=0x1␣
↪actions=resubmit(,220)
cookie=0x6900000,table=244,priority=62060,ct_state=-new-est+rel-inv+trk,ct_mark=0x1␣
↪actions=resubmit(,220)
cookie=0x6900000,table=244,priority=62050,reg6=0x200/0xffffff00,ct_state=+inv+trk␣
↪actions=drop
cookie=0x6900000,table=244,priority=62050,reg6=0x300/0xffffff00,ct_state=+inv+trk␣
↪actions=drop
cookie=0x6900000,table=244,priority=62040,tcp,reg6=0x200/0xffffff00 actions=goto_
↪table:247
cookie=0x6900000,table=244,priority=62030,reg6=0x200/0xffffff00,metadata=0x100/
↪0xffffff actions=drop
cookie=0x6900000,table=244,priority=62010,reg6=0x200/0xffffff00 actions=write_
↪metadata:0x100/0xffffff,goto_table:245
cookie=0x6900000,table=244,priority=62030,reg6=0x300/0xffffff00,metadata=0x300/
↪0xffffff actions=drop
cookie=0x6900000,table=244,priority=62020,reg6=0x300/0xffffff00,metadata=0x200/
↪0xffffff actions=write_metadata:0x300/0xffffff,goto_table:245
cookie=0x6900000,table=244,priority=62020,reg6=0x300/0xffffff00,metadata=0x100/
↪0xffffff actions=write_metadata:0x200/0xffffff,goto_table:245
cookie=0x6900000,table=244,priority=62010,reg6=0x300/0xffffff00 actions=write_
↪metadata:0x100/0xffffff,goto_table:245
cookie=0x6900000,table=244,priority=0 actions=drop

cookie=0x6900000,table=245,priority=61010,tcp,reg6=0x300/0xffffff00,metadata=0x100/
↪0xffffff actions=goto_table:246
cookie=0x6900000,table=245,priority=61010,udp,reg6=0x300/0xffffff00,metadata=0x100/
↪0xffffff actions=goto_table:246
cookie=0x6900000,table=245,priority=61010,icmp,reg6=0x300/0xffffff00,metadata=0x200/
↪0xffffff actions=goto_table:246
cookie=0x6900000,table=245,priority=0 actions=resubmit(,244)

cookie=0x6900000,table=246,priority=61010,ip,metadata=0x100/0xffffff,nw_src=10.10.10.
↪6 actions=goto_table:247
cookie=0x6900000,table=246,priority=61010,ip,metadata=0x100/0xffffff,nw_src=10.10.10.
↪12 actions=goto_table:247
cookie=0x6900000,table=246,priority=61010,ip,metadata=0x200/0xffffff,nw_src=10.10.10.
↪4 actions=goto_table:247
cookie=0x6900000,table=246,priority=61010,ip,metadata=0x300/0xffffff,nw_src=10.10.10.
↪8 actions=goto_table:247
cookie=0x6900000,table=246,priority=0 actions=resubmit(,244)

cookie=0x6900000,table=247,priority=61010,ip,reg6=0x200/0xffffff00,metadata=0x0/0x2␣
↪actions=ct(commit,zone=5002,exec(set_field:0x1->ct_mark)),resubmit(,220)
cookie=0x6900000,table=247,priority=61010,ipv6,reg6=0x200/0xffffff00,metadata=0x0/0x2␣
↪actions=ct(commit,zone=5002,exec(set_field:0x1->ct_mark)),resubmit(,220)
cookie=0x6900000,table=247,priority=61010,reg6=0x200/0xffffff00,metadata=0x2/0x2␣
↪actions=resubmit(,220)
cookie=0x6900000,table=247,priority=0 actions=drop
```

Yang changes

Below yang container is used to support overlapping IP addresses while processing remote ACLs. This would remove dependency on ID manager which was used to generate flow priorities. With the yang approach, having a unique priority for each flow in ACL filter table is not required anymore.

```
container acl-ports-lookup {
    config false;
    description "Container used to manage list of ports per ACL based on
        port's IP address/prefix (including IP address/prefix specified in
        allowed-address-pair)";

    list acl-ports-by-ip {
        key "acl-name";
        description "Refers to an ACL which are associated with list of
            ports filtered based on IP address/prefix.";

        leaf acl-name {
            type string;
            description "ACL name.";
        }
        list acl-ip-prefixes {
            key "ip-prefix";
            description "IP Prefixes and Allowed-Address-Pairs owned by
                ports where all such ports enforce the same ACL identified
                by acl-name";

            leaf ip-prefix {
                type ip-prefix-or-address;
                description "IP address/prefix";
            }
            list port-ids {
                key "port-id";
                description "Contains a list of ports that are enforcing
                    the same ACL identified by acl-name.";
                leaf port-id {
                    type string;
                    description "Port UUID string";
                }
            }
        }
    }
}
```

Configuration impact

None

Clustering considerations

With the proposed changes, ACL should work in cluster environment seamlessly as it's with the current ACL feature.

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

There will be improvements in scale and performance due to below reasons:

- Optimization for Remote ACL by reducing number of flows even for ports having multiple ACLs.
- Removal of ID manager dependency for generation of flow priorities configured in ACL filter tables.

Targeted Release

Oxygen

Alternatives

Currently, conntrack supports or recognizes only those IP protocols which carry point-to-point datapath information. Conntrack should support all the other IP protocols (VRRP, OSPF, etc) as well so that they are NOT classified as INVALID.

This approach was not selected as

- The support has to be provided in conntrack module. Or until it is supported in conntrack, the proposed change is required in ACL module.
- List of protocols to be supported in conntrack might need continuous updates or it has to be handled in generic way.

Usage

Features to Install

odl-netvirt-openstack

REST API

No new REST API is being added.

CLI

No new CLI being added.

Implementation

Assignee(s)

Primary assignee: Somashekar Byrappa <somashekar.b@altencalsoftlabs.com>

Other contributors: Shashidhar R <shashidharr@altencalsoftlabs.com>

Work Items

- Support protocols like OSPF, VRRP etc in ACL that are not supported by conntrack in stateful mode.
- Handle overlapping IP addresses while processing remote ACLs by making use of new yang container (acl-ports-lookup).

Dependencies

No new dependencies.

Testing

Unit Tests

Following test cases will need to be added/expanded

1. Verify ACL functionality with VRRP, OSPF protcols
2. Verify ACL functionality with other IP protocols not supported by conntrack
3. Verify ACL with ports having overlapping IP addresses.
4. Verify ACL with ports having single SG.
5. Verify ACL with ports having multiple SGs.

Also, existing unit tests have to be updated to include new pipeline/flow changes.

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 functionality with VRRP, OSPF protcols
2. Verify ACL functionality with other IP protocols not supported by conntrack
3. Verify ACL with ports having overlapping IP addresses.
4. Verify ACL with ports having single SG.
5. Verify ACL with ports having multiple SGs.

Documentation Impact

None

References

Table of Contents

Conntrack Based SNAT

https://git.opendaylight.org/gerrit/#/q/topic:snat_conntrack

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

Problem description

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

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

Use Cases

The following use case will be realized by the implementation

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

Proposed change

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

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

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

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

Pipeline changes

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

Outbound NAPT

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

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

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

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

Sample Flows

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

Inbound NAPT

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

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

Sample Flows

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

Yang changes

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

```
typedef nx-action-nat-range-present {
    type enumeration {
        enum NX_NAT_RANGE_IPV4_MIN {
            value 1;
            description "IPV4 minimum value is present";
        }
        enum NX_NAT_RANGE_IPV4_MAX {
            value 2;
            description "IPV4 maximum value is present";
        }
        enum NX_NAT_RANGE_IPV6_MIN {
            value 4;
            description "IPV6 minimum value is present in range";
        }
        enum NX_NAT_RANGE_IPV6_MAX {
            value 8;
            description "IPV6 maximum value is present in range";
        }
        enum NX_NAT_RANGE_PROTO_MIN {
            value 16;
            description "Port minimum value is present in range";
        }
        enum NX_NAT_RANGE_PROTO_MAX {
            value 32;
            description "Port maximum value is present in range";
        }
    }
 }
```

(continues on next page)

(continued from previous page)

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

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

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

(continues on next page)

(continued from previous page)

```
            container nx-action-nat {
                leaf flags {
                    type uint16;
                }
                leaf range_present {
                    type uint16;
                }
                leaf ip-address-min {
                    type inet:ip-address;
                }
                leaf ip-address-max {
                    type inet:ip-address;
                }
                leaf port-min {
                    type uint16;
                }
                leaf port-max {
                    type uint16;
                }
            }
        }
    }
}
```

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

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

Configuration impact

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

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

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

Clustering considerations

NA

Other Infra considerations

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

Security considerations

NA

Scale and Performance Impact

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

Targeted Release

Carbon

Alternatives

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

Usage

Create External Network

Create an external flat network and subnet

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

Create Internal Network

Create an internal n/w and subnet

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

Create Router

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

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

Features to Install

odl-netvirt-openstack

REST API

NA

CLI

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

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

Implementation

Assignee(s)

Aswin Suryanarayanan <asuryana@redhat.com>

Work Items

https://trello.com/c/DMLsrLfq/9-snat-decentralized-ovs-nat-based

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

Dependencies

NA

Testing

Unit Tests

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

Integration Tests

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

CSIT

Run the CSIT with conntrack based SNAT configured.

Documentation Impact

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

References

Table of Contents

Cross site connectivity with federation service

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

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

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

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

Problem description

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

Use Cases

This feature will cover the following use cases:

L2 switching use cases

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

L3 forwarding use cases

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

Proposed change

For Carbon release, cross-site connectivity will be based on the current HPE downstream federation plugin codebase. 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";
```

(continues on next page)

(continued from previous page)

```
    }

    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";
```

(continues on next page)

(continued from previous page)

```
}

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 {
```

(continues on next page)

(continued from previous page)

```
            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;
  }
```

(continues on next page)

(continued from previous page)

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

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

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

Federation Plugin routed RPC Yang

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

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

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

   import ietf-yang-types {
       prefix yang;
   }

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

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

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

(continues on next page)

(continued from previous page)

```
            }

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

Configuration impact

None.

Clustering considerations

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

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

None

Targeted Release

Carbon

Alternatives

None

Usage

Features to Install

odl-netvirt-federation

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

REST API

Connecting neutron networks from remote sites

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

Sample JSON data

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

CLI

None.

Implementation

Assignee(s)

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

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

Shlomi Alfasi <shlomi.alfasi@hpe.com>

Yair Zinger <yair.zinger@hpe.com>

Work Items

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

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

Dependencies

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

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

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

Testing

Unit Tests

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

Integration Tests

None

CSIT

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

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

1 Install odl-netvirt-federation feature

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

2 Connect sites A,B

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

3 Connect site C to A,B

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

4 Disconnect site C from A,B

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

5 Disconnect sites A,B

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

6 Federation cluster test

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

Documentation Impact

None.

References

[1] OpenDaylight Documentation Guide

[2] Federation project

[3] Federation service API

[4] Support of VxLAN based connectivity across Datacenters

[5] VNI based L2 switching, L3 forwarding and NATing

[6] Cross site manager presentation ODL Summit 2016

Table of Contents

DHCP Server Dynamic Allocation Pool

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

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

The feature supports IPv4 only.

Problem description

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

Use Cases

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

Proposed change

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

The allocation pool DHCP server will support:

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

Pipeline changes

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

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

Yang changes

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

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

Listing 28: dhcp_allocation_pool.yang

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

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

            list allocation-instance {
                key "mac";
                leaf mac {
                    description "requesting mac";
                    type yang:phys-address;
                }
                leaf allocated-ip {
                    description "allocated ip address";
                    type inet:ip-address;
                }
            }
        }
        list allocation-pool {
            key "subnet";
            leaf subnet {
                description "subnet for the dhcp to allocate ip addresses";
                type inet:ip-prefix;
            }
            leaf allocate-from {
                description "low allocation limit";
                type inet:ip-address;
```

(continues on next page)

(continued from previous page)

```
            }
            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 29: 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>
```

(continues on next page)

(continued from previous page)

```
  <dhcp-dynamic-allocation-pool-enabled>false</dhcp-dynamic-allocation-pool-enabled>
</dhcpservice-config>
```

Clustering considerations

Support clustering.

Other Infra considerations

None.

Security considerations

None.

Scale and Performance Impact

None.

Targeted Release

Carbon.

Alternatives

Implement and maintain an external DHCP server.

Usage

Features to Install

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

REST API

Introducing a new REST API for the feature

Dynamic allocation pool

URL: /config/dhcp_allocation_pool:dhcp_allocation_pool/

Sample JSON data

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

Static address allocation

URL: /config/dhcp_allocation_pool:dhcp_allocation_pool/

Sample JSON data

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

CLI

None.

Implementation

Assignee(s)

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

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

Work Items

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

Dependencies

None.

Testing

Unit Tests

N.A.

Integration Tests

N.A.

CSIT

N.A.

Documentation Impact

??

References

Table of Contents

Discovery of directly connected PNFs in Flat/VLAN provider networks

https://git.opendaylight.org/gerrit/#/q/topic:directly_connected_pnf_discovery

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

Problem description

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

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

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

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

Subnet-Route

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

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

Current limitations of Subnet-Route feature:

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

Aliveness monitor

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

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

Use Cases

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

Proposed change

Subnet-route

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

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

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

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

Communication between VMs and PNFs in different tenant networks.

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

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

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

ARP messages

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

Pipeline changes

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

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

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

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

SNAT table (28) `match: vpn-id=ext-subnet-id,src-ip=fip set src-mac=fip-mac` =>
FIB table (21) `match: vpn-id=ext-subnet-id, dst-ip=ext-subnet-ip` =>
Subnet Route table (22): => Output to Controller

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

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

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

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

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

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

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id` =>
Pre SNAT table (26) `match: vpn-id=router-id` =>
Outbound NAPT table (46) `match: src-ip=vm-ip,port=int-port set src-ip=router-gw-ip,vpn-id=router-gw-subnet-id,port=ext-port` =>

NAPT PFIB tabl (47) `match: vpn-id=router-gw-subnet-id` =>
FIB table (21) `match: vpn-id=ext-subnet-id, dst-ip=ext-subnet-ip` =>
Subnet Route table (22) => Output to Controller

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

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id` =>
Pre SNAT table (26) `match: vpn-id=router-id` =>
Outbound NAPT table (46) `match: vpn-id=router-id TBD set vpn-id=external-net-id` =>
NAPT PFIB table (47) `match: vpn-id=external-net-id` =>
FIB table (21) `match: vpn-id=ext-network-id, dst-ip=pnf-ip set dst-mac=pnf-mac, reg6=provider-lport-tag` =>
Egress table (220) output to provider port

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

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

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) `match: vpn-id=router-id,dst-mac=router-interface-mac` =>
FIB table (21) `match: vpn-id=router-id dst-ip=subnet-ip` =>
Subnet Route table (22): => Output to Controller

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

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

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

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

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

Yang changes

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

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

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

Configuration impact

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

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

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

Targeted Release

Carbon

Alternatives

None

Usage

Create external network with a subnet

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

Create internal networks with subnets

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

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

This will allow communication with PNFs in provider network

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

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

This will allow East/West communication between VMs and PNFs

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

Features to Install

odl-netvirt-openstack

REST API

CLI

Implementation

Assignee(s)

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

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

Work Items

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

Dependencies

This feature depends on hairpinning feature [2]

Testing

Unit Tests

Unit tests will be added for the new functionality

Integration Tests

CSIT

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

Documentation Impact

References

[1] https://docs.google.com/presentation/d/1ByvEQXUtIyH-H7Bin6OBJNrHjOv-3hpHYzU6Sf6hDbA/edit#slide=id.g11657174d1_0_31 [2] http://docs.opendaylight.org/en/latest/submodules/netvirt/docs/specs/hairpinning-flat-vlan.html

Table of Contents

ECMP Support for BGP based L3VPN

https://git.opendaylight.org/gerrit/#/q/topic:l3vpn_ecmp

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

Problem description

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

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

Use Cases

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

High-Level Components:

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

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

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

Proposed change

The following components within OpenDaylight Controller needs to be enhanced:

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

Pipeline changes

Local FIB entry/Nexthop Group programming:

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

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

```
cookie=0x8000003, duration=46.020s, table=21, n_packets=0, n_bytes=0, priority=34,ip,
 ↪metadata=0x222e4/0xfffffffe, nw_dst=100.0.0.0/24 actions=write_actions(group:150002)
group_id=150002,type=select,bucket=weight:50,actions=group:150001,bucket=weight:50,
 ↪actions=group:150000
group_id=150001,type=all,bucket=actions=set_field:fa:16:3e:34:ff:58->eth_dst,
 ↪load:0x200->NXM_NX_REG6[],resubmit(,220)
group_id=150000,type=all,bucket=actions=set_field:fa:16:3e:eb:61:39->eth_dst,
 ↪load:0x100->NXM_NX_REG6[],resubmit(,220)
```

Remote FIB entry/Nexthop Group programming:

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

  on remote compute node,

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

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

  on local compute node,

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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 30: 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 31: 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 32: 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 33: odl-l3vpn.yang

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

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

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

(continues on next page)

(continued from previous page)

```
                }
            }
        }
    }
```

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 34: 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 35: odl-fib.yang

```
    container l3vpn-lb-nexthops {
        config false;
```

(continues on next page)

(continued from previous page)

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

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

ECMP forwarding through multiple Compute Node and VMs

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

ECMP forwarding for dispersed VMs

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

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

ECMP forwarding for co-located VMs

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

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

ECMP forwarding through two DC-Gateways

The current ITM implementation provides support for creating multiple GRE tunnels for the provided list of DC-GW IP addresses from compute node. This should help in creating corresponding load balancing group whenever Quagga BGP is advertising two routes on same IP prefix pointing to multiple DC GWs. The group id of this load balancing group can be derived from sorted order of `DC GW TEP IP addresses` with the following format dc_gw_tep_ip _address_1: dc_gw_tep_ip_address_2. This will be useful when multiple external IP prefixes share the same next hops. The load balancing next hop group buckets is programmed according to sorted remote end point DC-Gateway IP address. The support of action move:NXM_NX_REG0(1) -> `MPLS label` is not supported in ODL openflowplugin. It has to be implemented. Since there are two DC gateways present for the data center, it is possible that multiple equal cost routes are supplied to ODL by Quagga BGP like Fig 2. The current Quagga BGP doesn't have multipath support and it will be done. When Quagga BGP announces route with multiple paths, then it is ODL responsibility to program Fib entries in all compute nodes where VPN instance blueprint is present, so that traffic can be load balanced between these two DC gateways. It requires changes in existing odl-fib.yang model (like below) to support multiple routes for same destination IP prefix.

BGPManager should be able to create vrf entry for the advertised IP prefix with multiple route paths. VrfEntryListener listens to DCN on these vrf entries and program Fib entries (21) based on number route paths available for given IP prefix. For the given (external) destination IP prefix, if there is only one route path exists, use the existing approach to program FIB table flow entry matches on (vpnid, ipv4_dst) and actions with push `MPLS label` and output to gre tunnel port. For the given (external) destination IP prefix, if there are two route paths exist, then retrieve next hop ip address from routes list in the same sorted order (i.e. using same logic which is used to create buckets for load balancing next hop group for DC- Gateway IP addresses), then program FIB table flow entry with an instruction like Fig 3. It should have two set field actions where first action sets `MPLS label` to NX_REG0 for first sorted DC-GW IP address and second action sets `MPLS label` to NX_REG1 for the second sorted DC-GW IP address. When more than two DC Gateways are used, then more number of NXM Registries have to be used to push appropriate `MPLS label` before sending it to next hop group. It needs operational DS container to have mapping between DC Gateway IP address and NXM_REG. When one of the route is withdrawn for the IP prefix, then modify the FIB table flow entry with with push `MPLS label` and output to the available gre tunnel port.

ECMP for Intra-DC L3VPN communication

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

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

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

To allocate group id for this load balancing next hop, the same destination IP prefix is used as the group key. The remote FIB table flow should point to this next hop group after writing prefix `label` into tunnel_id. The bucket weight of remote next hop is adjusted according to number of VMs associated to given extra route and on which compute node the VMs are connected. For example, two compute node having one VM each, then bucket weight is

50 each. One compute node having two VMs and another compute node having one VM, then bucket weight is 66 and 34 each. The hop-count property in vrfEntry data store helps to decide what is the bucket weight for each bucket.

ECMP Path decision based on Internal/External Tunnel Monitoring

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

GRE tunnel state handling

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

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

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

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

VxLAN tunnel state handling

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

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

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

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

Assumptions

The support for action move:NXM_NX_REG0(1) -> `MPLS label` is already available in Compute Node.

Reboot Scenarios

This feature support all the following Reboot Scenarios for EVPN:

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

Clustering considerations

The feature should operate in ODL Clustered environment reliably.

Other Infra considerations

N.A.

Security considerations

N.A.

Scale and Performance Impact

Not covered by this Design Document.

Targeted Release

Carbon.

Alternatives

Alternatives considered and why they were not selected.

Usage

Features to Install

This feature doesn't add any new karaf feature.

REST API

Implementation

Assignee(s)

Primary assignee(s):

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

Other contributors:

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

Work Items

The Trello cards have already been raised for this feature under l3vpn_ecmp.

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

Dependencies

Quagga BGP multipath support and APIs. This is needed to support when two DC-GW advertises routes for same external prefix with different route `labels` GRE tunnel monitoring. This is need to implement ECMP forwarding based on MPLSoGRE tunnel state. Support for action move:NXM_NX_REG0(1) -> `MPLS label` in ODL openflowplugin

Testing

Capture details of testing that will need to be added.

Unit Tests

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

Integration Tests

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

CSIT

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

Documentation Impact

This will require changes to User Guide and Developer Guide.

References

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

Table of Contents

ELAN Service Recovery Test Plan

Test plan for testing service recovery manager functionalities.

Test Setup

Test setup consists of ODL with *odl-netvirt-openstack* feature installed and minimum two DPNs connected to ODL over OVSDB and OpenflowPlugin.

Hardware Requirements

N.A

Software Requirements

Openstack queens + OVS 2.8

Test Suite Requirements

Test Suite Bringup

Following steps are followed at the beginning of test suite:

- Bring up controller with *odl-netvirt-openstack* feature installed
- Bring up minimum two DPNs with tunnel between them
- Create network
- Create subnet
- Create at least two VMs in each DPN
- Verify table 50/51 flows in both DPNs

Test Suite Cleanup

Following steps are followed at the end of test suite:

- Delete VMs
- Delete subnet
- Delete network

Debugging

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

Test Cases

ELAN Service Recovery

Verify SRM by recovering ELAN Service.

Test Steps and Pass Criteria

1. Delete table 50/51 flow(s) corresponding to MAC address(es) of VM(s) (try deleting multiple flows) in any of the DPNs manually or via REST
2. Verify if table 50/51 flow(s) is/are deleted in both controller and OVS.
3. Login to karaf and use elan service recovery CLI
4. Verify if corresponding table flow(s) is/are recovered in on both controller and ovs

ELAN Interface Recovery

Verify SRM by recovering ELAN Interface.

Test Steps and Pass Criteria

1. Delete table 50/51 flow corresponding to MAC address of any of the VMs.
2. Verify if table 50/51 flow is deleted in both controller and OVS.
3. Login to karaf and use elan interface recovery CLI
4. Verify if corresponding table flow is recovered in both controller and OVS.

Implementation

Assignee(s)

Primary assignee: Swati Niture (swati.udhavrao.niture@ericsson.com)

Other contributors: N.A.

Work Items

N.A.

References

http://docs.opendaylight.org/en/latest/submodules/genius/docs/specs/service-recovery.html#srm-operations

Table of Contents

Element Counters

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

This feature depends on the Netvirt statistics feature.

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

Problem description

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

Use Cases

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

Proposed change

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

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

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

Pipeline changes

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

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

VM A Outgoing Traffic (vm interface)

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

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

VM A Incoming Traffic (vm interface)

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

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

Assuming we want statistics on VM A incoming TCP traffic.

VM A Outgoing Traffic (vm interface)

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

Assuming we want statistics on VM A outgoing UDP traffic.

VM A Incoming traffic (vm interface)

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

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

VM A Incoming traffic (vm interface)

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

Yang changes

Netvirt Statistics module will be enhanced with the following RPC:

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

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

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

(continues on next page)

(continued from previous page)

```
                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;
        }
```

(continues on next page)

(continued from previous page)

```
        container outgoingTraffic {
            uses elementRequestData;
        }
        uses filters;
    }
    output {
        leaf incomingTrafficHandler {
            type string;
        }
        leaf outcoingTrafficHandler {
            type string;
        }
    }
}

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

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

Configuration impact

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

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

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

Targeted Release

Carbon

Alternatives

None

Usage

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

Run the following to get interface ids:

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

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

Acquire counter request handler:

Listing 36: 10.0.77.135:8181/restconf/operations/statistics-plugin:acquireElementCountersRequestHandler

```
{
    "input": {
        "portId":"4073b4fe-a3d5-47c0-b37d-4fb9db4be9b1",
        "incomingTraffic": {
            "filters": {
```

(continues on next page)

(continued from previous page)

```
                "ipFilter": {
                    "ip":"1.1.3.9"
                }
            }
        }
    }
}
```

Release handler:

Listing 37: 10.0.77.135:8181/restconf/operations/statistics-plugin:releaseElementCountersRequestHandler

```
{
    "input": {
        "handler":"1"
    }
}
```

Get counters:

Listing 38: 10.0.77.135:8181/restconf/operations/statistics-plugin:getElementCountersByHandler

```
{
    "input": {
        "handler":"1"
    }
}
```

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",
```

(continues on next page)

(continued from previous page)


```
                                "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

Faster DC-GW Failure Detection

https://git.opendaylight.org/gerrit/#/q/topic:faster-dcgw-failure-detection

In L3VPN, local and external routes are exchanged using BGP protocol with DC-Gwy (Data Center Gateway). BGP control plane failure detection is based on hold-timer. BGP hold-timer is negotiated to be the lowest value between two neighbors. Once BGP hold-timer expires, peer can remove the routes received from its neighbor. Minimum value of hold-timer can be 3 seconds.

In some deployment scenarios like inter-DC (Data Center), BGP-VPN(internet connectivity), ... ODL will have Quagga BGP stack (qbgp). This qbgp will form neighborship with DC-Gwy and exchange routes based on 'route descriptor', 'route targets'. This will enable communication between "Hosts/VM's in DC" to external connectivity.

Problem description

In some existing deployment scenarios "OPNFV clustered environment", BGP neighborship establishment with DC-Gwy takes ~24 Seconds. Expectation to form neighborship is within 3-5 Seconds.

Use Cases

Redundant DC-Gwy with ECMP enabled in DPN(Data Path Node): DC-Gwy failover detection using existing control plane mechanism takes ~24 seconds in OPNFV-clustered environment.

why ~24 seconds?

In a clustered setup, BGP-EoS owner node gets rebooted and the node happens to be pacemaker seed node:

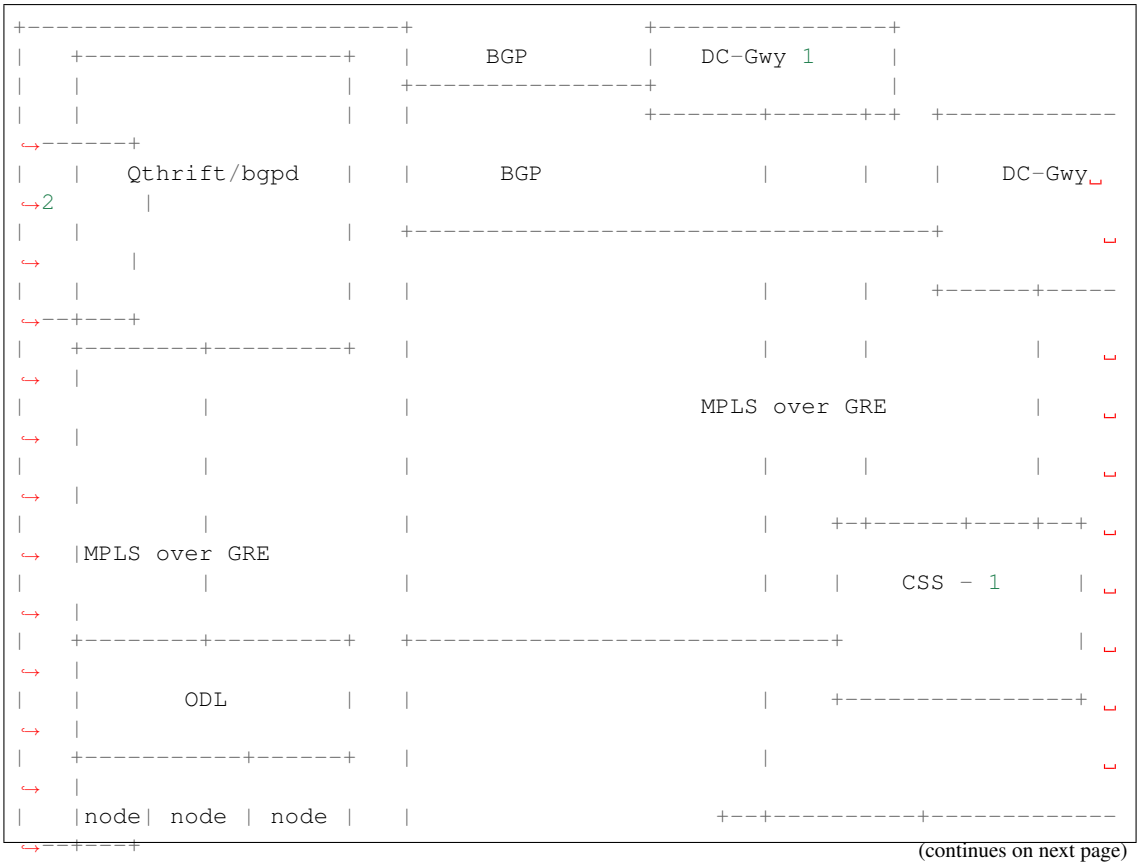
- To detect node failure and synchronize ~10 Seconds
- seed node election ~5Seconds.
- Time taken to elect the next EoS owner : ~5Seconds
- Once the cluster is formed and ODL/BgpManager opens thrift port (< 1 second)
- Configuring qthrift/bgpd ~2Seconds.

Note: above timings are based on observations, MAXIMUM time taken in clustered OPNFV environment. In summary : For a "ODL with qbgp in OPNFV(clustered) environment" to recover from a node reboot takes ~24 seconds. BGP keepalive, hold timers are negotiated and the least values between two neighbors is selected.

HOLD timer is used to purge the routes from DC-Gwy in case of neighbor down, it shall be configured beyond 24 seconds to avoid purging.

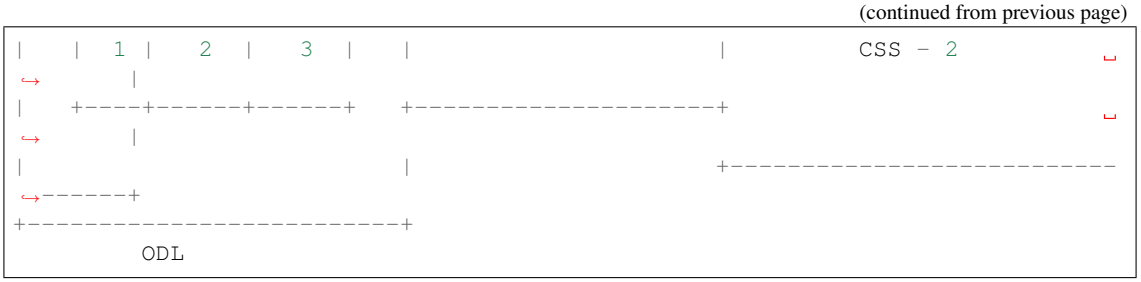
In a HA scenario of DC-Gwy, it is expected to switch the NEXT-HOP of a prefix to a available DC-Gwy. But, this was not achieved due to BGP HOLD timer value.

Setup Representation



(continues on next page)

(continued from previous page)

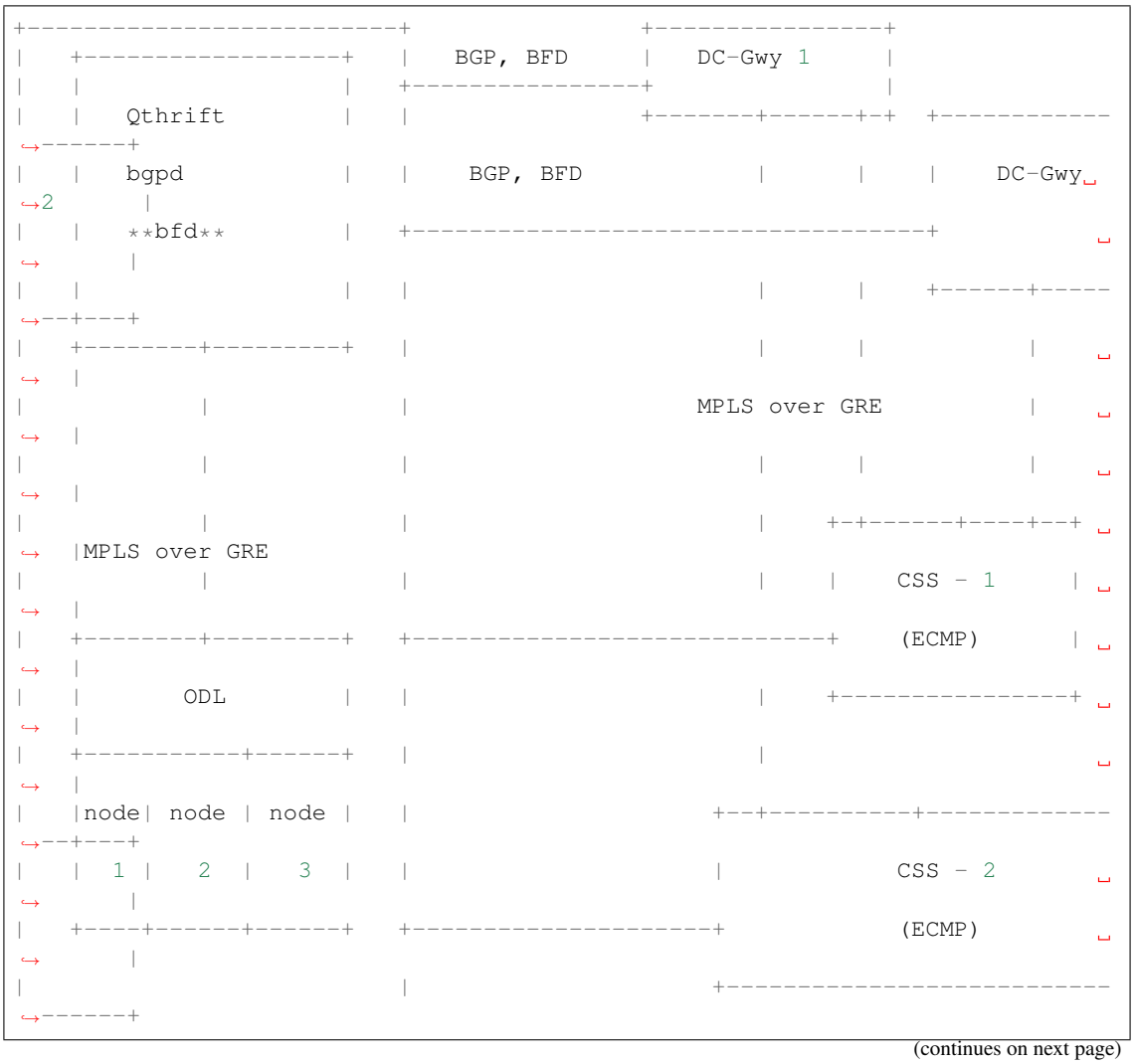


Proposed change

Proposal

Enable Control Path monitoring using BFD (between ODL and DC-Gwy).

Setup Representation



(continues on next page)

(continued from previous page)

```
+-------------------------+
          ODL
```

Using publicly available BFD implementation, BFD Daemon (BFDD) would be integrated with QBGP. In this approach, the BFDD would make use of BFD messages to detect the health of the connection with the DC-Gwy.

Upon connection failure, the BFDD would inform QBGP about the failure of the DC-Gwy. QBGP will terminate the neighborship and purge the routes received from failed DC-Gwy. Route withdrawl from QBGP results in, removing routes from dataplane.

BFDD =(1)=> BGPD =(2)=> ODL

(1) BFD tells neighbor DC-Gwy1 is lost

(2) BGP sends peer-down notification to ODL

(3) BGP withdraw prefixes learned from DC-Gwy1

The QBGP will ONLY depend on the BFDD to know the health status of DC-Gwy. So long as the DC-Gwy is marked as down by BFDD, the QBGP will REJECT connections from the DC-Gwy and will stop sending BGP OPEN messages to the DC-Gwy.

Similarly, when BFD Daemon detects that the DC-Gwy is back online, it informs QBGP about the same. The QBGP would now start accepting BGP connections from the DC-Gwy. It will also send out BGP Open messages to the DC-Gwy.

Since BFD monitoring interval can be set to 300-500ms, it would be possible to achieve sub-second DC-Gwy failure detection with BFD based monitoring.

Since the failure detection mechanism does NOT USE HOLD TIME, the QBGP failure recovery will be independent of DC-Gwy failure detection.

The proposal makes use of BFD in the control plane to detect the failure of the DC-Gwy. The Control Path between QBGP and DC-Gwy BGP Daemon is monitored using BFD. Failure of the control plane is used to purge the corresponding routes in the data plane.

With ECMP, alternate routes are preprogrammed in the data plane. Consequently, when the routes received from the failed DC-Gwy are purged, the flows automatically take the alternate path to reach their destination.

Below parameters are required for configuring BFD:

- Desired Min TX Interval: The QBGP must program this value to be equal to 1/3rd of the HOLD TIME value configured by default. By default, this value would be 60 seconds. The solution will provide a method to configure this value from the thrift interface.
- Required Min RX Interval: This would be configured to the value configured in bfdRxInterval
- bfdFailureDetectionThreshold: The bfdFailureDetectionThreshold will be used by the BFD implementation to identify the failure. When the number of lost packets exceed bfdFailureDetectionThreshold, the BFD protocol detects failure of the neighbour.
- bfdDebounceDown: This indicates the amount of time BFDD must wait to inform the QBGP about DC-Gwy failure. When BFDD detects DC-Gwy failure, it starts a timer with the value configured in bfdDebounceDown microseconds. Upon the expiry of the timer, the latest BFD state is checked. If the latest BFD state still indicates DC-Gwy failure, then the corresponding failure is reported to QBGP. If the latest BFD state indicates that DC-Gwy is restored, no message is sent to QBGP.
- bfdDebounceUp :This indicates the amount of time BFDD must wait to inform the QBGP about DC-Gwy Restoration. When BFDD detects DC-Gwy Restoration, it starts a timer with the value configured in bfdDebounceUp microseconds. Upon the expiry of the timer, the latest BFD state is checked. If the latest BFD

indicates DC-Gwy restoration, then the corresponding restoration is reported to QBGP. If the latest BFD state indicates DC-Gwy failure, no message is sent to QBGP.

Pipeline changes

None

Yang changes

A new yang file ebgp-bfd.yang, published to accomodate bfd parameters, which has bfd-config container as below.

Listing 39: ebgp-bfd.yang

```
container bfd-config {
    config        "true";

    leaf bfd-enabled {
        description  "is BFD enabled";
        type         boolean;
        default      false;
    }

    leaf detect-mult {
        type        uint32;
        default     3;
        description "The number of packets that have to be missed
                     in a row to declare the session to be down.";
    }
    leaf min-rx {
        type        uint32 {
                        range "50..50000";
        }
        default     500;
        description "The shortest interval, in milli-seconds, at
                     which this BFD session offers to receive
                     BFD control messages. Defaults to 500";
    }
    leaf min-tx {
        type        uint32 {
                        range "1000 .. 60000";
        }
        default     6000;
        description "The shortest interval, in milli-seconds,
                     at which this BFD session is willing to
                     transmit BFD control messages. Defaults
                     to 6000";
    }

    leaf multihop {
        type        boolean;
        default     true;
        description "Value of True indicates suppport for BFD multihop";
        config      "true";
    }
}
```

Changes will be needed in `ebgp.yang`.

- dc-gw TEP ip will be modified as a container

Listing 40: ebgp.yang

```
container dcgw-tep-list {
     list dcgw-tep {
          key          "dc-gw-ip";
          description  "mapping: DC-Gwy ip <> TEP ip";

          leaf dc-gw-ip {
              type string;
          }
          leaf-list tep-ips {
              type string;
          }
     }
}
```

Configuration impact

New BFD configuration parameters will be added with this feature.

- enable-bfd(default: true)
- min-rx (default: 500ms)
- monitor-window (default: 3)
- min-tx (default: 60 sec)
- failure-threshold (default: 100ms)
- success-threshold (default: 5 sec)
- AssociateTEPDCGW([tep-ip], DC-Gwy):

How will it impact existing deployments? There is NO impact on existing deployments.

Clustering considerations

There is no impact on clustering, as the bfdd/bgpd/zrpcd processes are supposed to run on only one node. If the bgp-controller-node goes down, it is the responsibility of CLUSTER environment to bringup on other nodes.

Other Infra considerations

Security considerations

none

Scale and Performance Impact

What are the potential scale and performance impacts of this change?

- There shall be no impact on performance.

Does it help improve scale and performance or make it worse?

- There shall be no impact on performance.

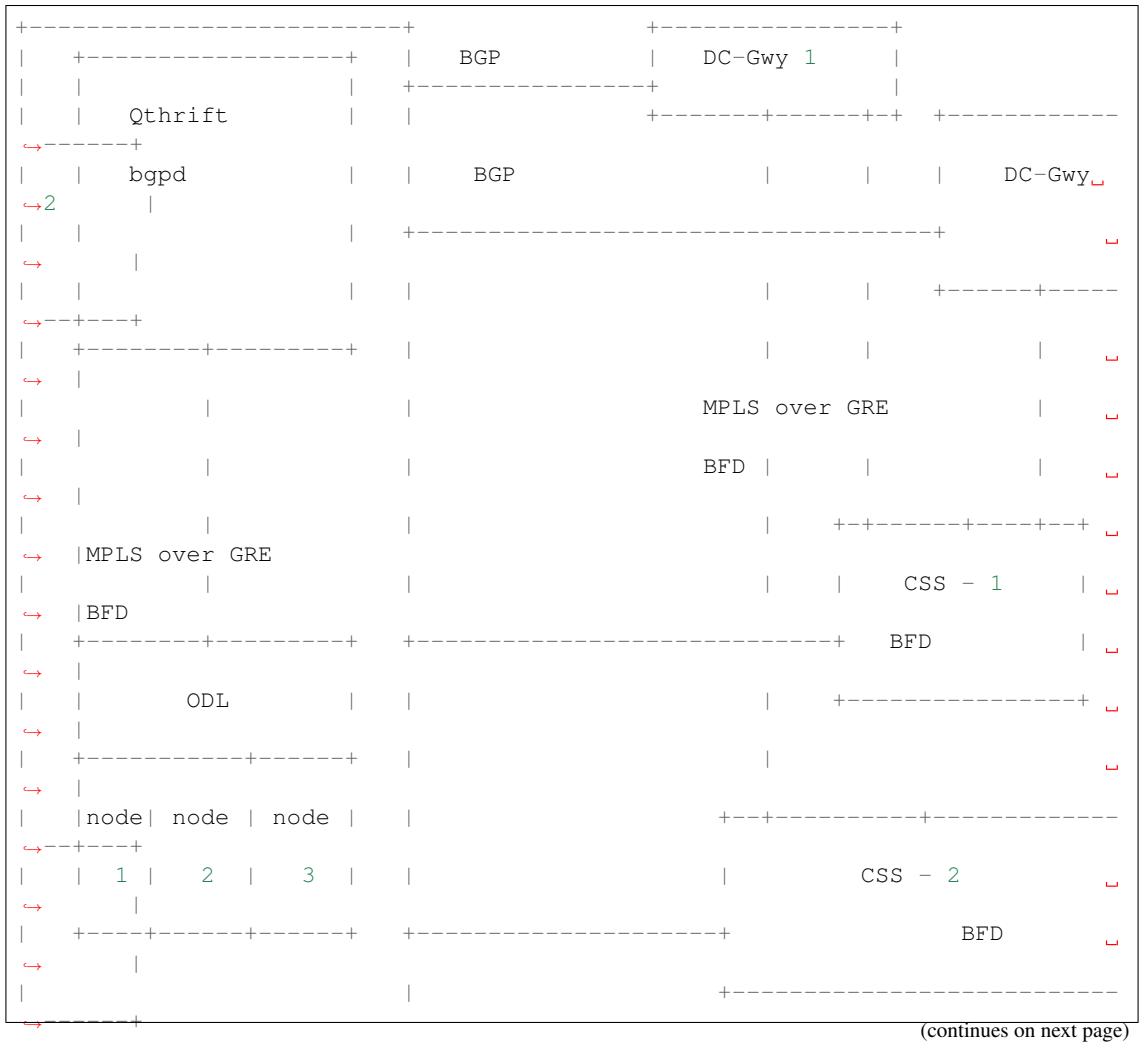
Targeted Release

What release is this feature targeted for? Oxygen/Fluorine.

Alternatives

Enable tunnel monitoring in Data Path using BFD (between CSS and DC-Gwy).

Setup Representation



(continues on next page)

(continued from previous page)

```
+-------------------------+
          ODL
```

This was not being implemented, as most of the DC-gwy's do not support BFD monitoring on MPLS/GRE tunnels.

Usage

As described in diagram, this feature is mainly to "switchover traffic to surviving DC-Gwy, in case of a DC-Gwy failure" and to reduce impact on Data Path.

Features to Install

odl-netvirt-openstack package : qthrift (with bfdd, bgpd)

REST API

will be added, when we start with implementation.

CLI

Yes, new CLI to configure bfdd (along with REST).

Implementation

1. Enabling bfdd to be part of ODL deployment.
2. Configuration of bfdd from ODL via thrift interface (bfdRxInterval, bfdFailureThreshold, bfdTxInterval, bfd-DebounceDown, bfdDebounceUp)
3. BFDD shall inform session status to BGPD.
4. BGP shall react to BFDD session notifications with DC-Gwy.
5. ODL shall implement, new thrift api's for "(un)configuring bfdd", "peer notifications up/down".
6. on peer down notification from bfd, ODL shall disable ECMP bucket for the respective tunnel towards the peer. Raise an alarm, indicating peer-down.
7. on peer up notification from bfd, bgpd shall enable BGP communication with peer. ODL shall disable peer-down alaram.
8. Configuration/debugging : new CLI (command line interface) for configuration and debugging. REST interface for configuration.

Assignee(s)

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

Primary assignee:

- Ashvin Lakshmikantha
- Siva Kumar Perumalla

Other contributors:

- Vyshakh Krishnan C H
- Shankar M

Work Items

Will be added before start of implementation.

Dependencies

- DC-Gwy: MUST support BFD monitoring of the BGP control plane
- genius: yang changes in aliveness monitor

Testing

- Configuration: bgp, bfdd peer configuration, neighborship establishment and route exchange between DC-Gwy1, DC-Gwy2 and ODL with ECMP enabled OVS.
- Data Path: Advertise prefix p1 from both DC-Gwy1, DC-Gwy2, traffic shall be distributed to both DC-Gwy(s).
- Reboot DC-Gwy2, peer down notification shall be observed in logs within 2Seconds. Traffic shall be switched to DC-Gwy1.
- When DC-Gwy2 comes back up, peer up notification shall be observed in logs, traffic shall be distributed between DC-Gwy1 and DC-Gwy2.
- Verification of bfdDebounceDown/bfdDebounceUp timers by flaping connection between ODL and DC-Gwy(s)
- Sanity check of existing BGP behavior, by disabling bfd.
- non-HA scenario: sanity check of existing BGP behavior, with single DC-Gwy (includes Graceful-Restart, admin down, . . .).

Unit Tests

Integration Tests

CSIT

Documentation Impact

Yes, Documentation will have an impact.

Contributors to documentation

- Ashvin Lakshmikantha
- Siva Kumar Perumalla

References

none.

Table of Contents

Hairpinning of floating IPs in flat/VLAN provider networks

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

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

Problem description

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

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

Use Cases

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

Proposed change

- The vpn-id used for classification of floating ips and router gateway external addresses in flat/VLAN provider networks is based on the external network id. It will be changed to reflect the subnet id associated with the floating ip/router gateway. This will allow traffic from the SNAT/outbound NAPT table to be resubmitted back to the FIB while preserving the subnet id.
- Each floating ip already has VRF entry in the fib table. The vpn-id of this entry will also be based on the subnet id of the floating ip instead of the external network id. If the VM associated with the floating ip is located on remote compute node, the traffic will be routed to the remote compute based on the provider network of the subnet from which the floating ip was allocated e.g. if the private network is VxLAN and the external network is VLAN provider, traffic to floating ip on remote compute node will be routed to the provider port associated with the VLAN provider and not the tunnel associated with the VxLAN provider.

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

Pipeline changes

Egress traffic from VM with floating IP to the internet

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

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

Ingress traffic from the internet to VM with floating IP

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

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

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

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

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

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

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

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=ext-net-id` =>
GW Mac table (19) `match vpn-id=ext-net-id,dst-mac=router-gw mac` =>
FIB table (21) `match: vpn-id=ext-net-id,dst-ip=router-gw set vpn-id=router-gw-subnet-id` =>

Inbound NAPT table (44) `match: dst-ip=router-gw,port=ext-port set dst-ip=vm-ip,vpn-id=router-id,port=int-port` =>
PFIB table (47) `match: vpn-id=router-id` =>
FIB table (21) `match: vpn-id=router-id,dst-ip=vm-ip` =>
Local Next-Hop group: `set dst-mac=vm-mac,reg6=vm-lport-tag` =>
Egress table (220) output to VM port

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

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

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

Hairpinning - VM traffic to floating ip on remote compute node

VM originating the traffic (Ingress DPN):

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

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

Egress table (220) output to provider network

VM receiving the traffic (Egress DPN):

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

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

Hairpinning - traffic from VM with no associated floating IP to floating ip on remote compute node

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

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

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

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

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

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>

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

VM receiving the traffic (Egress DPN):

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

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

Yang changes

odl-nat module will be enhanced with the following container

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

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

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

```
container ext-routers {
  list routers {
    key router-name;
    leaf router-name {
      type string;
    }
    ...

    leaf-list external-subnet-id {
      type yang:uuid; }
    }
  }
}
```

Configuration impact

None

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

None

Targeted Release

Carbon

Alternatives

None

Usage

Create external network with two subnets

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

Create internal networks with subnets

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

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

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

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

```
neutron router-create router3
neutron router-interface-add <router3-uuid> <private-subnet3-uuid>
neutron router-interface-add <router3-uuid> <private-subnet4-uuid>
neutron router-gateway-set --fixed-ip subnet_id=<public-subnet1-uuid> --fixed-ip␣
↪subnet_id=<public-subnet2-uuid>
<router3-uuid> <public-net-uuid>
```

Create floating ips from both subnets

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

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

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

Connectivity tests

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

Features to Install

odl-netvirt-openstack

REST API

N/A

CLI

N/A

Implementation

Assignee(s)

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

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

Work Items

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

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

Dependencies

None

Testing

Unit Tests

Integration Tests

CSIT

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

Documentation Impact

None

References

[1] OpenDaylight Documentation Guide

Table of Contents

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

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

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

Problem description

Provide IPv6 external connectivity to virtual machines located in Data center can be achieved through use of Globally Unique Addresses and usage of BGP VPN concepts. Even if VPN IPv6 is made to interconnect hosts without the help of any NAT mechanisms, routing to the external network for internet should be easily configured.

Keep in mind that key aspects of configuring IPv6 external connectivity should rely on Openstack and VPN concepts.

There are already solutions to provide north south communication for IPv6 as depicted in [6]. This document relies on L3VPN concepts to provide the same behaviour.

The document explores how VPN could be configured so as to provide IPv6 external connectivity. The document explores a solution for Only IPv6 Globally Unique Address.

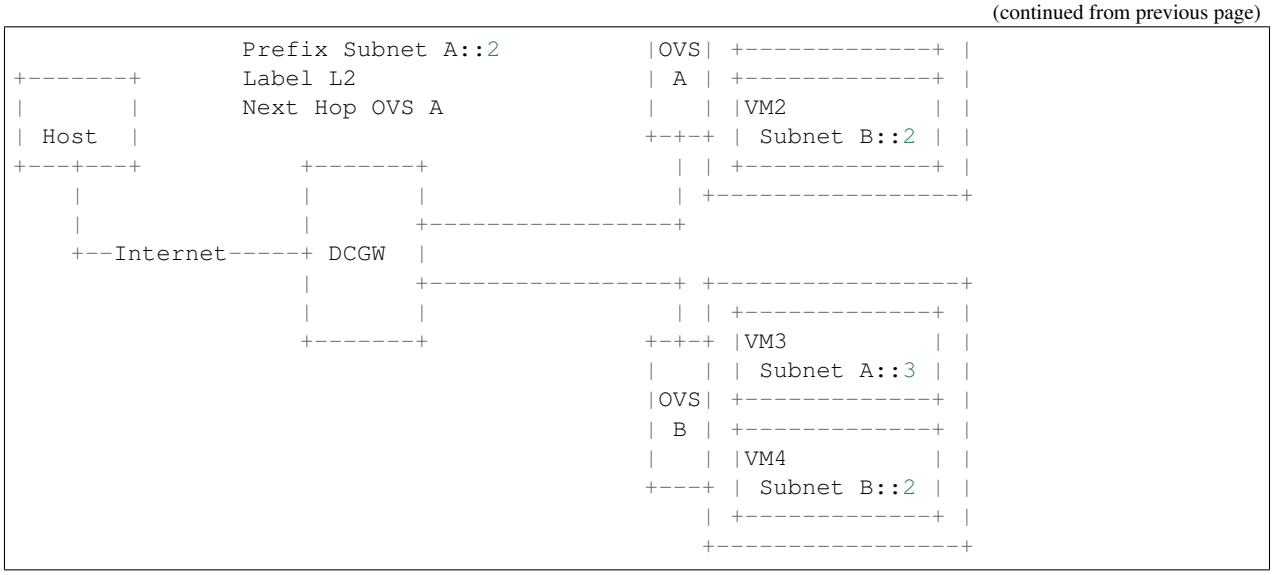
Some caution need to be taken care with the solution chosen. As this is private VPN, that means that it should be possible to use a VPN for both usages, that is to say inter-DC and IPv6 external connectivity. Also, some security concerns must be taken care. Because VPN interacts with external equipment, the internal prefixes that are not authorised to access to the internet, should not be made visible to the DC-GW.

Following schema stands for what happens on the flows on the datacenter. For instance, the same MPLSoGRE tunnel can be used for both Inter-DC and IPv6 external connectivity.

```
                                OVS A flow:
                                IP dst not in advertised list
                    VPN configuration explained in use case chapter
                                               +-----------------+
                                               | +-------------+ |
                                           +---+ |VM1          | |
                  BGP table                |   | | Subnet A::2 | |
```

(continues on next page)

(continued from previous page)



Use Cases

Let's say an operator can configure data center gateways with a VPN dedicated to Internet connectivity.

Based on this configuration, if a virtual machine is spawned within a data center subnet, then it should be possible for that IPv6 GUA subnet to be imported to that VPN. As consequence of this relationship, a BGP UPDATE message containing MP-BGP attributes required for reaching the VM outside the datacenter would be sent to the DC-GW. In the same manner, adding extra-route or declaring subnetworks will trigger the same.

There are several techniques for tenant VMs to access the Internet, through usage of VPNs. Those methods are described in [8], on section 11. Also a note describes in [7] the different techniques that could be applied to the DC-GW case. Note that not all solutions are compliant with the RFC. One of the solutions from [7] are discussed in sub-chapter 'Proposal based on VPN semantics'. It is demonstrated that [7] is not correct.

An other solution, described in [9], on slides 41, and 42, discusses the problem differently. It relies on openstack neutron concepts. It proposes that IPv6 local entries could be exported to an external network, whenever that network is attached to a neutron router, and that external network is associated to an internet VPN. Solution is exposed in sub-chapter 'Proposal based on External Network'.

Solution described in [9] will be the chosen one. Consecutive chapters will describe how to implement [9], slide 41, 42.

Proposal based on External Network

Neutron configuration perspective

Configuring an external network and associate an internet VPN to that external network is the solution the specification wants to promote for IPv6 GUA.

Following scheme can help. It shows a logical overview of what needs to be configured on openstack point of view. As you can see, router is the object that makes relationship between internal and external world. On internal view, you can configure either subnetwork with router, directly. You can also associate an external BGPVPN to a second private network (here subnet B). This is for inter DC purposes. Even, you can associate router (here router 2) with an external BGPVPN 2, for inter DC purposes.

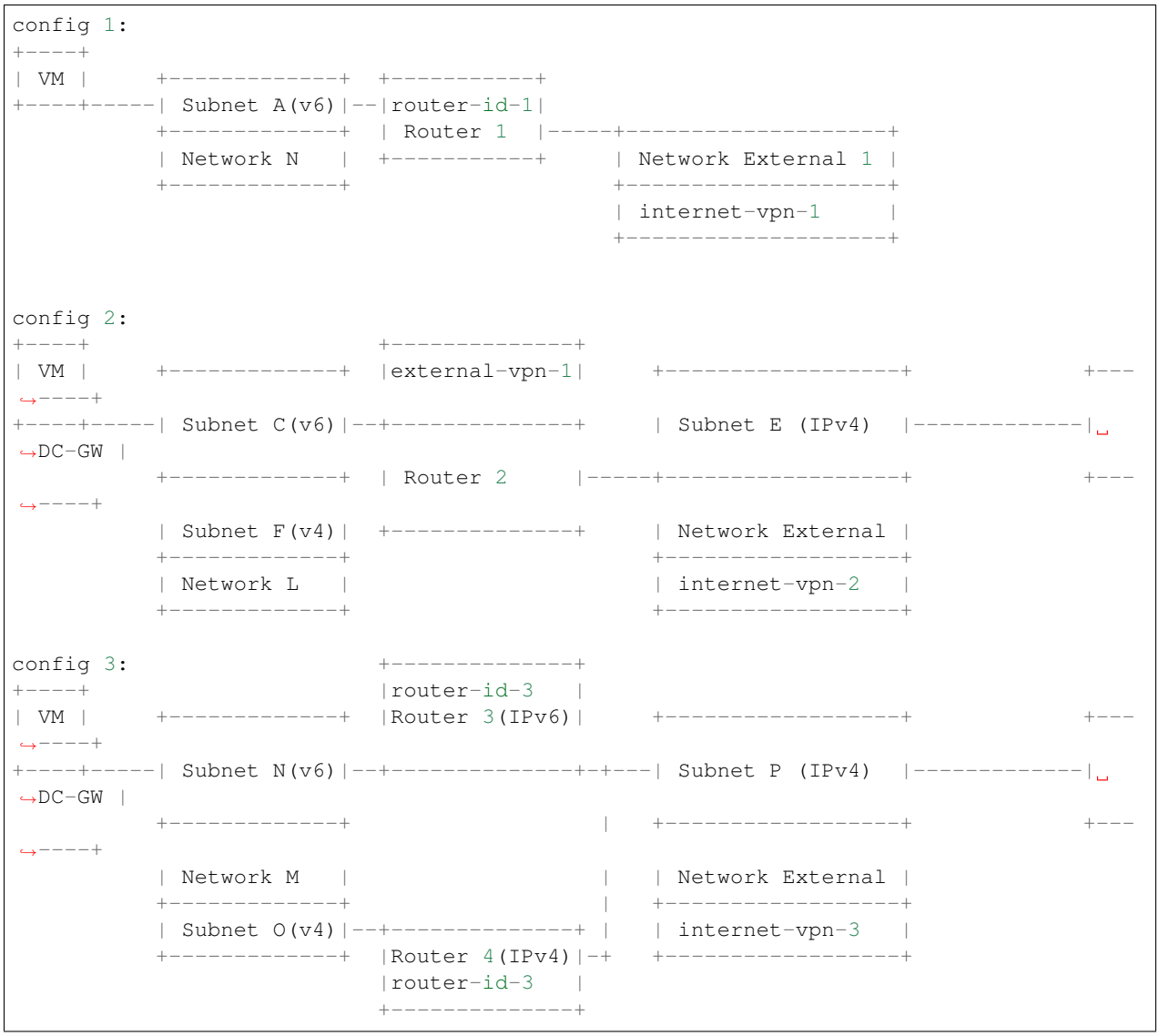
The drawing illustrates also the dual stack example, because the router we are working on may be Dual stack. That is to say that it may host both IPv4 and IPv6 subnetworks.

Also, an other use case (config 4) involves a two router solution, with one IPv4 router , one IPv6 router solution. The customer can choose to tear-down access to external network for IPv4 ONLY (or) for IPv6 ONLY subnets for such DualStack VM, by doing a router-gateway-clear on the respective router. This provides good flexibility.

In all cases, to reach the external connectivity, you need to configure an external network, using one of the two methods described.

The following order will be used to support external network connectivity:

- config 1: IPv6 network connectivity using BGP VPN in single router solution
- config 2: IPv6 network connectivity using BGP VPN in dual stack router solution
- config 4: IPv6 network connectivity using BGP VPN in a two router solution



Discussion of the various setups solutions

In all cases, the following happens:

- All subnetworks from external network will be imported into the VPN as before. In our case, as we have an IPv4 provider network, the IPv4 public IP address will be imported.
- Second, all IPv6 subnets attached to the router that use that external network will be imported in that internet VPN. Note that in the case of a dual stack router, IPv4 subnets are not concerned, since those IPv4 subnets are private.
- Note that it is not necessary to configure a default gateway IP, because all traffic is encapsulated into MPLSoGRE tunnel.

To summarise, the proposal impacts only IPv6 private subnets, even in dual stack routers, and two router solution. There are no changes for IPv4 subnets, and floating IPs (related to IPv4). The implementation should be OK independently of the various orchestration choices used.

About the solution involving single stack IPv6 router, the admin must create an external IPv4 network. This is the necessary condition to have IPv6 encapsulated in MPLSoGRE IPv4 tunnel.

About the solution involving a two router solution, a work is in progress in [10]. Testing will be possible on such solution, only when [10] will be made available.

Discussion on internet VPN impact with IPv4

The internet VPN proposal is still assuming the fact that the user wants to deploy IPv6 GUA. Whenever a subnetwork, IPv4 or IPv6, wants to reach the outside, it uses openstack neutron router. With IPv6, it only needs to configure an external network. If IPv4 is also needed, then it needs to configure a neutron sub-network. Because this method is used, no default gateway is needed, since the VPN handles the forwarding to the DC-GW.

If the IPv4 traffic is used, then the NAT mechanism will be put in place by "natting" the private network with the outgoing IP address of the external router. All subnets from external network will be imported into the internet VPN. If the IPv6 traffic is used, then the users that want to provide internet connectivity, will use L3VPN feature to import private IP to the VPN that has been created for internet connectivity. That VPN could be called "Internet VPN", and must be associated to the external network defined in the router. That association will be administratively configured by using command "neutron bgpvpn-assoc-create" command, so as to associate external network with BGPVPN. Note also that using this command does not control the private IPv6 subnets that will be imported by that BGPVPN. The IPv6 subnetworks can be either GUA or LUA, since no control is done for that. It will be up to the administrator to be cautious regarding the configuration, and use only IPv6 subnetworks. As the "Internet VPN" also imports internet routes provided by DC-GW, that VPN is able to create the necessary pipeline rules (the necessary MPLS over GRE tunnels), so that the various VMs that are granted, can access to the Internet.

Configuration steps

Configuration steps in a datacenter, based on config 1 described above:

- Configure ODL and Devstack networking-odl for BGP VPN.
- Create a transport zone to declare that a tunneling method is planned to reach an external IP: the IPv6 interface of the DC-GW
- Create a network and an IPv6 GUA subnetwork private, using GUA prefix

```
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 <GUA prefix>
```

- Create a Neutron Router

```
neutron router-create <router>
```

- Create an external network. No IPv4 or IPv6 subnetwork needs to be configured.

```
neutron net-create --router:external=true gateway_net
```

- The step create the L3VPN instances. As illustrated, the route distinguisher and route target are set to 100:1.

```
neutron bgpvpn-create --route-distinguishers <internetvpn>
   --route-targets <internetvpn> --tenant-id b954279e1e064dc9b8264474cb3e6bd2 --name␣
↪internetvpn
```

- step (1) : Connect the router ports to the internal subnets that need to access to the internet.

```
neutron router-interface-add <router> ipv6-int-subnet
```

- step (2) : The external network will be associated with the "internet VPN" instance.

```
operations:neutronvpn:associateNetworks ( "network-id":"<uuid of external network␣
↪gateway_net >"
                                          "vpn-id":"<uuid of internetvpn>")
```

- step (3) : The external network will be associated to the router.

```
neutron router-gateway-set <router> gateway_net
```

The last 3 operations on configuration steps have a step number: step (x) for example. Note that step-ids (1), (2), and (3) can be combined in different orders.

Proposed change

The proposal based on external network is the one chosen to do changes. The change relies on config 1 and config 3 described above.

The changes consist in :

- extending the neutronvpn.yang subnet structure so as to link the internet vpn to the private subnetwork.
- each existing external sub-network is imported to the internet VPN. This is the case for IPv4 subnetwork, as it has been described above. This can also be the case for IPv6 sub-networks.
- for each new VM, extra route, subnet new to the private network or the private VPN, only the IPv6 information is imported to the internet VPN.
- providing a fallback rule that says that no other rules in routing table of the virtual router is available, then a default route is conveyed to that external network.

For doing L3 forwarding, the packet will be transported to either the neutron router, or the private VPN. In both cases, the packet will reach table 17, for L3forwarding. If there is no external VPN attached, then the packet is transported to the table 17, using vpn-id=router-id[1/2/3/4]. If there is an external VPN attached, then the packet is transported to table 17, using vpn-id=vpn-external-1. Then, a check will be done against <internet-vpn-[1/2/3]>.

For IPv6 traffic, the internet VPN will be a fallback mechanism so that they go to the Internet. A fallback mechanism similar to option 2 from [7] will be put in place, only for IPv6.

That means that in such configuration, if a dual stack router is configured with both IPv4 and IPv6, then the VPN would only consider IPv4 public addresses and IPv6. IPv4 private traffic should follow NAT rules applied to the

router. Then if the new IPv4 public packet's destination IP address matches addresses from the internet VPN, then the packet will be encapsulated into the MPLSoGRE tunnel.

Neutron VPN Changes

Neutron's role fill in internet VPN information in a subnetmap structure.

VPN - IPv6 subnetwork relationship established

The 3 following conditions must be met, so that prefixes importation to the internet VPN will occur. - on that subnet, some routing information is bound: (VMs allocated IPs, extra route or subnet-routing configured) - the same router has an external network configured - the external network is being associated a VPN. - only IPv6 subnetworks are imported, because IPv4 subnetworks may be private.

NeutronVPN listens for events that involve change of the above, that is to say:

- attach a subnetwork from router. A check is done on the nature of the subnet: IPv6. A check is done also to see on the list of external networks configured on the router, if there are any attached VPN.
- attach an external network to router. A check is done on the presence of a VPN to the external network or not.
- associate network to VPN. If the network associated is external, a check is done on the routers that use that network.

If above condition is met, NeutronVPN will update subnetmap structure.

VPN - IPv6 Subnetwork Relationship unestablished

If above condition is not met, the following will be triggered, depending on the incoming events.

- for a detached subnetwork from router, a check is done if a VPN is associated to the external network of that router.
- for an external network detached from router, a check is done to see if that network had a VPN instance.
- for a VPN disassociated from a network, the VPN instance is elected.

If above condition is met, NeutronVPN will update subnetmap structure.

VPN Manager Changes

Upon subnetmap structure change, VPN manager will create subnetopdataentries structures corresponding to the two kind of VPN handled by subnetmap structure : either internet or external VPN.

So that at maximum, for one subnet instance, two subnetopdataentries instances will be created.

Consecutive to that change, VPN manager will add or delete FIB entries according to the information stored on subnetopdataentry.

A populate of the FIB will be triggered for all adjacencies linked to that subnetID of the subnetOpdataEntry. The specific route distinguisher of the corresponding VPN will be used.

Pipeline changes

Associating BGPVPN to external network will act as if a second network was accessible through internet-vpn-id.

Pipeline change for upstream. Indeed, the internet VPN will be translated into a fallback rule for external access. This happens if there is external connectivity access, by using VPN associated to external network. This applies only to IPv6 traffic.

Packets going out from VM will match against either L3 forwarding in the DC, or L3 forwarding using L3VPN. Assuming this, once in table 21 (L3 FIB table), the packet will be tested against an IPv6 packet. If it is the case, the packet will be resubmitted to table 21 (L3 FIB table), to see if it matches some entries of the internet VPN table. If it is the case, then the packet will be encapsulated with the correct MPLSoGRE tag.

Below are illustrated 3 use cases that have been identified.

- case 1 based on config 1 described above
- case 2 based on config 3 described above
- case 3 based on config 1 with multipath case

Case VM to DC-GW with VPN internet configured, and standard Layer 3 routing (config 1)

Note that this rule is available only for IPv6 traffic.

Lport Dispatcher Table (17) `match: LportTag l3 service: set vpn-id=router-id` =>
DMAC Service Filter (19) `match: dst-mac=router-internal-interface-mac vpn-id=router-id` =>
L3 FIB Table (21) `priority=0,match: ipv6,vpn-id=router-id, set vpn-id=internetvpn-id, resubmit(,21)` =>
L3 FIB Table (21) `match: vpn-id=internet-vpn-id, nw-dst=<IP-from-internetvpn> set tun-id=mpls_label output to MPLSoGRE tunnel port` =>

Case VM to DC-GW with VPN internet configured, and Inter-DC VPN configured (config 3)

Note that this rule is available only for IPv6 traffic.

Classifier Table (0) =>
Lport Dispatcher Table (17) `match: LportTag l3vpn service: set vpn-id=external-l3vpn-id` =>
DMAC Service Filter (19) `match: dst-mac=router-internal-interface-mac vpn-id=external-vpn-id` =>
L3 FIB Table (21) `match: vpn-external-vpn-id=external-vpn-id, nw-dst=<IP-from-vpn> set tun-id=mpls_label output to MPLSoGRE tunnel port` =>
L3 FIB Table (21) `priority=0,match: ipv6, vpn-id=l3vpn-id, set vpn-id=internet-vpn-id, resubmit(,21)` =>
L3 FIB Table (21) `match: vpn-id=internet-vpn-id, nw-dst=<IP-from-internetvpn> set tun-id=mpls_label output to MPLSoGRE tunnel port` =>

Yang changes

The neutronvpn.yang subnetmap structure will be modified. subnetmap structure will have a new field called

```
leaf vpn-external-id {
   type    yang:uuid;
   description "Internet VPN to which this subnet belongs";
}
```

The odl-l3vpn.yang subnet-op-data-entry will be modified. The key for this structure is being added a new field: vpnname. Vpnname will stand for either the external VPN or the internet VPN.

```
--- a/vpnservice/vpnmanager/vpnmanager-api/src/main/yang/odl-l3vpn.yang
+++ b/vpnservice/vpnmanager/vpnmanager-api/src/main/yang/odl-l3vpn.yang
@@ -346,19 +346,19 @@ module odl-l3vpn {
container subnet-op-data {
config false;
list subnet-op-data-entry {
-             key subnet-id;
+             key "subnet-id vpn-name";
              leaf subnet-id {
              type    yang:uuid;
              description "UUID representing the subnet ";
              }
              leaf vpn-name {
              type string;
              description "VPN Instance name";
              }
              leaf nh-dpnId {
                  type uint64;
                  description "DpnId for the DPN used as nexthop for this subnet";
              }
              leaf vrf-id {
              type string;
              }
```

Configuration impact

None

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

None

Targeted Release

Carbon

Alternatives

None

Usage

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

```
rd="100:2" # internet VPN
  import-rts="100:2"
  export-rts="100:2"
 rd="100:1" # vpn1
  import-rts="100:1 100:2"
  export-rts="100:1 100:2"
```

Following operations are done.

```
POST config/ebgp:bgp
{
    "ebgp:as-id": {
          "ebgp:stalepath-time": "360",
          "ebgp:router-id": "<ip-bgp-stack>",
```

(continues on next page)

(continued from previous page)

```
          "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 a neutron router

```
neutron router-create router1
```

- Create an external network

```
neutron net-create --router:external=true gateway_net
```

- 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"
         }
      ]
   }
}
```

- Associate the private network with the router

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

- Associate the external network with the router

```
neutron router-gateway-set router5 GATEWAY_NET
```

- Associate internet L3VPN To Network

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

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

- Spawn a VM in the tenant network

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

- Dump ODL BGP FIB

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

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

(continues on next page)

(continued from previous page)

```
            },
          ]
        }
      ]
    }
}
```

Features to Install

odl-netvirt-openstack

REST API

CLI

Implementation

Assignee(s)

Primary assignee: Philippe Guibert <philippe.guibert@6wind.com>

Other contributors: Noel de Prandieres <prandieres@6wind.com>

Valentina Krasnobaeva <valentina.krasnobaeva@6wind.com>

Work Items

- Validate proposed changes - reuse subnetmap
- Implement NeutronVpn and VpnManager
- Testing

Dependencies

[5]

Testing

The configurations 1 and 2 will be used. For each of the configs used, the internet VPN method will be used. Also, each config will be done with dual stack router, and with IPv6 router only. 3 operations will trigger the association between private network and external network: - associate subnet to router - associate Router to External Network - associate External Network to Internet VPN

Following workflows should be tested OK

- Subnets -> Router, Router -> Ext Net, Ext Net -> Int. VPN
- Subnets -> Router, Ext Net -> Int. VPN, Router -> Ext Net
- Ext Net -> Int. VPN, Router -> Ext Net, Subnets -> Router

- Router -> Ext Net, Ext Net -> Int. VPN, Subnets -> Router
- Router -> Ext Net, Subnets -> Router, Ext Net -> Int. VPN
- Ext Net -> Int. VPN, Subnets -> Router, Router -> Ext Net

Unit Tests

TBD

Integration Tests

TBD

CSIT

TBD

Documentation Impact

A design document will be provided. 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] BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN

[5] Spec to support IPv6 Inter DC L3VPN connectivity using BGPVPN.

[6] Spec to support IPv6 North-South support for Flat/VLAN Provider Network.

[7] External Network connectivity in IPv6 networks.

[8] BGP/MPLS IP Virtual Private Networks (VPNs)

[9] IPv6 Support in MPLS over GRE overlays

[10] Spec to support L3VPN dual stack for VMs

Table of Contents

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

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

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

Problem description

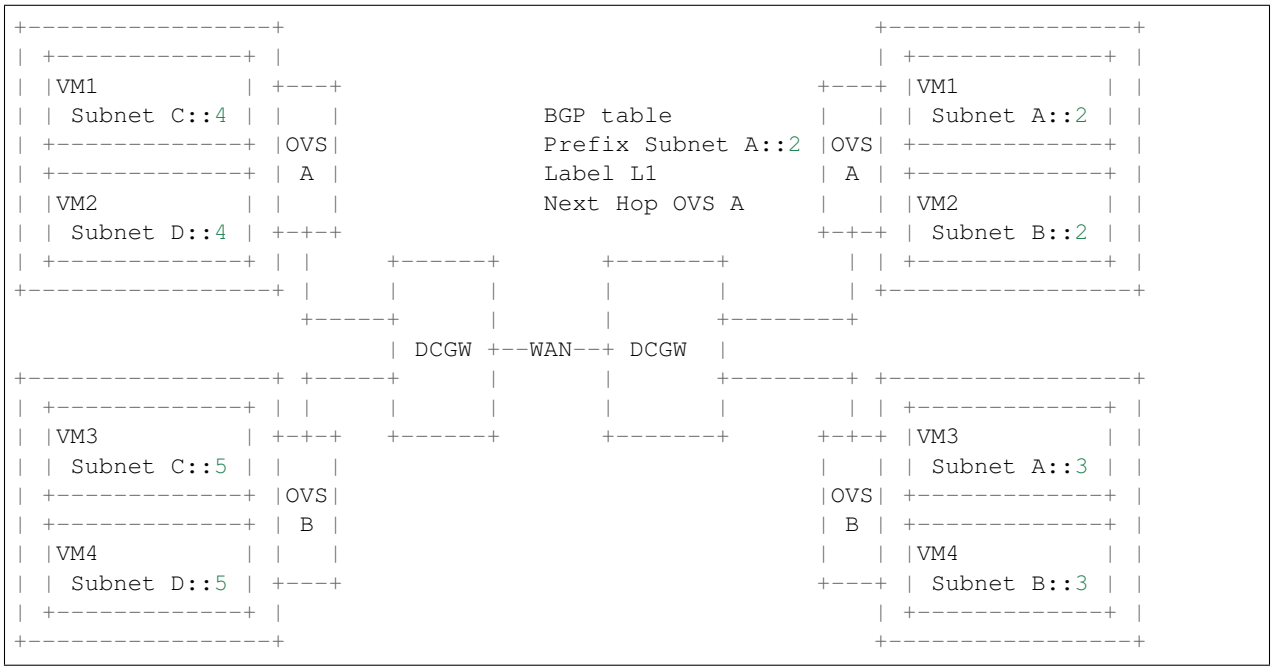
Provide IPv6 connectivity to virtual machines located in different subnets spread over multiple sites or Data center can be achieved through use of Globally Unique Addresses and capacity to update enough routing tables to forge a path between the two. Even if IPv6 is made to interconnect hosts without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

Keep in mind that key aspects of L3VPN connectivity is Route Targets and VPN-IPv6 address family. Assuming an operator can configure both data center gateways with same Route Distinguisher or set of imported Route Targets, each time a virtual machine is spawned within a new subnet, it will trigger the send of a BGP UPDATE message containing

MP-BGP attributes required for reaching the VM. Such behavior can be achieved by configuring a neutron router a default gateway.

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

Following schema could help :



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

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

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

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

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

Use Cases

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

Steps in both data centers :

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

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

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

Proposed change

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

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

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

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

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

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

Pipeline changes

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

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

Classifier Table (0) =>
LFIB Table (20) `match: tun-id=mpls_label set vpn-id=l3vpn-id, pop_mpls label, set output to nexthopgroup-dst-vm` =>
NextHopGroup-dst-vm: `set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

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

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

Classifier Table (0) =>
Lport Dispatcher Table (17) `match: LportTag l3vpn service: set vpn-id=l3vpn-id` =>
DMAC Service Filter (19) `match: dst-mac=router-internal-interface-mac l3vpn service: set vpn-id=l3vpn-id` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=ext-ip-address set tun-id=mpls_label output to MPLSoGRE tunnel port` =>

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

Yang changes

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

EBGP YANG changes

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

Listing 41: ebgp.yang

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

    leaf rd {
          type string;
    }

    leaf prefix-len {
          type string;
    }

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

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

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

Configuration impact

None

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

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

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

This fact also impact the scaling of the BGP speaker implementation (Quagga BGP) with many thousands of BGPVPNv4 and BGPVPNv6 prefixes (as much as number of spawned VMs) with best path selection algorithm on route updates, graceful restart procedure, and multipath.

Targeted Release

Carbon

Alternatives

None

Usage

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

```
POST config/ebgp:bgp
{
    "ebgp:as-id": {
          "ebgp:stalepath-time": "360",
          "ebgp:router-id": "<ip-bgp-stack>",
          "ebgp:announce-fbit": "true",
          "ebgp:local-as": "<as>"
    },
    "ebgp:vrfs": [
     {
       "ebgp:export-rts": [
         "<export-rts>"
       ],
       "ebgp:rd": "<RD>",
```

(continues on next page)

(continued from previous page)

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

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

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

- Create a router and associate it to internal subnets.

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

- Use neutronvpn:createL3VPN REST api to create L3VPN

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

- Associate L3VPN To Routers

```
POST /restconf/operations/neutronvpn:associateRouter
{
```

(continues on next page)

(continued from previous page)

```
  "input":{
     "vpn-id":"vpnid_uuid",
     "router-id":[ "router_uuid" ]
   }
}
```

- Create MPLSoGRE tunnel between DPN and DCGW

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

- Spawn a VM in the tenant network

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

- Dump ODL BGP FIB

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

Features to Install

odl-netvirt-openstack

REST API

CLI

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

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

Implementation

Assignee(s)

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

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

Work Items

- Implement necessary APIs to allocate a transport over IPv6 requirement configuration for a given Route Target as the primary key.
- Support of BGPVPNv6 prefixes within MD-SAL. Enhance RIB-manager to support routes learned from other bgp speakers, [un]set static routes.
- BGP speaker implementation, Quagga BGP, to support BGPVPN6 prefixes exchanges with other BGP speakers (interoperability), and thrift interface updates.
- Program necessary pipeline flows to support IPv6 to MPLS/GRE (IPv4) communication.

Dependencies

Quagga from 6WIND is publicly available at the following url

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

Testing

Unit Tests

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

Integration Tests

TBD

CSIT

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

Documentation Impact

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

References

[1] OpenDaylight Documentation Guide

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

[3] http://docs.openstack.org/developer/networking-bgpvpn/overview.html

[4] Spec to support IPv6 North-South support for Flat/VLAN Provider Network.

[5] BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN

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

Table of Contents

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

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

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

Problem description

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

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

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

Initially, we would like to pursue a Centralized IPv6 router (CVR) use-case and look into a fully distributed router via a future spec. One of the main reasons for pursuing the CVR over DVR is that OpenStack Neutron creates only a single router gateway port (i.e., port with device owner as network:router_gateway) when the router is associated with the external network. When implementing a distributed router, we cannot use the same router gateway port MAC address

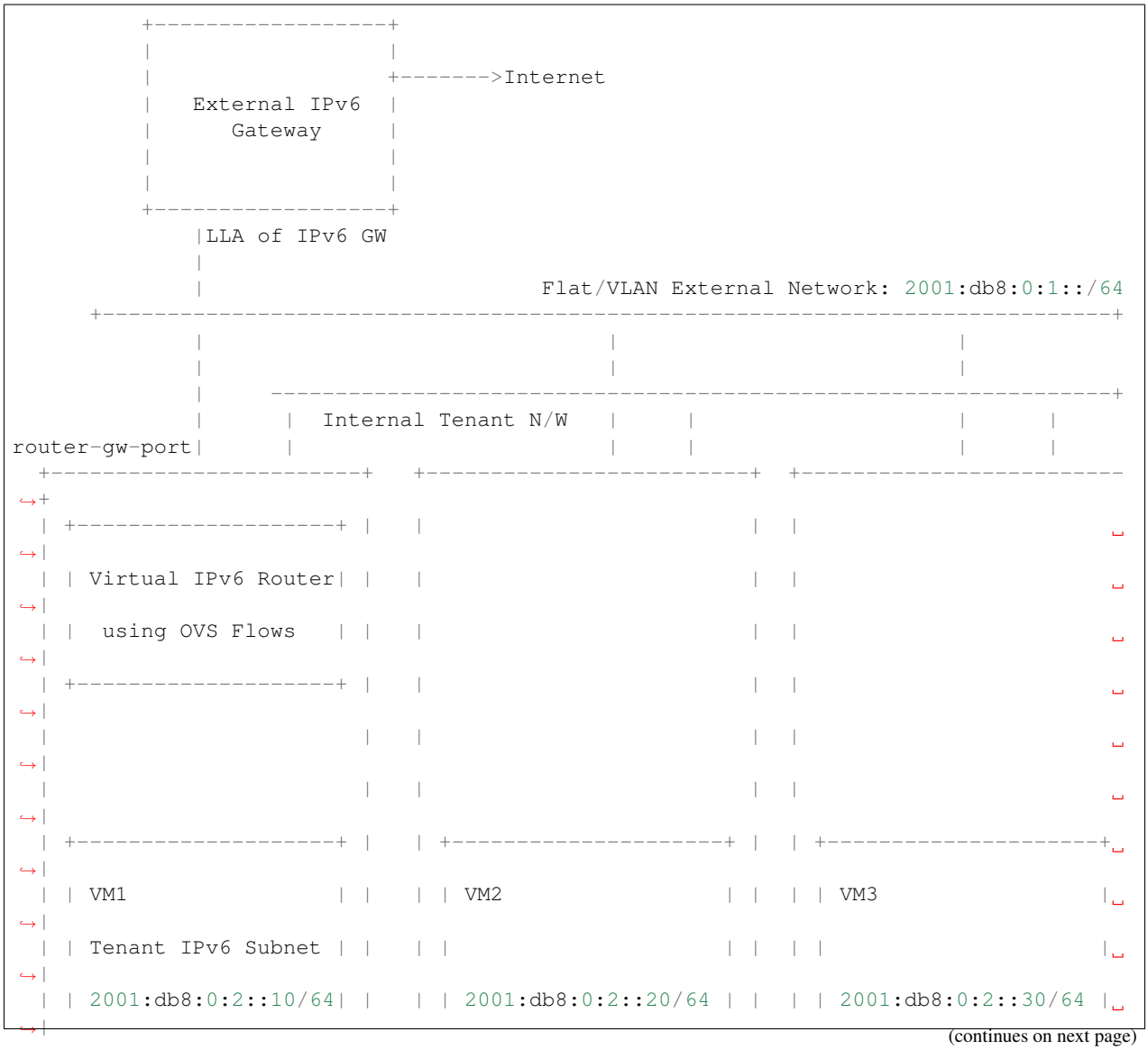
from multiple Compute nodes as it could create issues in the underlying physical switches. In order to implement a fully distributed router, we would ideally require a router-gateway-port per compute node. We will be addressing the distributed router in a future spec taking into consideration both IPv4 and IPv6 use-cases.

Use Cases

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

Steps:

- Create a tenant network with IPv6 subnet using GUA/ULA prefix or an admin-created-shared-ipv6-subnet-pool.
- Create an external network of type FLAT/VLAN with an IPv6 subnet where the gateway_ip points to the Link Local Address (LLA) of external/physical IPv6 gateway.
- Create a Neutron Router and associate it with the internal subnets and external network.
- Spawn VMs on the tenant network.



(continues on next page)

(continued from previous page)

```
  | +--------------------+ |    | +--------------------+ |   | +--------------------+␣
↪|
  +------------------------+    +------------------------+   +------------------------
↪+
  Compute Node-1 designated           Compute Node-2               Compute Node-3
  as NAPT Switch for router1
```

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 ProviderNetworkGroup 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;
```

(continues on next page)

(continued from previous page)

```
  }

  revision "2017-02-10" {
      description "IPv6 Neighbor Discovery Util module";
  }

  grouping interfaces {
      list interface-address {
         key interface;
         leaf interface {
              type leafref {
                  path "/if:interfaces/if:interface/if:name";
              }
         }
         leaf src-ip-address {
              type inet:ipv6-address;
         }
         leaf src-mac-address {
              type yang:phys-address;
          }
      }
  }

  rpc send-neighbor-solicitation {
      input {
          leaf target-ip-address {
              type inet:ipv6-address;
          }
          uses interfaces;
      }
  }
}
```

neighbor-solicitation-packet container in neighbor-discovery.yang would be enhanced with Source Link Layer optional header.

```
container neighbor-solicitation-packet {
  uses ethernet-header;
  uses ipv6-header;
  uses icmp6-header;
  leaf reserved {
      type uint32;
  }
  leaf target-ip-address {
      type inet:ipv6-address;
  }
  leaf option-type {
      type uint8;
  }
  leaf source-addr-length {
      type uint8;
  }
  leaf source-ll-address {
      type yang:mac-address;
  }
}
```

Configuration impact

None

Clustering considerations

None

Other Infra considerations

None

Security considerations

Scale and Performance Impact

- In the proposed implementation, we have to configure a static route on the external IPv6 Gateway with next-hop as the router-gateway-ip. In a future patch, we would enhance the implementation to use BGP for advertising the necessary routes.
- When the external IPv6 Gateway wants to contact the tenant VMs, it forwards all the traffic to the router-gateway-port on the designated NAPT Switch. To know the target-link-layer address of the router-gw-port, the external IPv6 Gateway would send out a Neighbor Solicitation for the router-gateway-port-ip. This request would be punted to the Controller and ipv6service would respond with the corresponding Neighbor Advertisement. In large deployments this can become a bottleneck. Note: Currently, OpenFlow does not have support to auto-respond to Neighbor Solicitation packets like IPv4 ARP. When the corresponding support is added in OpenFlow, we would program the necessary ovs flows to auto-respond to the Neighbor Soliciation requests for router-gateway-ports.

Targeted Release

Carbon

Alternatives

An alternate solution is to implement a fully distributed IPv6 router and would be pursued in a future SPEC.

Usage

- Create an external FLAT/VLAN network with an IPv6 (or dual-stack) subnet.

```
neutron net-create public-net -- --router:external --is-default
--provider:network_type=flat --provider:physical_network=public

neutron subnet-create --ip_version 6 --name ipv6-public-subnet
--gateway <LLA-of-external-ipv6-gateway> <public-net-uuid> 2001:db8:0:1::/64
```

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

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

- Create a router and associate the external and internal subnets. Explicitly specify the fixed_ip of router-gateway-port, as it would help us when manually configuring the downstream route on the external IPv6 Gateway.

```
neutron router-create router1
neutron router-gateway-set --fixed-ip subnet_id=<ipv6-public-subnet-id>,ip_
↪address=2001:db8:0:10 router1 public-net
neutron router-interface-add router1 ipv6-int-subnet
```

- Manually configure a downstream route in the external IPv6 gateway for the IPv6 subnet "2001:db8:0:2::/64" with next hop address as the router-gateway-ip.

```
Example (on Linux host acting as an external IPv6 gateway):
ip -6 route add 2001:db8:0:2::/64 via 2001:db8:0:10
```

- Spawn a VM in the tenant network

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

Features to Install

odl-netvirt-openstack

REST API

CLI

Implementation

Assignee(s)

Primary assignee: Sridhar Gaddam <sgaddam@redhat.com>

Other contributors: TBD

Work Items

https://trello.com/c/cqjOFmow/147-ipv6-centralized-router-l3-north-south-support-for-flat-vlan-provider-networks

- Program necessary pipeline flows to support IPv6 North-South communication.
- Enhance ipv6service to send out Neighbor Solicitation requests for the external/physical IPv6 gateway-ip and parse the response.
- Support controller based Neighbor Advertisement for router-gateway-ports on the external network.
- Implement Unit and Integration tests to validate the use-case.

Dependencies

None

Testing

Unit Tests

Necessary Unit tests would be added to validate the use-case.

Integration Tests

Necessary Integration tests would be added to validate the use-case.

CSIT

We shall explore the possibility to validate this use-case in CSIT.

Documentation Impact

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

References

[1] OpenDaylight Documentation Guide

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

[3] Spec to support IPv6 Inter DC L3VPN connectivity using BGPVPN

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

Table of Contents

Dual Stack VM support in OpenDaylight

https://git.opendaylight.org/gerrit/#/q/topic:l3vpn-dual-stack-vms

In this specification we will introduce a support of basic L3 forwarding for dualstack VMs connectivity over L3 in NetVirt. Dualstack VM is a virtual machine that has at least two IP addresses with different ethertypes: IPv4 address and IPv6 address.

In addition to this, the specification ensures initial support of dualstack VMs inside L3 BGPVPN. L3 forwarding for dualstack VMs connectivity inside L3 BGPVPN will be provided for the following variations of L3 BGPVPN:

A. L3 BGPVPN constructed purely using networks;

B. L3 BGPVPN constructed purely using a router;

C. L3 BGPVPN constructed using multiple networks and a router.

Problem description

As a dualstack VM, we assume a VM which has one Neutron Port, i.e. one VNIC, that inherits two IPs addresses with different ethertypes: one IPv4 address and one IPv6 address. We also will use in this document a term singlestack VM to describe a VM, which VNIC possesses either IPv4 or IPv6 address, but not both simultaneously.

So, dualstack VM has two IP addresses with different ethertypes. This could be achieved by two ways:

1. VM was initially created with one VNIC, i.e. one Neutron Port from network with IPv4 subnet. Second VNIC, corresponded to a Neutron Port from another network with IPv6 subnet, was added to this machine after its creation.

2. VM has one Neutron Port from a network, which contains 2 subnets: IPv4 subnet and IPv6 subnet.

OpenDaylight has already provided a support for the first way, so this use-case is not in the scope of the specification. For the second way the specification doesn't intend to cover a use-case when, Neutron Port will possess several IPv4 and several IPv6 addresses. More specifically this specification covers only the use-case, when Neutron Port has only one IPv4 and one IPv6 address.

Since there are more and more services that use IPv6 by default, support of dualstack VMs is important. Usage of IPv6 GUA addresses has increased during the last couple years. Administrators want to deploy services, which will be accessible from traditional IPv4 infrastructures and from new IPv6 networks as well.

Dualstack VM should be able to connect to other VMs, be they are of IPv4 (or) IPv6 ethertypes. So in this document we can handle following use cases:

- Intra DC, Inter-Subnet basic L3 Forwarding support for dualstack VMs;
- Intra DC, Inter-Subnet L3 Forwarding support for dualstack VMs within L3 BGPVPN.

Current L3 BGPVPN allocation scheme picks up only the first IP address of dualstack VM Neutron Port. That means that the L3 BGPVPN allocation scheme will not apply both IPv4 and IPv6 network configurations for a port. For example, if the first allocated IP address is IPv4 address, then L3 BGPVPN allocation scheme will only apply to IPv4 network configuration. The second IPv6 address will be ignored.

Separate VPN connectivity for singlestack VMs within IPv4 subnetworks and within IPv6 subnetworks is already achieved by using distinct L3 BGPVPN instances. What we want is to support a case, when the same L3 BGPVPN instance will handle both IPV4 and IPv6 VM connectivity.

Regarding the problem description above, we would propose to implement in OpenDaylight two following solutions, applying to two setups

1. **two-router** setup solution

One router belongs to IPv4 subnetwork, another one belongs to IPv6 subnetwork. This setup brings flexibility to manage access to external networks. More specifically, by having two routers, where one is holding IPv4 subnet and another is holding IPv6 subnet, customer can tear-down access to external network for IPv4 subnet ONLY or for IPv6 subnet ONLY by doing a router-gateway-clear on a respective router.

Now this kind of orchestration step entail us to put a Single VPN Interface (representing the VNIC of DualStack VM) in two different Internal-VPNs, where each VPN represents one of the routers. To achive this we will use L3 BGPVPN concept. We will extend existing L3 BGPVPN instance implementation to give it an ability to be associated with two routers. As consequence, IPv4 and IPv6 subnetworks, added as ports in associated routers and, hence, IPv4 and IPv6 FIB entries, would be gathered in one L3 BGPVPN instance.

L3 BGPVPN concept is the easiest solution to federate two routers in a single L3 BGPVPN entity. From the orchestration point of view and from the networking point of view, there is no any reason to provide IPv4 L3VPN and IPv6 L3VPN access separately for dualstack VMs. It makes sense to have the same L3 BGPVPN entity that can handle both IPv4 and IPv6 subnetworks.

The external network connectivity using L3 BGPVPN is not in scope of this specification. Please, find more details about this in [6]. Right now, this configuration will be useful for inter-subnet and intra-dc routing.

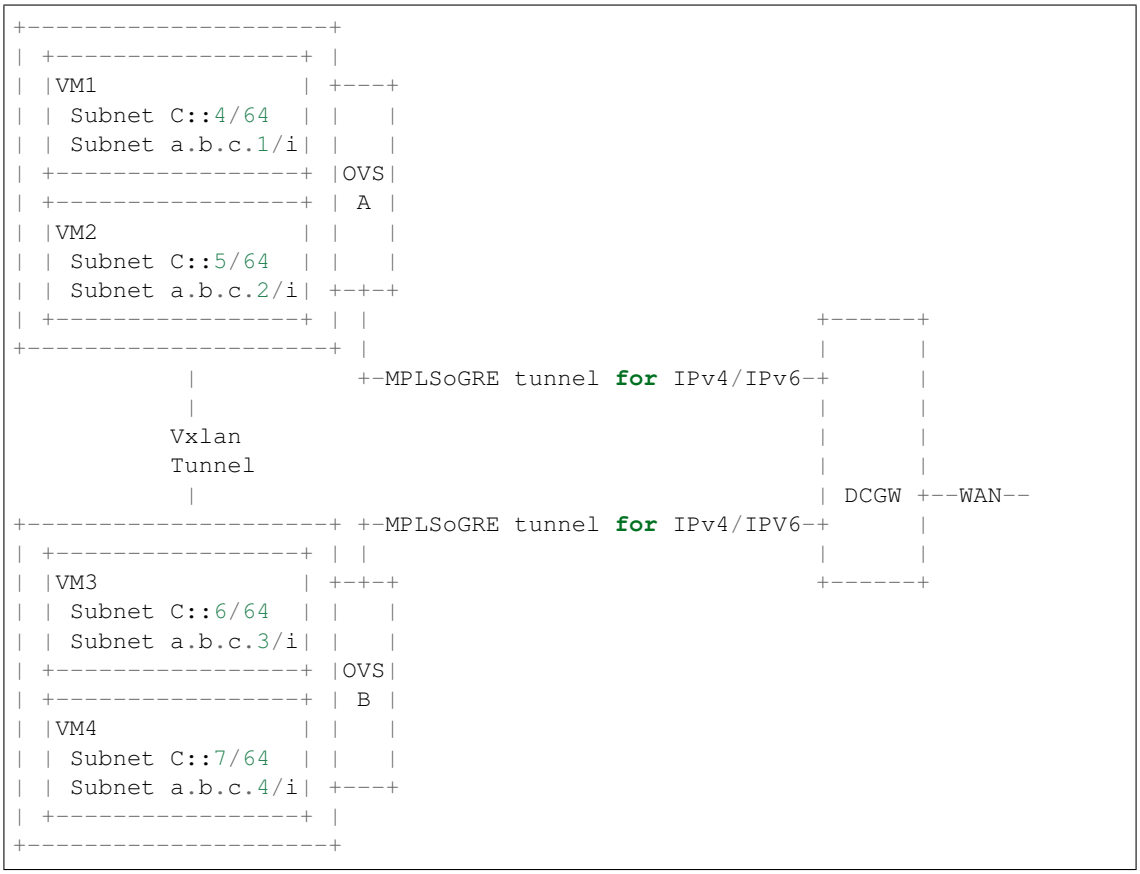
2. **dualstack-router** setup solution

The router with 2 ports (one port for IPv4 subnet and another one for IPv6 subnet) is attached to a L3 BGPVPN instance.

The external network connectivity using L3 BGPVPN is not in the scope of this specification.

Setup Presentation

Following drawing could help :



We identify there 2 subnets:

- IPv4 subnet: a.b.c.x/i
- IPv6 subnet: C::x/64

Each VM will receive IPs from these two defined subnets.

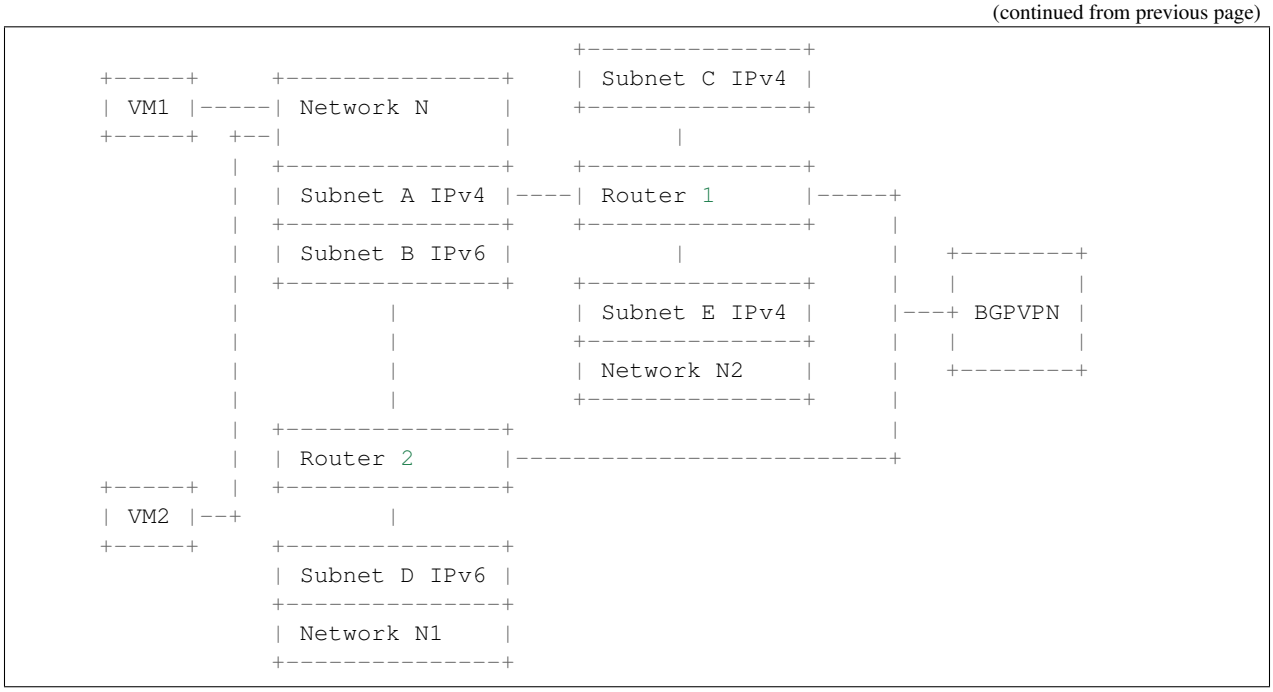
Following schemes stand for conceptual representation of used neutron configurations for each proposed solution.

```
setup 1: two singlestack routers, associated with one BGPVPN
         ("two-router" solution)

                                          +---------------+
                                          | Network N3    |
```

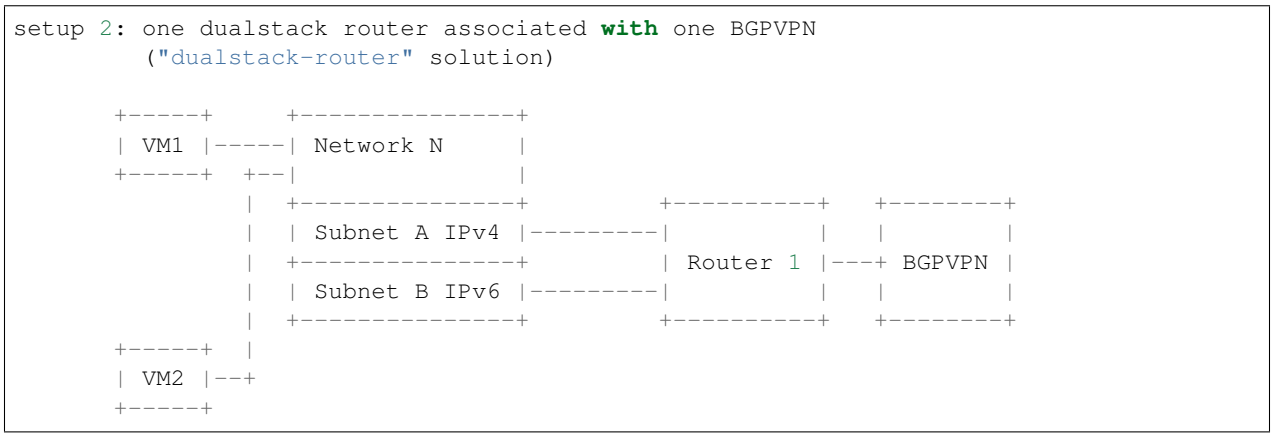
(continues on next page)

(continued from previous page)



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.

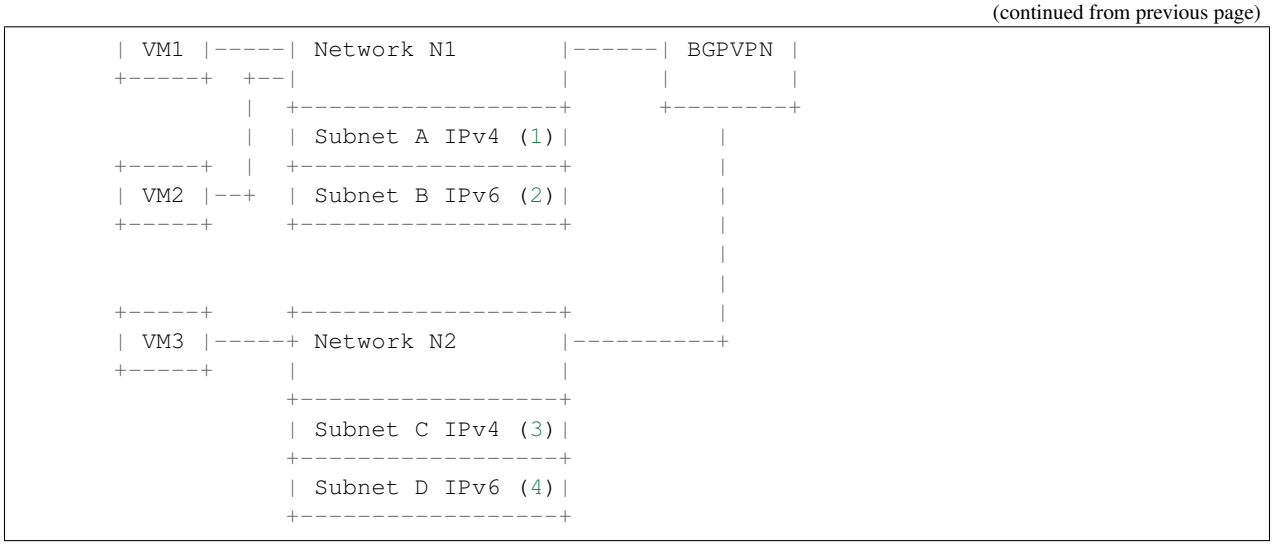


Network N gathers 2 subnetworks, subnet A IPv4 and subnet B IPv6. This makes possible to create Neutron Ports, which will have 2 IP addresses and whose attributes will inherit information (extraroutes, etc) from these 2 subnets A and B.

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.

(continues on next page)

(continued from previous page)



Network N1 gathers 2 subnets, subnet A with IPv4 ethertype and subnet B with IPv6 ethertype. When Neutron Port was created in the network N1, it has 1 IPv4 address and 1 IPv6 address. If user lately will add others subnets to the Network N1 and will create the second Neutron Port, anyway the second VPN port, constructed for a new Neutron Port will keep only IP addresses from subnets (1) and (2). So valid network configuration in this case is a network with only 2 subnets: IPv4 and IPv6. See the chapter "Configuration impact" for more details. Second dualstack network N2 can be added to the same L3 BGPVPN instance.

It is valid for all schemes: in dependency of chosen ODL configuration, either ODL, or Neutron Dhcp Agent will provide IPv4 addresses for launched VMs. Please note, that currently DHCPv6 is supported only by Neutron Dhcp Agent. ODL provides only SLAAC GUA IPv6 address allocation for VMs launched in IPv6 private subnets attached to a Neutron router.

It is to be noted that today, setup 3 can not be executed for VPNv6 with the above allocation scheme previously illustrated. Indeed, only a neutron router is able to send router advertisements, which is the corner stone for DHCPv6 allocation. Either IPv6 fixed IPs will have to be used for this setup, or an extra enhancement for providing router advertisements for such a configuration will have to be done. The setup 3 will be revisited in future.

Known Limitations

Currently, from Openstack-based Opendaylight Bgpvpn driver point-of-view, there is a check, where it does not allow more than one router to be associated to a single L3 BGPVPN. This was done in Openstack, because actually entire ODL modeling and enforcement supported only one router per L3 BGPVPN by design.

From Netvirt point of view, there are some limitations as well:

- We can not associate VPN port with both IPv4 and IPv6 Neutron Port addresses at the same time. Currently, any first Neutron Port IP address is using to create a VPN interface. If a Neutron Port possesses multiple IP Addresses, regardless of ethertype, this port might not work properly with ODL.
- It is not possible to associate a single L3 BGPVPN instance with two different routers.

Use Cases

There is no change in the use cases described in [6] and [7], except that the single L3 BGPVPN instance serves both IPv4 and IPv6 subnets.

Inter DC Access

1. **two-router** solution

IPv4 subnet Subnet A is added as a port in Router 1, IPv6 subnet Subnet B is added as a port in Router 2. The same L3 BGPVPN instance will be associated with both Router 1 and Router 2.

The L3 BGPVPN instance will distinguish ethertype of router ports and will create appropriate FIB entries associated to its own VPN entry, so IPv4 and IPv6 enries will be gathered in the same L3 BGPVPN.

2. **dualstack-router** solution

IPv4 subnet Subnet A is added as a port in Router 1, IPv6 subnet Subnet B is added as a port in Router 1 as well. L3 BGPVPN instance will be associated with Router 1.

The L3 BGPVPN instance will distinguish ethertype of routers ports and will create appropriate FIB entries associated to its own VPN entry as well. Appropriate BGP VRF context for IPv4 or IPv6 subnets will be also created.

External Internet Connectivity

External Internet Connectivity is not in the scope of this specification.

Proposed changes

All changes we can split in two main parts.

1. Distinguish IPv4 and IPv6 VRF tables with the same RD/iRT/eRT

    1.1 Changes in neutronvpn

    To support a pair of IPv4 and IPv6 prefixes for each launched dualstack VM we need to obtain information about subnets, where dualstack VM was spawned and information about extraroutes, enabled for these subnets. Obtained information will be stored in vmAdj and erAdjList objects respectively. These objects are attributes of created for new dualstack VM VPN interface. Created VPN port instance will be stored as part of already existed L3 BGPVPN node instance in MDSAL DataStore.

    When we update L3 BGPVPN instance node (associate/dissociated router or network), we need to provide information about ethertype of new attached/detached subnets, hence, Neutron Ports. New argument flags **ipv4On** and **ipv6On** will be introduced for that in **NeutronvpnManager** function API, called to update current L3 BGPVPN instance (*updateVpnInstanceNode()* method). *UpdateVpnInstanceNode()* method is also called, when we create a new L3 BGPVPN instance. So, to provide appropriate values for **ipv4On**, **ipv6On** flags we need to parse subnets list. Then in dependency of these flags values we will set either **Ipv4Family** attribute for the new L3 BGPVPN instance or **Ipv6Family** attribute, or both attributes. **Ipv4Family**, **Ipv6Family** attributes allow to create ipv4 or/and ipv6 VRF context for underlayed vpnmanager and bgpmanager APIs.

    1.2. Changes in vpnmanager

When L3 BGPVPN instance is created or updated, VRF tables must be created for QBGP as well. What we want, is to introduce separate VRF tables, created according to **IPv4Family/IPv6Family** VPN attributes, i.e. we want to distinguish IPv4 and IPv6 VRF tables, because this will bring flexibility in QBGP. For example, if QBGP receives an entry IPv6 MPLSVPN on a router, which is expecting to receive only IPv4 entries, this entry will be ignored. The same for IPv4 MPLSVPN entries respectively.

So, for creating **VrfEntry** objects, we need to provide information about L3 BGPVPN instance ethertype (**Ipv4Family/Ipv6Family** attribute), route distinguishers list, route imports list and route exports lists (**RD/iRT/eRT**). **RD/iRT/eRT** lists will be simply obtained from subnetworks, attached to the chosen L3 BGPVPN. Presence of **IPv4Family**, **IPv6Family** in VPN will be translated in following VpnInstanceListener class attributes: **afiIpv4**, **afiIpv6**, **safiMplsVpn**, **safiEvpn**, which will be passed to *addVrf()* and *deleteVrf()* bgpmanager methods for creating/deleting either **IPv4 VrfEntry** or **IPv6 VrfEntry** objects.

RD/iRT/eRT lists will be the same for both **IPv4 VrfEntry** and **IPv6 VrfEntry** in case, when IPv4 and IPv6 subnetworks are attached to the same L3 BGPVPN instance.

1.3 Changes in bgpmanager

In bgpmanager we need to change signatures of *addVrf()* and *deleteVrf()* methods, which will trigger signature changes of underlying API methods *addVrf()* and *delVrf()* from *BgpConfigurationManager* class.

This allows *BgpConfigurationManager* class to create needed IPv4 VrfEntry and IPv6 VrfEntry objects with appropriate **AFI** and **SAFI** values and finally pass this appropriate **AFI** and **SAFI** values to *BgpRouter*.

BgpRouter represents client interface for thrift API and will create needed IPv4 and IPv6 VRF tables in QBGP.

1.4 Changes in yang model

To support new attributes **AFI** and **SAFI** in bgpmanager classes, it should be added in ebgp.yang model:

```
list address-families {
 key "afi safi";
  leaf afi {
    type uint32;
    mandatory "true";
  }
  leaf safi {
    type uint32;
    mandatory "true";
  }
}
```

1.5 Changes in QBGP thrift interface

To support separate IPv4 and IPv6 VRF tables in QBGP we need to change signatures of underlying methods *addvrf()* and *delvrf()* in thrift API as well. They must include the address family and subsequent address families informations:

```
enum af_afi {
    AFI_IP = 1,
    AFI_IPV6 = 2,
}
```

(continues on next page)

(continued from previous page)

```
i32 addVrf(1:layer_type l_type, 2:string rd, 3:list<string>␣
↪irts, 4:list<string> erts,
           5:af_afi afi, 6:af_safi afi),
i32 delVrf(1:string rd, 2:af_afi afi, 3:af_safi safi)
```

2. Support of two routers, attached to the same L3 BGPVPN

   2.1 Changes in neutronvpn

   two-router solution assumes, that all methods, which are using to create, update, delete VPN interface or/and VPN instance must be adapted to a case, when we have a list of subnetworks and/or list of router IDs to attach. Due to this, appropriate changes need to be done in nvpnManager method APIs.

   To support **two-router** solution properly, we also should check, that we do not try to associate to L2 BGPVPN a router, that was already associated to that VPN instance. Attached to L3 BGPVPN router list must contain maximum 2 router IDs. Routers, which IDs are in the list must be only singlestack routers. More information about supported router configurations is available below in chapter "Configuration Impact".

   For each created in dualstack network Neutron Port we take only the last received IPv4 address and the last received IPv6 address. So we also limit a length of subnets list, which could be attached to a L3 BGPVPN instance, to two elements. (More detailed information about supported network configurations is available below in chapter "Configuration Impact".) Two corresponding **Subnetmap** objects will be created in *NeutronPortChangeListener* class for attached subnets. A list with created subnetmaps will be passed as argument, when *createVpnInterface* method will be called.

   2.2 Changes in vpnmanager

   VpnMap structure must be changed to support a list with router IDs. This change triggers modifications in all methods, which retry router ID from *VpnMap* object.

   VpnInterfaceManager structure must be also changed, to support a list of VPN instance name. So all methods, which gives VPN router ID from *VpnInterfaceManager* should be modified as well.

   As consequence, in operDS, a *VpnInterfaceOpDataEntry* structure is created, inherited from *VpnInterface* in configDS. While the latter structure has a list of VPN instance name, the former will be instantiated in operDS as many times as there are VPN instances. The services that were handling *VPNInterface* in operDS, will be changed to handle *VPNInterfaceOpDataEntry*. That structure will be indexed by InterfaceName and by VPNName. The services include natservice, fibmanager, vpnmanager, cloud service chain.

   Also, an augment structure will be done for *VPNInterfaceOpDataEntry* to contain the list of operational adjacencies. As for *VpnInterfaceOpDataEntry*, the new *AdjacenciesOp* structure will replace Adjacencies that are in operDS. Similarly, the services will be modified for that.

   Also, *VPNInterfaceOpDataEntry* will contain a *VPNInterfaceState* that stands for the state of the VPN Interface. Code change will be done to reflect the state of the interface. For instance, if VPNInstance is not ready, associated VPNInterfaceOpDataEntries will have the state changed to INACTIVE. Reversely, the state will be changed to ACTIVE.

   2.3 Changes in yang model

   To provide change in *VpnMap* and in *VpnInterfaceManager* structures, described above, we need to modify following yang files.

2.3.1 neutronvpn.yang

- Currently, container *vpnMap* holds one router-id for each L3 BGPVPN instance ID. A change consists in replacing one router-id leaf by a leaf-list of router-ids. Obviously, no more than two router-ids will be used.
- Container *vpnMaps* is used internally for describing a L3 BGPVPN. Change router-id leaf by router-ids leaf-list in this container is also necessary.

```
--- a/vpnservice/neutronvpn/neutronvpn-api/src/main/yang/
↪neutronvpn.yang
+++ b/vpnservice/neutronvpn/neutronvpn-api/src/main/yang/
↪neutronvpn.yang
@@ -1,4 +1,3 @@
-
module neutronvpn {

namespace "urn:opendaylight:netvirt:neutronvpn";
@@ -120,7 +119,7 @@ module neutronvpn {
Format is ASN:nn or IP-address:nn.";
}

-         leaf router-id {
+         leaf-list router-ids {
          type     yang:uuid;
          description "UUID router list";
      }
@@ -173,7 +172,7 @@ module neutronvpn {
description "The UUID of the tenant that will own the subnet.";
}

-             leaf router-id {
+             leaf-list router_ids {
              type     yang:uuid;
              description "UUID router list";
          }
```

2.3.2 l3vpn.yang

- Currently, list vpn-interface holds a leaf vpn-instance-name, which is a container for VPN router ID. A change consists in replacing leaf vpn-instance-name by a leaf-list of VPN router IDs, because L3 BGPVPN instance can be associated with two routers. Obviously, no more than two VPN router-IDs will be stored in leaf-list vpn-instance-name.

```
--- a/vpnservice/vpnmanager/vpnmanager-api/src/main/yang/
↪l3vpn.yang
+++ b/vpnservice/vpnmanager/vpnmanager-api/src/main/yang/
↪l3vpn.yang
        @@ -795,21 +795,21 @@

      list vpn-interface  {
        key "name";
        max-elements "unbounded";
        min-elements "0";
        leaf name {
          type leafref {
            path "/if:interfaces/if:interface/if:name";
```

(continues on next page)

(continued from previous page)

```
          }
        }
-       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
```

(continues on next page)

(continued from previous page)

```
            from vpn-name available as one of the keys.";

            type enumeration {
             enum active {
                value "0";
                description
                "Active state";
             }
             enum inactive {
                value "1";
                description
                "Inactive state";
             }
            }
            default "active";
        }
    }
}
```

Pipeline changes

There is no change in the pipeline, regarding the changes already done in [6] and [7].

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

The DC-GW has the information, that permits to detect an underlay destination IP and MPLS label for a packet coming from the Internet or from anotherr DC-GW.

Classifier Table (0) =>
LFIB Table (20) `match: tun-id=mpls_label set vpn-id=l3vpn-id, pop_mpls label, set output to nexthopgroup-dst-vm` =>
NextHopGroup-dst-vm: `set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

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

Classifier Table (0) =>
Lport Dispatcher Table (17) `match: LportTag l3vpn service: set vpn-id=l3vpn-id` =>
DMAC Service Filter (19) `match: dst-mac=router-internal-interface-mac l3vpn service: set vpn-id=l3vpn-id` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=ext-ipv4-address set tun-id=mpls_label output to MPLSoGRE tunnel port` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=ext-ipv6-address set tun-id=mpls_label output to MPLSoGRE tunnel port` =>

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

Configuration impact

1. Limitations for router configurations

   1.1 Maximum number of singlestack routers that can be associated to a L3BGPVPN is limited to 2. Maximum number of dualstack routers that can be associated with a BGPVPN is limited to 1.

   1.2 If a L3 BGPVPN has already associated with a one singlestack router and we try to associate this VPN instance again with a dualstack router, exception will not be raised. But this configuration will not be valid.

   1.3 If a singlestack router is already associated to a L3 BGPVPN instance, and it has more than one port and we try to add a port to this router with another ethertype, i.e. we try to make this router dualstack, exception will not be raised. But this configuration will not be valid and supported.

   1.4 When a different ethertype port is added to a singlestack router, which already has only one port and which is already associated to a L3 BGPVPN instance, singlestack router in this case becomes dualstack router with only two ports. This router configuration is allowed by current specification.

2. Limitations for subnetworks configurations

   2.1 Maximum numbers of different ethertype subnetworks associated to a one L3 BGPVPN instance is limited to two. If a network contains more than two different ethertype subnetworks, exception won't be raised, but this configuration isn't supported.

   2.2 When we associate a network with a L3 BGPVPN instance, we do not care if subnetworks from this network are ports in some routers and these routers were associated with other VPNs. This configuration is not considered as supported as well.

3. Limitations for number of IP addresses for a Neutron Port

The specification only targets dual-stack networks, that is to say with 1 IPv4 address and one IPv6 address only. For other cases, that is to say, adding subnetworks IPv4 or IPv6, will lead to undefined or untested use cases. The multiple subnets test case would be handled in a future spec.

ECMP impact

ECMP - Equal Cost multiple path.

ECMP feature is currently provided for Neutron BGPVPN networks and described in the specification [10]. 3 cases have been cornered to use ECMP feature for BGPVPN usability.

- ECMP of traffic from DC-GW to OVS (inter-DC case)
- ECMP of traffic from OVS to DC-GW (inter-DC case)
- ECMP of traffic from OVS to OVS (intra-DC case)

In each case, traffic begins either at DC-GW or OVS node. Then it is sprayed to end either at OVS node or DC-GW.

ECMP feature for Neutron BGPVPN networks was successfully (OK) tested with IPv4 L3 BGPVPN and IPv6 L3 BGPVPN (OK). the dual stack VM connectivity should embrace ECMP

We've included this chapter to remind, that code changes for supporting dualstack VMs should be tested against ECMP scenario as well.

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

None

Targeted Release

Carbon

Alternatives

None

Usage

Assume, that in the same provider network we have OpenStack installed with 1 controller and 2 compute nodes, DC-GW node and OpenDaylight node.

- create private tenant networks and subnetworks
  - create Network N;
  - declare Subnet A IPv4 for Network N;
  - declare Subnet B IPv6 for Network N;
  - create two ports in Network N;
  - each port will inherit a dual IP configuration.
- create routers
  - **two-router** solution + create two routers A and B, each router will be respectively connected to IPv4 and IPv6 subnets; + add subnet A as a port to router A; + add subnet B as a port to router B.
  - **dualstack-router** solution + create router A; + add subnet A as a port to router A; + add subnet B as a port to router A.
- Create MPLSoGRE tunnel between DPN and DCGW

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

- create the DC-GW VPN settings
  - Create a L3 BGPVPN context. This context will have the same settings as in [7].In dualstack case both IPv4 and IPv6 prefixes will be injected in the same L3 BGPVPN.
- create the ODL L3 BGPVPN settings
  - Create a BGP context. This step permits to start QBGP module depicted in [8] and [9]. ODL has an API, that permits interfacing with that external software. The BGP creation context handles the following:
    - start of BGP protocol;
    - declaration of remote BGP neighbor with the AFI/SAFI affinities. In our case, VPNv4 and VPNv6 address families will be used.
  - Create a L3 BGPVPN, this L3 BGPVPN will have a name and will contain VRF settings.
- associate created L3 BGPVPN to router
  - **two-router** solution: associate routers A and B with a created L3 BGPVPN;
  - **dualstack-router** solution: associate router A with a created L3 BGPVPN.
- Spawn a VM in a created tenant network:

  The VM will possess IPv4 and IPv6 addresses from subnets A and B.
- Observation: dump ODL BGP FIB entries

  At ODL node, we can dump ODL BGP FIB entries and we should see entries for both IPv4 and IPv6 subnets prefixes:

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

(continues on next page)

(continued from previous page)

```
    }
}
```

Features to Install

odl-netvirt-openstack

REST API

CLI

A new option `--afi` and `--safi` will be added to command `odl:bgp-vrf`:

```
odl:bgp-vrf --rd <> --import-rt <> --export-rt <> --afi <1|2> --safi <value> add|del
```

Implementation

Assignee(s)

Primary assignee: Philippe Guibert <philippe.guibert@6wind.com>

Other contributors:

- Valentina Krasnobaeva <valentina.krasnobaeva@6wind.com>
- Noel de Prandieres <prandieres@6wind.com>

Work Items

- QBGP Changes
- BGPManager changes
- VPNManager changes
- NeutronVpn changes

Dependencies

Quagga from 6WIND is available at the following urls:

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

Testing

Unit Tests

Some L3 BGPVPN testing may have be done. Complementary specification for other tests will be done.

Integration Tests

TBD

CSIT

Basically, IPv4 and IPv6 vpnservice functionality have to be validated by regression tests with a single BGPVRF.

CSIT specific testing will be done to check dualstack VMs connectivity with network configurations for **two-router** and **dualstack-router** solutions.

Two-router solution test suite:

1. Create 2 Neutron Networks NET_1_2RT and NET_2_2RT.

   1.1 Query ODL restconf API to check that both Neutron Network objects were successfully created in ODL.

   1.2 Update NET_1_2RT with a new description attribute.

2. In each Neutron Network create one Subnet IPv4 and one Subnet IPv6: SUBNET_V4_1_2RT, SUBNET_V6_1_2RT, SUBNET_V4_2_2RT, SUBNET_V6_2_2RT, respectively.

   2.1 Query ODL restconf API to check that all Subnetwork objects were successfully created in ODL.

   2.2 Update SUBNET_V4_2RT, SUBNET_V6_2RT with a new description attribute.

3. Create 2 Routers: ROUTER_1 and ROUTER_2.

   3.1 Query ODL restconf API to check that all Router objects were successfully created in ODL.

4. Add SUBNET_V4_1_2RT, SUBNET_V4_2_2RT to ROUTER_1 and SUBNET_V6_1_2RT, SUBNET_V6_2_2RT to ROUTER_2.

5. Create 2 security-groups: SG6_2RT and SG4_2RT. Add appropriate rules to allow IPv6 and IPv4 traffic from/to created subnets, respectively.

6. In network NET_1_2RT create Neutron Ports: PORT_11_2RT, PORT_12_2RT, attached with security groups SG6_2RT and SG4_2RT; in network NET_2_2RT: PORT_21_2RT, PORT_22_2RT, attached with security groups SG6_2RT and SG4_2RT.

   6.1 Query ODL restconf API to check, that all Neutron Port objects were successfully created in ODL.

   6.2 Update Name attribute of PORT_11_2RT.

7. Use each created Neutron Port to launch a VM with it, so we should have 4 VM instances: VM_11_2RT, VM_12_2RT, VM_21_2RT, VM_22_2RT.

   7.1 Connect to NET_1_2RT and NET_2_2RT dhcp-namespaces, check that subnet routes were successfully propagated.

   7.2 Check that all VMs have: one IPv4 address and one IPv6 addresses.

8. Check IPv4 and IPv6 VMs connectivity within NET_1_2RT and NET_2_2RT.

9. Check IPv4 and IPv6 VMs connectivity across NET_1_2RT and NET_2_2RT with ROUTER_1 and ROUTER_2.

   9.1 Check that FIB entries were created for spawned Neutron Ports.

   9.2 Check that all needed tables (19, 17, 81, 21) are presented in OVS pipelines and VMs IPs, gateways MAC and IP addresses are taken in account.

10. Connect to VM_11_2RT and VM_21_2RT and add extraroutes to other IPv4 and IPv6 subnets.

    10.1 Check other IPv4 and IPv6 subnets reachability from VM_11_2RT and VM_21_2RT.

11. Delete created extraroutes.
12. Delete and recreate extraroutes and check its reachability again.
13. Create L3VPN and check with ODL REST API, that it was successfully created.
14. Associate ROUTER_1 and ROUTER_2 with created L3VPN and check the presence of router IDs in VPN instance with ODL REST API.
15. Check IPv4 and IPv6 connectivity accross NET_1_2RT and NET_2_2RT with associated to L3VPN routers.

    15.1 Check with ODL REST API, that VMs IP addresses are presented in VPN interfaces entries.

    15.2 Verify OVS pipelines at compute nodes.

    15.3 Check the presence of VMs IP addresses in vrfTables objects with ODL REST API query.

16. Dissociate L3VPN from ROUTER_1 and ROUTER_2.
17. Delete ROUTER_1 and ROUTER_2 and its interfaces from L3VPN.
18. Try to delete router with NonExistentRouter name.
19. Associate L3VPN to NET_1_2RT.
20. Dissociate L3VPN from NET_1_2RT.
21. Delete L3VPN.
22. Create multiple L3VPN.
23. Delete multiple L3VPN.

Documentation Impact

Necessary documentation would be added if needed.

References

[1] OpenDaylight Documentation Guide

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

[3] http://docs.openstack.org/developer/networking-bgpvpn/overview.html

[4] Spec to support IPv6 North-South support for Flat/VLAN Provider Network.

[5] BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN

[6] Spec to support IPv6 DC to Internet L3VPN connectivity using BGPVPN

[7] Spec to support IPv6 Inter DC L3VPN connectivity using BGPVPN

[8] Zebra Remote Procedure Call

[9] Quagga BGP protocol

Listener Dependency Helper

https://git.opendaylight.org/gerrit/#/q/topic:ListenerDepedencyHelper

Listener Dependency Helper makes "Data Store Listeners" independent from dependency resolution.

Problem description

When a DataStore-Listener is fired with config add/update/delete event, as part of listener processing it may try to read the other data store objects, at times those datastore objects are not yet populated. In this scenario, listener event processing has to be delayed (or) discarded, as the required information is NOT entirely available. Later when the dependant data objects are available, this listener event will not be triggered again by DataStore.

This results in some events not getting processed resulting in possible data-path, bgp control and data plane failures.

Example: VpnInterface add() callback triggered by MD-SAL on vpnInterface add. While processing add() callback, the corresponding vpnInstance is expected to be present in MD-SAL operational DS; which means that vpnInstance creation is complete (updating the vpn-targets in Operational DS and BGP).

Information: vpnInstance Config-DS listener thread has to process vpnInstance creation and update vpnInstance in operational DS. vpnInstance creation listener callback is handled by different listener thread.

Use Cases

Use Case 1: VPNInterfaces may get triggered before VPNInstance Creation.

Current implementation: Delay based waits for handling VPNInterfaces that may get triggered before VPNInstance Creation(waitForVpnInstance()).

Use Case 2: VPNManager to handle successful deletion of VPN which has a large number of BGP Routes (internal/external):

Current implementation: Delay-based logic on VPNInstance delete in VPNManager (waitForOpRemoval()).

Use Case 3: VpnSubnetRouteHandler that may get triggered before VPNInstance Creation.

Current implementation: Delay based waits in VpnSubnetRouteHandler which may get triggered before VPNInstance Creation(waitForVpnInstance()).

Use Case 4: VPN Swaps (Internal to External and vice-versa)

Current implementation: Currently we support max of 100 VM's for swap (VpnInterfaceUpdateTimerTask, waitForFibToRemoveVpnPrefix()).

Proposed change

During Listener event call-back (AsyncDataTreeChangeListenerBase) from DataStore, check for pending events in "Listener-Dependent-Queue" with same InstanceIdentifier to avoid re-ordering.

Generic Queue Event Format:

key : Instance Identifier eventType : Type of event (ADD/UPDATE/DELETE) oldData : Data before modification (for Update event); newData : Newly populated data queuedTime : at which the event is queued to LDH. lastProcessedTime : latest time at which dependency list verified expiryTime : beyond which processing for event is useless waitBetweenDependencyCheckTime : wait time between each dependency check dependentIIDs : list of dependent InstanceIdentifiers retryCount : max retries allowed. databroker : data broker. deferTimerBased : flag to choose between (timer/listener based).

For Use Case - 1: deferTimerBased shall be set to TRUE (as per the specification).

During processing of events (either directly from DataStore or from "Listener-Dependent-Queue"), if there any dependent objects are yet to populated; queue them to "Listener-Dependent-Queue".

Expectations from Listener: Listener will push the callable instance to "Listener-Dependent-Queue" if it cannot proceed with processing of the event due to dependent objects/InstanceIdentifier and list of dependent IID's.

There are two approaches the Listener Dependency check can be verified.

> **approach-1** Get the list of dependent-IID's, query DataStore/Cache for 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 DSType> List), this shall return either Null list (or) the list which has dependencies yet to be resolved. In case Listener has local-cache implemented for set of dependencies, it can look at cache and identify. This api will be called from LDH-task-pool of thread(s).

instanceIdentifier is the MD-SAL key value which need to be verified for existence/non-existence of data. Boolean shouldDataExist: shall be TRUE, if the Listener expects to have the information exists in MD-SAL; False otherwise.

> **approach-2** Register Listener for wild-card path of IID's.

When a Listener gets queued to ""Listener-Dependent-Queue", LDH shall register itself as Listener for the dependent IID's (using wild-card-path/parent-node). Once the listener gets fired, identify the dependent listeners waiting for the Data. Once the dependent Listener is identified, if the dependent-IID list is NULL. Trigger listener for processing the event. LDH-task-pool shall unregister itself from wild-card-path/parent-node once there are no dependent listeners on child-nodes.

Re-Ordering

The following scenario, when re-ordering can happen and avoidance of the same:

Example: Key1 and Value1 are present in MD-SAL Data Store under Tree1, SubTree1 (for say). Update-Listener for Key1 is dependent on Dependency1.

Key1 received UPDATE event (UPDATE-1) with value=x, at the time of processing UPDATE-1, dependency is not available. So Listener Queued 'UPDATE-1' event to "UnProcessed-EventQueue". same key1 received UPDATE event (UPDATE-2) with value=y, at the time of processing UPDATE-2, dependency is available (Dependency1 is resolved), so it goes and processes the event and updates value of Key1 to y.

After WaitTime, event Key1, UPDATE-1 is de-queued from "UnProcessed-EventQueue" and put for processing in Lister. Listener processes it and updates the Key1 value to x. (which is incorrect, happened due to re-ordering of events).

To avoid reordering of events within listener, every listener call back shall peek into "UnProcessed-EventQueue" to identify if there exists a pending event with same key value; if so, either suppress (or) queue the event. Below are event ordering expected from MD-SAL and respective actions:

what to consider before processing the event to avoid re-ordering of events:

Current Event| Queued Event| Action
ADD | ADD | NOT EXPECTED
ADD | REMOVE | QUEUE THE EVENT
ADD | UPDATE | NOT EXPECTED
UPDATE | ADD | QUEUE EVENT
UPDATE | UPDATE | QUEUE EVENT
UPDATE | REMOVE | NOT EXPECTED
REMOVE | ADD | SUPPRESS BOTH
REMOVE | UPDATE | EXECUTE REMOVE SUPPRESS UPDATE
REMOVE | REMOVE | NOT EXPECTED

Pipeline changes

none

Yang changes

none

Configuration impact

none

Clustering considerations

In the two approaches mentioned: 1 - Timer: polling MD-SAL for dependency resolution may incur in more number of reads.

2 - RegisterListener: RegisterListener may some impact at the time of registering listener after which a notification message to cluser nodes.

Predifined List of Listeners

operational/odl-l3vpn:vpn-instance-op-data/vpn-instance-op-data-entry/* operational/odl-l3vpn:vpn-instance-op-data/vpn-instance-op-data-entry/ vpn-id/vpn-to-dpn-list/* config/l3vpn:vpn-instances/*

Other Infra considerations

Security considerations

none

Scale and Performance Impact

this infra, shall improve scaling of application without having to wait for dependent data store gets populated. Performance shall remain intact.

Targeted Release

Alternatives

- use polling/wait mechanisms

Features to Install

REST API

CLI

CLI will be added for debugging purpose.

Implementation

Assignee(s)

Primary assignee: Siva Kumar Perumalla (sivakumar.perumalla@ericsson.com)

Other contributors: Suneelu Verma K.

Work Items

Dependencies

Testing

Unit Tests

Integration Tests

CSIT

Documentation Impact

References

Acronyms

IID: InstanceIdentifier

Table of Contents

New SFC Classifier

https://git.opendaylight.org/gerrit/#/q/topic:new-sfc-classifier

The current SFC Netvirt classifier only exists in the old Netvirt. This blueprint explains how to migrate the old Netvirt classifier to a new Netvirt classifier.

Terminology

- NSH - Network Service Headers, used as Service Chaining encapsulation. NSH RFC Draft [1]
- NSI - Network Service Index, a field in the NSH header used to indicate the next hop
- NSP - Network Service Path, a field in the NSH header used to indicate the service chain
- RSP - Rendered Service Path, a service chain.
- SFC - Service Function Chaining. SFC RFC [2] ODL SFC Wiki [3].
- SF - Service Function
- SFF - Service Function Forwarder
- VXGPE - VXLAN GPE (Generic Protocol Encapsulation) Used as transport for NSH. VXGPE uses the same header format as traditional VXLAN, but adds a Next Protocol field to indicate NSH will be the next header. Traditional VXLAN implicitly expects the next header to be ethernet. VXGPE RFC Draft [4].

Problem description

In the Boron release, an SFC classifier was implemented, but in the old Netvirt. This blueprint intends to explain how to migrate the old Netvirt classifier to a new Netvirt classifier, which includes integrating the classifier and SFC with Genius.

The classifier is an integral part of Service Function Chaining (SFC). The classifier maps client/tenant traffic to a service chain by matching the packets using an ACL, and once matched, the classifier encapsulates the packets using some sort of Service Chaining encapsulation. Currently, the only supported Service Chaining encapsulation is NSH

using VXGPE as the transport. Very soon (possibly in the Carbon release) Vxlan will be added as another encapsulation/transport, in which case NSH is not used. The transport and encapsulation information to be used for the service chain is obtained by querying the Rendered Service Path (RSP) specified in the ACL action.

The transport and encapsulation used between the classifier and the SFF, and also between SFFs will be VXGPE+NSH. The transport and encapsulation used between the SFF and the SF will be Ethernet+NSH.

The following image details the packet headers used for Service Chaining encapsulation with VXGPE+NSH.

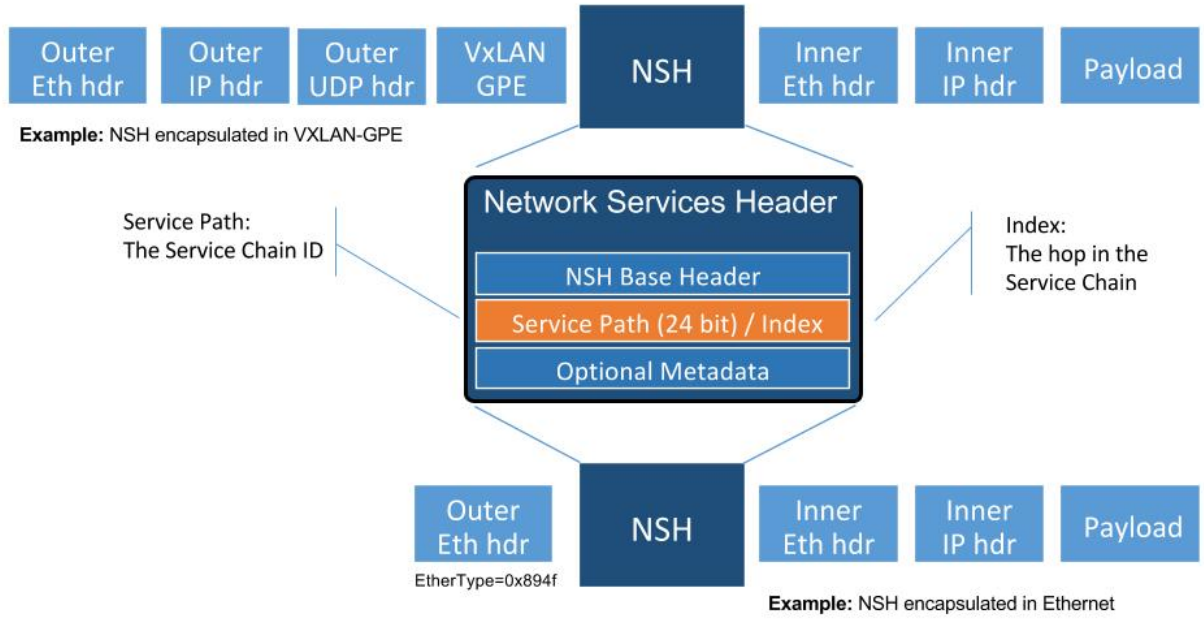


Diagram source [5].

The problem was originally discussed using the slides in this link [12] as a guideline. These slides are only intended for reference, and are not to be used for implementation.

Use Cases

The main use case addressed by adding an SFC classifier to Netvirt is to integrate SFC with Netvirt, thus allowing for Service Chaining to be used in an OpenStack virtual deployment, such as the OPNFV SFC project [6].

SFC works with both OVS and VPP virtual switches, and its even possible to have a hybrid setup whereby Netvirt is hosted on OVS and SFC is hosted on VPP switches. This blueprint only addresses the use of SFC with NetVirt and OVS.

As mentioned previously, currently SFC works with VXGPE+NSH and Eth+NSH transport/encapsulation, and soon SFC will work with VXLAN as the transport and encapsulation. The first version of this implementation will focus on VXGPE+NSH and Eth+NSH. In the future, when VXLAN is implemented in SFC, VXLAN can be added to the Netvirt SFC classifier. Changes in the transport and encapsulation used for service chains will have no affect on the Netvirt ACL model, since the transport and encapsulation information is obtained via the RSP specified in the RSP.

Proposed change

The existing old Netvirt SFC code can be found here:

- netvirt/openstack/net-virt-sfc/{api,impl}

Once the new Netvirt SFC classifier is implemented and working, the old Netvirt SFC classifier code will be left in place for at least one release cycle.

The new Netvirt SFC code base will be located here:

- netvirt/vpnservice/sfc/classifier/{api,impl}

The new Netvirt SFC classifier implementation will be new code. This implementation is not to be confused with the existing Netvirt aclservice, which is implemented for Security Groups. More details about the Genius integration can be found in the following section, but the Netvirt SFC classifier will be in a new Genius classifier service. The SFC implementation is already integrated with Genius and is managed via the Genius SFC service.

Integration with Genius

Genius [7], [8] is an OpenDaylight project that provides generic infrastructure services to other OpenDaylight projects. New Netvirt makes use of Genius and the new Netvirt classifier will also make use of Genius services. Among these services, the interface manager, tunnel manager and service binding services are of special relevance for the new Netvirt classifier.

Genius interface manager handles an overlay of logical interfaces on top of the data plane physical ports. Based on these logical interfaces, different services/applications may be bound to them with certain priority ensuring that there is no interference between them. Avoiding interference between services/applications is called Application Coexistence in Genius terminology. Typically, the effect of an application binding to a logical interface is that downstream traffic from that interface will be handed off to that application pipeline. Each application is then responsible to either perform a termination action with the packet (i.e output or drop action) or to return the packet back to Genius so that another application can handle the packet. There is a predefined set of types of services that can bind, and Classifier is one of them.

For OpenStack environments, Netvirt registers Neutron ports as logical interfaces in the Genius interface manager. Classifying traffic for a client/tenant ultimately relies on classifying traffic downstream from their corresponding Neutron ports. As such, the Netvirt classifier will bind on these interfaces as a newly defined Genius Classifier service through the Genius interface manager. It was considered integrating the Netvirt classifier with the existing Netvirt security groups, but the idea was discarded due to the possible conflicts and other complications this could cause.

Netvirt also keeps track of the physical location of these Neutron ports in the data plane and updates the corresponding Genius logical interface with this information. Services integrated with Genius may consume this information to be aware of the physical location of a logical interface in the data plane and it's changes when a VM migrates from one location to another. New Netvirt classifier will install the classification rules based on the data plane location of the client/tenant Neutron ports whose traffic is to be classified. On VM migration, the classifier has to remove or modify the corresponding classification rules accounting for this location change, which can be a physical node change or a physical port change.

The classifier is responsible for forwarding packets to the first service function forwarder (SFF) in the chain. This SFF may or may not be on the same compute host as the classifier. If the classifier and SFF are located on the same compute host, then the encapsulated packet is sent to the SFF via the Genius Dispatcher and OpenFlow pipelines. The packets can be forwarded to the SFF locally via the ingress or egress classifier, and it will most likely be performed by the egress classifier, but this decision will be determined at implementation time.

In scenarios where the first SFF is on a different compute host than the client node, the encapsulated packet needs to be forwarded to that SFF through a tunnel port. Tunnels are handled by the Genius tunnel manager (ITM) with an entity called transport zone: all nodes in a transport zone will be connected through a tunnel mesh. Thus the netvirt classifier needs to ensure that the classifier and the SFF are included in a transport zone. The transport type is also

specified at the transport zone level and for NSH it needs to be VXGPE. The classifier needs to make sure that this transport zone is handled for location changes of client VMs. Likewise, SFC needs to make sure the transport zone is handled for SF location changes.

The afore-mentioned Genius ITM is different than the tunnels currently used by Netvirt. SFC uses VXGPE tunnels, and requests they be created via the Genius ITM.

Classifier and SFC Genius Services

There will be 2 new Genius services created in Netvirt for the new Netvirt SFC classifier, namely an "Ingress SFC Classifier" and an "Egress SFC Classifier". There will also be a Genius service for the SFC SFF functionality that has already been created in the SFC project.

The priorites of the services will be as follows:

Ingress Dispatcher:

- SFC - P1
- IngressACL - P2
- Ingress SFC Classifier - P3
- IPv6, IPv4, L2 - P4...

Egress Dispatcher:

- EgressACL - P1
- Egress SFC Classifier - P2

The Ingress SFC classifier will bind on all the Neutron VM ports of the Neutron Network configured in the ACL. All packets received from these Neutron ports will be sent to the Ingress SFC classifier via the Genius Ingress Dispatcher, and will be subjected to ACL matching. If there is no match, then the packets will be returned to the Genius dispatcher so they can be sent down the rest of the Netvirt pipeline. If there is an ACL match, then the classifier will encapsulate NSH, set the NSP and NSI accordingly, initialize C1 and C2 to 0, and send the packet down the rest of the pipeline. Since the SFC service (SFF) will most likely not be bound to this same Neutron port, the packet wont be processed by the SFF on the ingress pipeline. If the classifier and first SFF are in the same node, when the packet is processed by the egress SFC classifier, it will be resubmitted back to the Ingress SFC service (SFC SFF) for SFC processing. If not, the packet will be sent to the first SFF.

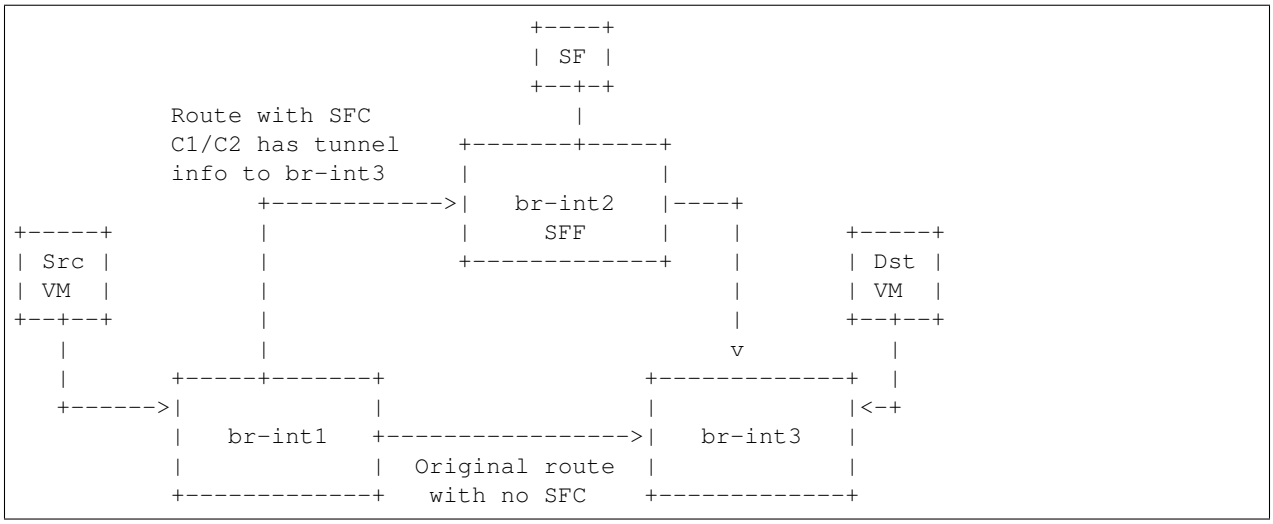
The Ingress SFC service (SFF) will bind on the Neutron ports for the Service Functions and on the VXGPE ports. The Ingress SFC service will receive packets from these Neutron and VXGPE ports, and also those that have been resubmitted from the Egress SFC Classifier. It may be possible that packets received from the SFs are not NSH encapsulated, so any packets received by the Ingress SFC service that are not NSH encapsulated will not be processed and will be sent back to the Ingress Dispatcher. For the NSH packets that are received, the Ingress SFC service will calculate the Next-Hop and modify either the VXGPE header if the next hop is a different SFF, or modify the Ethernet encapsulation header if the next hop is an SF on this same SFF. Once NSH packets are processed by the Ingress SFC service, they will be sent to the Egress Dispatcher.

The Egress SFC classifier service is the final phase of what the Ingress SFC classifier service started when an ACL match happens. The packet needed to go down the rest of the pipeline so the original packet destination can be calculated. The Egress SFC classifier will take the information prepared by the rest of the Netvirt pipeline and store the TunIPv4Dst and VNID of the destination compute host in C1 and C2 respectively. If the packet is not NSH encapsulated, then it will be sent back to the Egress Dispatcher. If the packet does have NSH encapsulation, then if C1/C2 is 0, then the fields will be populated as explained above. If the C1/C2 fields are already set, the packet will be sent out to either the Next Hop SF or SFF.

At the last hop SFF, when the packet egresses the Service Chain, the SFF will pop the NSH encapsulation and use the NSH C1 and C2 fields to tunnel the packet to its destination compute host. If the destination compute host is the same

as the last hop SFF, then the packet will be sent down the rest of the Netvirt pipeline so it can be sent to its destination VM on this compute host. When the destination is local, then the inport will probably have to be adjusted.

An example of how the last hop SFF routing works, imagine the following diagram where packet from the Src VM would go from br-int1 to br-int3 to reach the Dst VM when there is no service chaining employed. When the packets from the Src VM are subjected to service chaining, the pipeline in br-int1 need to calculate the the final destination is br-int3, and the appropriate information needs to be set in the NSH C1/C2 fields. Then the SFC SFF on br-int2, upon chain egress will use C1/C2 to send the packets to br-int3 so they can ultimately reach the Dst VM.



Pipeline changes

The existing Netvirt pipeline will not change as a result of adding the new classifier, other than the fact that the Ingress SFC classifier and Egress SFC classifier Genius Services will be added, which will change the Genius Service priorities as explained previously. The Genius pipelines can be found here [10].

Ingress Classifier Flows:

The following flows are an approximation of what the Ingress Classifier service pipeline will look like. Notice there are 2 tables defined as follows:

- **table 100: Ingress Classifier Filter table.**
  - Only allows Non-NSH packets to proceed in the classifier
- **table 101: Ingress Classifier ACL table.**
  - Performs the ACL classification, and sends packets to Ingress Dispatcher

The final table numbers may change depending on how they are assigned by Genius.

```
  // Pkt has NSH, send back to Ingress Dispatcher
cookie=0xf005ball00000101 table=100, n_packets=11, n_bytes=918,
    priority=550,nsp=42 actions=resubmit(,GENIUS_INGRESS_DISPATCHER_TABLE)

  // Pkt does NOT have NSH, send to GENIUS_INGRESS_DISPATCHER_TABLE
cookie=0xf005ball00000102 table=100, n_packets=11, n_bytes=918,
    priority=5 actions=goto_table:GENIUS_INGRESS_DISPATCHER_TABLE

  // ACL match: if TCP port=80
  // Action: encapsulate NSH and set NSH NSP, NSI, C1, C2, first SFF
```

(continues on next page)

(continued from previous page)

```
  // 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=0xf005ba1100000103, table=101, n_packets=11, n_bytes=918,
    tcp,tp_dst=80, in_port=10
    actions=push_nsh,
        load:0x1->NXM_NX_NSH_MDTYPE[],
        load:0x0->NXM_NX_NSH_C1[],
        load:0x0->NXM_NX_NSH_C2[],
        load:0x2a->NXM_NX_NSP[0..23],
        load:0xff->NXM_NX_NSI[],
        load:0x0a00010b->NXM_NX_REG0[],
        resubmit(,GENIUS_INGRESS_DISPATCHER_TABLE)
```

Egress Classifier Flows:

The following flows are an approximation of what the Egress Classifier service pipeline will look like. Notice there are 3 tables defined as follows:

- **table 221: Egress Classifier Filter table.**
  - Only allows NSH packets to proceed in the egress classifier
- **table 222: Egress Classifier NextHop table.**
  - Set C1/C2 accordingly
- **table 223: Egress Classifier TransportEgress table.**
  - Final egress processing and egress packets
  - Determines if the packet should go to a local or remote SFF

The final table numbers may change depending on how they are assigned by Genius.

```
  // If pkt has NSH, goto table 222 for more processing
cookie=0x14 table=221, n_packets=11, n_bytes=918,
    priority=260,md_type=1
    actions=goto_table:222

  // Pkt does not have NSH, send back to Egress Dispatcher
cookie=0x14 table=110, n_packets=0, n_bytes=0,
    priority=250
    actions=resubmit(,GENIUS_EGRESS_DISPATCHER_TABLE)


  // Pkt has NSH, if NSH C1/C2 = 0, Set C1/C2 and overwrite TunIpv4Dst
  // with SFF IP (Reg0) and send to table 223 for egress
cookie=0x14 table=222, n_packets=11, n_bytes=918,
    priority=260,nshc1=0,nshc2=0
    actions=load:NXM_NX_TUN_IPV4_DST[]->NXM_NX_NSH_C1[],
            load:NXM_NX_TUN_ID[]->NXM_NX_NSH_C2[],
            load:NXM_NX_REG0[]->NXM_NX_TUN_IPV4_DST[]
            goto_table:223

  // Pkt has NSH, but NSH C1/C2 aleady set,
  // send to table 223 for egress
cookie=0x14 table=222, n_packets=11, n_bytes=918,
    priority=250
```

(continues on next page)

(continued from previous page)

```
    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
```

(continues on next page)

(continued from previous page)

```
  // table 87 for further processing
cookie=0x14 table=86, n_packets=8, n_bytes=836,
    priority=5
    actions=goto_table:87

  // Match on specific NSH NSI/NSP, C1/C2 set
  // prepare pkt for egress, send to Egress Dispatcher
cookie=0xba5eba1100000101 table=87, n_packets=11, n_bytes=918,
        priority=650,nsi=255,nsp=42
        actions=move:NXM_NX_NSH_MDTYPE[]->NXM_NX_NSH_MDTYPE[],
                move:NXM_NX_NSH_NP[]->NXM_NX_NSH_NP[],
                move:NXM_NX_TUN_ID[0..31]->NXM_NX_TUN_ID[0..31],
                load:0x4->NXM_NX_TUN_GPE_NP[],
                resubmit(,GENIUS_EGRESS_DISPATCHER_TABLE)
```

Yang changes

The api YANGs used for the classifier build on the ietf acl models from the mdsal models.

Multiple options can be taken, depending on the desired functionality. Depending on the option chosen, YANG changes *might be* required.

Assuming no YANG changes, SFC classification will be performed on all VMs in the same neutron-network - this attribute is already present in the YANG model. **This is the proposed route**, since it hits a sweet-spot in the trade-off between functionality and risk.

If classifying the traffic from specific interfaces is desired, then the YANG model would need to be updated, possibly by adding a list of interfaces on which to classify.

Configuration impact

None

Clustering considerations

None

Other Infra considerations

Since SFC uses NSH, and the new Netvirt Classifier will need to add NSH encapsulation, a version of OVS that supports NSH must be used. NSH has not been officially accepted into the OVS project, so a branched version of OVS is used. Details about the branched version of OVS can be found here [9].

Security considerations

None

Scale and Performance Impact

None

Targeted Release

This change is targeted for the ODL Carbon release.

Alternatives

None

Usage

The new Netvirt Classifier will be configured via the REST JSON configuration mentioned in the REST API section below.

Features to Install

The existing old Netvirt SFC classifier is implemented in the following Karaf feature:

odl-ovsdb-sfc

When the new Netvirt SFC classifier is implemented, the previous Karaf feature will no longer be needed, and the following will be used:

odl-netvirt-sfc

REST API

The classifier REST API wont change from the old to the new Netvirt. The following example is how the old Netvirt classifier is configured.

Defined in netvirt/openstack/net-virt-sfc/api/src/main/yang/netvirt-acl.yang

An ACL is created which specifies the matching criteria and the action, which is to send the packets to an SFC RSP. Notice the "network-uuid" is set. This is for binding the Netvirt classifier service to a logical port. The procedure will be to query Genius for all the logical ports in that network uuid, and bind the Netvirt classifier service to each of them.

If the RSP has not been created yet, then the classification can not be created, since there wont be any information available about the RSP. In this case, the ACL information will be buffered, and there will be a separate listener for RSPs. When the referenced RSP is created, then the classifier processing will continue.

```
URL: /restconf/config/ietf-access-control-list:access-lists/

{
  "access-lists": {
    "acl": [
      {
        "acl-name": "ACL1",
        "acl-type": "ietf-access-control-list:ipv4-acl",
        "access-list-entries": {
          "ace": [
            {
              "rule-name": "ACE1",
              "actions": {
                "netvirt-sfc-acl:rsp-name": "RSP1"
              },
              "matches": {
                "network-uuid" : "eccb57ae-5a2e-467f-823e-45d7bb2a6a9a",
                "source-ipv4-network": "192.168.2.0/24",
                "protocol": "6",
                "source-port-range": {
                    "lower-port": 0
                },
                "destination-port-range": {
                    "lower-port": 80
                }
              }
            }
          ]
        }
      }]}}
```

CLI

None.

Implementation

Assignee(s)

Primary assignee:

- <brady.allen.johnson@ericsson.com>

Other contributors:

- <brady.allen.johnson@ericsson.com>
- <david.suarez.fuentes@ericsson.com
- <jaime.camaano.ruiz@ericsson.com>
- <miguel.duarte.de.mora.barroso@ericsson.com>

Work Items

Simple scenario:

- Augment the provisioned ACL with the 'neutron-network' augmentation - [11]
- From the neutron-network, get a list of neutron-ports - the interfaces connecting the VMs to that particular neutron-network. For each interface, do as follows:
  - Extract the DPN-ID of the node hosting the VM having that neutron-port
  - Extract the DPN-ID of the node hosting the first SF of the RSP
  - The forwarding logic to implement depends on the co-location of the client's VM with the first SF in the chain.
    - When the VMs are co-located (i.e. located in the same host), the output actions are to forward the packet to the first table of the SFC pipeline.
    - When the VMs are **not** co-located (i.e. hosted on different nodes) it is necessary to:
      - Use genius RPCs to get the interface connecting 2 DPN-IDs. This will return the tunnel endpoint connecting the compute nodes.
      - Use genius RPCs to get the list of actions to reach the tunnel endpoint.

Enabling VM mobility:

1. Handle first SF mobility

   Listen to RSP updates, where the only relevant migration is when the first SF moves to another node (different DPN-IDs). In this scenario, we delete the flows from the *old* node, and install the newly calculated flows in the new one. This happens for **each** node having an interface to classify attached to the provisioned neutron-network.

2. Handle client VM mobility

   Listen to client's InterfaceState changes, re-evaluating the Forwarding logic, since the tunnel interface used to reach the target DPN-ID is different. This means the action list to implement it, will also be different. The interfaces to listen to will be ones attached to the provisioned neutron-network.

3. **Must** keep all the nodes having interfaces to classify (i.e. nodes having neutron-ports attached to the neutron-network) and the first SF host node within the same transport zone. By listening to InterfaceState changes of clients within the neutron-network & the first SF neutron ports, the transport zone rendering can be redone.

   TODO: *is there a better way to identify when the transport zone needs to be updated?*

Dependencies

No dependency changes will be introduced by this change.

Testing

Unit Tests

Unit tests for the new Netvirt classifier will be modeled on the existing old Netvirt classifier unit tests, and tests will be removed and/or added appropriately.

Integration Tests

The existing old Netvirt Classifier Integration tests will need to be migrated to use the new Netvirt classifier.

CSIT

The existing Netvirt CSIT tests for the old classifier will need to be migrated to use the new Netvirt classifier.

Documentation Impact

User Guide documentation will be added by one of the following contributors:

- <brady.allen.johnson@ericsson.com>
- <david.suarez.fuentes@ericsson.com
- <jaime.camaano.ruiz@ericsson.com>
- <miguel.duarte.de.mora.barroso@ericsson.com>

References

[1] https://datatracker.ietf.org/doc/draft-ietf-sfc-nsh/

[2] https://datatracker.ietf.org/doc/rfc7665/

[3] https://wiki.opendaylight.org/view/Service_Function_Chaining:Main

[4] https://datatracker.ietf.org/doc/draft-ietf-nvo3-vxlan-gpe/

[5] https://docs.google.com/presentation/d/1kBY5PKPETEtRA4KRQ-GvVUSLbJoojPsmJlvpKyfZ5dU/edit?usp=sharing

[6] https://wiki.opnfv.org/display/sfc/Service+Function+Chaining+Home

[7] http://docs.opendaylight.org/en/stable-boron/user-guide/genius-user-guide.html

[8] https://wiki.opendaylight.org/view/Genius:Design_doc

[9] https://wiki.opendaylight.org/view/Service_Function_Chaining:Main#Building_Open_vSwitch_with_VxLAN-GPE_and_NSH_support

[10] http://docs.opendaylight.org/en/latest/submodules/genius/docs/pipeline.html

[11] https://github.com/opendaylight/netvirt/blob/master/openstack/net-virt-sfc/api/src/main/yang/netvirt-acl.yang

[12] https://docs.google.com/presentation/d/1gN8GnpVGwku4mp1on7EBZiE41RI7lZ-FFmFS2QlUTKk/edit?usp=sharing

Table of Contents

Netvirt Statistics

https://git.opendaylight.org/gerrit/#/q/topic:netvirt-counters

The feature enables getting statistics on ports and switches.

Problem description

Being able to ask for statistics, given as input Netvirt identifiers. It will enable filtering the results and having aggregated result. In a later stage, it will be also used to get element to element counters. Examples for possible filters: RX only, TX only, port + VLAN counters...

Use Cases

- Getting port counters, given its interface id (ietf interface name).
- Getting node counters, given its node id.

Port counters can be useful also to get statistics on traffic going into tunnels when requesting it from the tunnel endpoint port. In addition, there will also be support in aggregated results. For example: Getting the total number of transmitted packets from a given switch.

Proposed change

Adding a new bundle named "statistics-plugin" to Netvirt. This bundle will be responsible for converting the Netvirt unique identifiers into OpenFlow ones, and will get the relevant statistics by using OpenFlowPlugin capabilities. It will also be responsible of validating and filtering the results. It will be able to provide a wide range of aggregated results in the future.

Work flow description: Once a port statistics request is received, it is translated to a port statistics request from openflow plugin. Once the transaction is received, the data is validated and translated to a user friendly data. The user will be notified if a timeout occurs. In case of a request for aggregated counters, the user will receive a single counter result divided to groups (such as "bits", "packets"...). The counters in each group will be the sum of all of the matching counters for all ports. Neither one of the counter request nor the counter response will not be stored in the configuration database. Moreover, requests are not periodic and they are on demand only.

Pipeline changes

None

Yang changes

The new plugin introduced will have the following models:

```
grouping result {
  list counterResult {
      key id;
      leaf id {
          type string;
      }
      list groups {
```

(continues on next page)

(continued from previous page)

```
            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 {
```

(continues on next page)

(continued from previous page)

```
        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",
```

(continues on next page)

(continued from previous page)


```
            "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
```


(continues on next page)

(continued from previous page)


```
            },
            {
              "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
```


(continues on next page)

(continued from previous page)


```
            },
            {
              "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
```


(continues on next page)

(continued from previous page)


```
            },
            {
              "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": [
        {
```


(continues on next page)

(continued from previous page)


```
            "name": "Duration",
            "counters": [
              {
                "name": "durationNanoSecondCount",
                "value": 787000000
              },
              {
                "name": "durationSecondCount",
                "value": 693545
              }
            ]
          },
          {
            "name": "Bytes",
            "counters": [
              {
                "name": "bytesReceivedCount",
                "value": 1455
              },
              {
                "name": "bytesTransmittedCount",
                "value": 14151073
              }
            ]
          },
          {
            "name": "Packets",
            "counters": [
              {
                "name": "packetsReceivedCount",
                "value": 9
              },
              {
                "name": "packetsTransmittedCount",
                "value": 9
              }
            ]
          }
        ]
      }
    ]
  }
}
```

Features to Install

odl-netvirt-openflowplugin-genius-openstack

REST API

CLI

Implementation

Assignee(s)

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

Other contributors: TBD

Work Items

https://trello.com/c/ZdoLQWoV/126-netvirt-statistics

- Support port counters.
- Support node counters.
- Support aggregated results.
- Support filters on results.

Dependencies

- Genius
- OpenFlow Plugin
- Infrautils

Testing

Capture details of testing that will need to be added.

Unit Tests

Integration Tests

CSIT

Documentation Impact

References

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

Table of Contents

Neutron Port Allocation For DHCP Service

https://git.opendaylight.org/gerrit/#/q/topic:neutron_port_dhcp

This feature will enable the Neutron DHCP proxy service within controller to reserve and use a Neutron port per subnet for communication with Neutron endpoints.

Problem description

The DHCP service currently assumes availability of the subnet gateway IP address and its mac address for its DHCP proxy service, which may or may not be available to the controller. This can lead to service unavailability.

Problem - 1: L2 Deployment with 3PP gateway

There can be deployment scenario in which L2 network is created with no distributed Router/VPN functionality. This deployment can have a separate gateway for the network such as a 3PP LB VM, which acts as a TCP termination point and this LB VM is configured with a default gateway IP. It means all inter-subnet traffic is terminated on this VM which takes the responsibility of forwarding the traffic.

But the current DHCP proxy service in controller hijacks gateway IP address for serving DHCP discover/request messages. If the LB is up, this can continue to work, DHCP broadcasts will get hijacked by the ODL, and responses sent as PKT_OUTs with SIP = GW IP.

However, if the LB is down, and the VM ARPs for the same IP as part of a DHCP renew workflow, the ARP resolution can fail, due to which renew request will not be generated. This can cause the DHCP lease to lapse.

Problem - 2: Designated DHCP for SR-IOV VMs via HWVTEP

In this Deployment scenario, L2 network is created with no distributed Router/VPN functionality, and HWVTEP for SR-IOV VMs. DHCP flood requests from SR-IOV VMs (DHCP discover, request during bootup), are flooded by the HWVTEP on the ELAN, and punted to the controller by designated vswitch. DHCP offers are sent as unicast responses from Controller, which are forwarded by the HWVTEP to the VM. DHCP renews can be unicast requests, which the HWVTEP may forward to an external Gateway VM (3PP LB VM) as unicast packets. Designated vswitch will never receive these pkts, and thus not be able to punt them to the controller, so renews will fail.

High-Level Components:

The following components of the Openstack - ODL solution need to be enhanced to provide port allocation for DHCP service.

- Openstack ODL Mechanism Driver
- OpenDaylight Controller (NetVirt VpnService/DHCP Service/Elan Service)

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

Proposed change

The following components within OpenDaylight Controller needs to be enhanced:

- Neutron VPN module
- DHCP module
- ELAN and L3VPN modules

OpenDaylight controller needs to preserve a Neutron port for every subnet so that DHCP proxy service can be enabled in Openstack deployment. The Neutron port's device owner property is set to `network:dhcp` and uses this port for all outgoing DHCP messages. Since this port gets a distinct IP address and MAC address from the subnet, both problem-1 and problem-2 will be solved.

ODL Driver Changes:

ODL driver will need a config setting when ODL DHCP service is in use, as against when Neutron DHCP agent is deployed (Community ODL default setting). This needs to be enabled for ODL deployment

ODL driver will insert an async call in subnet create/update workflow in POST_COMMIT for subnets with DHCP set to 'enabled', with a port create request, with device owner set to `network:dhcp`, and device ID set to controller hostname/IP (from ml2_conf.ini file)

ODL driver will insert an async call in subnet delete, and DHCP 'disable' workflow to ensure the allocated port is deleted

ODL driver needs to ensure at any time no more than a single port is allocated per subnet for these requirements

Pipeline changes

For example, If a VM interface is having 30.0.0.1/de:ad:be:ef:00:05 as its Gateway (or) Router Interface IP/MAC address and its subnet DHCP neutron port is created with IP/MAC address 30.0.0.4/de:ad:be:ef:00:04. The ELAN pipeline is changed like below.

```
LPort Dispatcher Table (17)=>ELAN ARP Check Table(43) => ARP Responder Group (5000) =>
↪ ARP Responder Table (81) => Egress dispatcher Table(220)

cookie=0x8040000, duration=627.038s, table=17, n_packets=0, n_bytes=0, priority=6,␣
↪metadata=0xc019a00000000000/0xffffff0000000000 actions=write_
↪metadata:0xc019a01771000000/0xfffffffffffffffe,goto_table:43
cookie=0x1080000, duration=979.712s, table=43, n_packets=0, n_bytes=0, priority=100,
↪arp,arp_op=1 actions=group:5000
cookie=0x1080000, duration=979.713s, table=43, n_packets=0, n_bytes=0, priority=100,
↪arp,arp_op=2 actions=CONTROLLER:65535,resubmit(,48)
cookie=0x8030000, duration=979.717s, table=43, n_packets=0, n_bytes=0, priority=0␣
↪actions=goto_table:48
cookie=0x262219a4, duration=312.151s, table=81, n_packets=0, n_bytes=0, priority=100,
↪arp,metadata=0xc019a000000/0xfffffffff000000,arp_tpa=30.0.0.1,arp_op=1␣
↪actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:de:ad:be:ef:00:05->eth_
↪src,load:0x2->NXM_OF_ARP_OP[], move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_
↪ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xdeadbeef0005->NXM_NX_ARP_SHA[],load:0x1e000001->
↪NXM_OF_ARP_SPA[],load:0->NXM_OF_IN_PORT[],load:0x19a000->NXM_NX_REG6[],resubmit(,
↪220)
cookie=0x262219a4, duration=312.151s, table=81, n_packets=0, n_bytes=0, priority=100,
↪arp,metadata=0xc019a000000/0xfffffffff000000,arp_tpa=30.0.0.4,arp_op=1␣
↪actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:de:ad:be:ef:00:04->eth_
↪src,load:0x2->NXM_OF_ARP_OP[], move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_
↪ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xdeadbeef0004->NXM_NX_ARP_SHA[],load:0x1e000001->
↪NXM_OF_ARP_SPA[],load:0->NXM_OF_IN_PORT[],load:0x19a000->NXM_NX_REG6[],resubmit(,
↪220)
```

(continues on next page)

(continued from previous page)

```
group_id=5000,type=all,bucket=actions=CONTROLLER:65535,bucket=actions=resubmit(,48),
↪bucket=actions=resubmit(,81)
```

ARP Changes for DHCP port

1. Client VM ARP requests for DHCP server IP need to be answered in L2 as well as L3 deployment. 2. Create ARP responder table flow entry for DHCP server IP in computes nodes on which ELAN footprint is available. 3. Currently ARP responder is part of L3VPN pipeline, however no L3 service may be available in an L2 deployment to leverage the current ARP pipeline, for DHCP IP ARP responses. To ensure ARP responses are sent in L2 deployment, ARP processing needs to be migrated to the ELAN pipeline. 4. ELAN service to provide API to other services needing ARP responder entries including L3VPN service (for router MAC, router-gw MAC and floating IPs, and EVPN remote MAC entries). 5. ELAN service will be responsible for punting a copy of each ARP packet to the controller if the source MAC address is not already learned.

Assumptions

Support for providing port allocation for DHCP service is available from Openstack Pike release.

Reboot Scenarios

This feature support all the following Reboot Scenarios for EVPN:

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

Clustering considerations

The feature should operate in ODL Clustered environment reliably.

Other Infra considerations

N.A.

Security considerations

N.A.

Scale and Performance Impact

Not covered by this Design Document.

Targeted Release

Nitrogen, Carbon

Alternatives

Alternatives considered and why they were not selected.

Usage

Features to Install

This feature doesn't add any new karaf feature.

REST API

Implementation

The programming of flow rules in Table 43 and Table 81 is handled in ELAN module and following APIs are exposed from `IElanService` so that L3VPN and DHCP modules can use it to program ARP responder table flow entries for Gateway/Router Interface, floating IPs and DHCP port.

```
void addArpResponderEntry(BigIneger dpId, String ingressInterfaceName,
    String ipAddress, String macAddress, Optional<Integer> lportTag);
void removeArpResponderEntry(BigIneger dpId, String ingressInterfaceName,
    String ipAddress, String macAddress, Optional<Integer> lportTag);
```

A new container is introduced to hold the subnet DHCP port information.

Listing 42: dhcpservice-api.yang

```
     container subnet-dhcp-port-data {
         config true;
         list subnet-to-dhcp-port {
             key "subnet-id";
```

(continues on next page)

(continued from previous page)

```
            leaf subnet-id {
                type string;
            }
            leaf port-name {
                type string;
            }
            leaf port-fixedip {
                type string;
            }
            leaf port-macaddress {
                type string;
            }
        }
    }
```

When no DHCP port is available for the subnet we will flag an error to indicate DHCP service failure for virtual endpoints on such subnets which are dhcp-enabled in Openstack neutron.

Assignee(s)

Primary assignee: Karthik Prasad <karthik.p@altencalsoftlabs.com> Achuth Maniyedath <achuth.m@altencalsoftlabs.com> Vijayalakshmi CN <vijayalakshmi.c@altencalsoftlabs.com>

Other contributors: Dayavanti Gopal Kamath <dayavanti.gopal.kamath@ericsson.com> Vivekanandan Narasimhan <n.vivekanandan@ericsson.com> Periyasamy Palanisamy <periyasamy.palanisamy@ericsson.com>

Work Items

Dependencies

Testing

CSIT

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

Documentation Impact

This will require changes to User Guide and Developer Guide.

References

- OpenStack Spec - https://review.openstack.org/#/c/453160

Table of Contents

Policy based path selection for multiple VxLAN tunnels

https://git.opendaylight.org/gerrit/#/q/topic:policy-based-path-selection

The purpose of this feature is to allow selection of primary and backup VxLAN tunnels for different types of VxLAN encapsulated traffic between a pair of OVS nodes based on some predefined policy.

Egress traffic can be classified using different characteristics e.g. 5-tuple, ingress port+VLAN, service-name to determine the best available path when multiple VxLAN endpoints are configured for the same destination.

Problem description

Today, netvirt is not able to classify traffic and route it over different tunnel endpoints based on a set of predefined characteristics. This is an essential infrastructure for applications on top of netvirt offering premium and personalized services.

Use Cases

- Forwarding of VxLAN traffic between hypervisors with multiple physical/logical ports.

Proposed change

The current implementation of transport-zone creation generates vtep elements based on the `local_ip` definition in the `other-config` column of the Open_vSwitch schema where the `local_ip` value represents the tunnel interface ip. This feature will introduce a new `other-config` property `local_ips`. `local_ips` will express the association between multiple tunnel ip addresses and multiple underlay networks using the following format:

```
local_ips=<tun1-ip>:<underlay1-net>,<tun2-ip>:<underlay2-net>,..,<tunN-ip>:<underlayN-
↪net>
```

Upon transport-zone creation, if the `local_ips` configuration is present, full tunnel mesh will be created between all TEP ips in the same underlay network considering the existing transport-zone optimizations i.e. tunnels will be created only between compute nodes with at least one spawned VM in the same VxLAN network or between networks connected to the same router if at least one of the networks is VxLAN-based.

Note that configuration of multiple tunnel IPs for the same DPN in the same underlay network is not a supported as part of this feature and requires further enhancements in both ITM and the transport-zone model.

The underlay networks are logical entities that will be used to distigush between multiple uplinks for routing of egress VxLAN traffic. They have no relation to Openstack and neutron networks definition. A new yang module is introduced to model the association between different types of OVS egress VxLAN traffic and the selected underlay network paths to output the traffic.

Policy-based path selection will be defined as a new egress tunnel service and depends on tunnel service binding functionality detailed in [3].

The policy service will be bounded only for tunnels of type logical tunnel group defined in [2].

The service will classify different types of traffic based on a predefined set of policy rules to find the best available path to route each type of traffic. The policy model will be agnostic to the specific topology details including DPN ids, tunnel interface and logical interface names. The only reference from the policy model to the list of preferred paths is made using underlay network-ids described earlier in this document.

Each policy references an ordered set of `policy-routes`. Each `policy-route` can be a `basic-route` referencing single underlay-network or `route-group` composed of multiple underlay networks. This set will get

translated in each DPN to OF *fast-failover* group. The content of the buckets in each DPN depends on the existing underlay networks configured as part of the `local_ips` in the specific DPN.

The order of the buckets in the *fast-failover* group depends on the order of the underlay networks in the `policy-routes` model. `policy-routes` with similar set of routes in different order will be translated to different groups.

Each bucket in the *fast-failover* group can either reference a single tunnel or an additional OF *select* group depending on the type of policy route as detailed in the following table:

Policy route type	Bucket actions	OF Watch type
Basic route	load reg6(tun-lport) resubmit(220)	watch_port(tun-port)
Route group	goto_group(select-grp)	watch_group(select-grp)

This OF *select* group does not have the same content as the select groups defined in [2] and the content of its' buckets is based on the defined `route-group` elements and weights.

Logical tunnel will be bounded to the policy service if and only if there is at least one `policy-route` referencing one or more of the underlay networks in the logical group.

This service will take precedence over the default weighted LB service defined in [2] for logical tunnel group interfaces.

Policy-based path selection and weighted LB service pipeline example:

```
cookie=0x6900000, duration=0.802s, table=220, n_packets=0, n_bytes=0, priority=6,
↪reg6=0x500
actions=load:0xe000500->NXM_NX_REG6[],write_metadata:0xe000500000000000/
↪0xffffffff00000000,goto_table:230
cookie=0x6900000, duration=0.802s, table=220, n_packets=0, n_bytes=0, priority=6,
↪reg6=0xe000500
actions=load:0xf000500->NXM_NX_REG6[],write_metadata:0xf000500000000000/
↪0xffffffff00000000,group:800002
cookie=0x8000007, duration=0.546s, table=220, n_packets=0, n_bytes=0, priority=7,
↪reg6=0x600 actions=output:3
cookie=0x8000007, duration=0.546s, table=220, n_packets=0, n_bytes=0, priority=7,
↪reg6=0x700 actions=output:4
cookie=0x8000007, duration=0.546s, table=220, n_packets=0, n_bytes=0, priority=7,
↪reg6=0x800 actions=output:5
cookie=0x9000007, duration=0.546s, table=230, n_packets=0, n_bytes=0,priority=7,ip,
metadata=0x222e0/0xfffffffe,nw_dst=10.0.123.2,tp_dst=8080 actions=write_
↪metadata:0x200/0xfffffffe,goto_table:231
cookie=0x9000008, duration=0.546s, table=230, n_packets=0, n_bytes=0,priority=0,
↪resubmit(,220)
cookie=0x7000007, duration=0.546s, table=231, n_packets=0, n_bytes=0,priority=7,
↪metadata=0x500000000200/0xffffff00fffffffe,
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),
```

(continues on next page)

(continued from previous page)

```
bucket=weight:25,watch_port=4,actions=load:0x700->NXM_NX_REG6[],resubmit(,220),
bucket=weight:25,watch_port=5,actions=load:0x800->NXM_NX_REG6[],resubmit(,220)
```

Each bucket in the *fast-failover* group will set the `watch_port` or `watch_group` property to monitor the liveness of the OF port in case of `basic-route` and underlay group in case of `route-group`. This will allow the OVS to route egress traffic only to the first live bucket in each *fast-failover* group.

The policy model rules will be based on IETF ACL data model [4]. The following enhancements are proposed for this model to support policy-based path selection:

	Name	Attributes	Description	OF implementation
ACE matches	ingress-interface	name	Policy match based on the ingress port and optionally the VLAN id	Match lport-tag metadata bits
		vlan-id		
	service	service-type	Policy match based on the service-name of L2VPN/L3VPN e.g. ELAN name/VPN instance name	Match service/vrf-id metadata bits depending on the service-type
		service-name		
ACE actions	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;
       }
```

(continues on next page)

(continued from previous page)

```
        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";
```

(continues on next page)

(continued from previous page)

```
            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;
        }
```

(continues on next page)

(continued from previous page)

```
            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";
```

(continues on next page)

(continued from previous page)

```
            }
        }

        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;
```

(continues on next page)

(continued from previous page)

```
    }

    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
```

(continues on next page)

(continued from previous page)

```
      +--ro underlay-network-counters* [network-name]
         +--ro network-name     -> /underlay-networks/underlay-network/network-name
         +--ro dpn-counters* [dp-id]
         |  +--ro dp-id      uint64
         |  +--ro counter?   uint32
         +--ro path-counters* [source-dp-id destination-dp-id]
            +--ro source-dp-id         uint64
            +--ro destination-dp-id    uint64
            +--ro counter?             uint32
 augment /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:access-list-entries/ietf-
↪acl:ace/ietf-acl:matches:
   +--rw name?      if:interface-ref
   +--rw vlan-id?   ethertype:vlan-id
 augment /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:access-list-entries/ietf-
↪acl:ace/ietf-acl:matches:
   +--rw service-type?   identityref
   +--rw service-name?     string
 augment /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:access-list-entries/ietf-
↪acl:ace/ietf-acl:actions:
   +--rw policy-classifier?   -> /policy-profiles/policy-profile/policy-classifier
   +--rw direction?           identityref
```

Configuration impact

This feature introduces a new `other_config` parameter `local_ips` to support multiple ip:network associations as detailed above. Compatibility with the current `local_ip` parameter will be maintained but if both are present, `local_ips` would take presedence over `local_ip`.

Clustering considerations

None

Other Infra considerations

None

Security considerations

None

Scale and Performance Impact

None

Targeted Release

Carbon

Alternatives

None

Usage

Features to Install

odl-netvirt-openstack

REST API

Sample JSON data

Create policy rule

URL: restconf/config/ietf-access-control-list:access-lists

The following REST will create rule to classify all http traffic to ports 8080-8181 from specific vpn-id

```
{
  "access-lists": {
        "acl": [
          {
            "acl-type": "policy-service:policy-acl",
            "acl-name": "http-policy",
            "access-list-entries": {
              "ace": [
                {
                  "rule-name": "http-ports",
                  "matches": {
                    "protocol": 6,
                    "destination-port-range": {
                      "lower-port": 8080,
                      "upper-port": 8181
                    },
                    "policy-service:service-type": "l3vpn",
                    "policy-service:service-name": "71f7eb47-59bc-4760-8150-
↪e5e408d2ba10"
                  },
                  "actions": {
                    "policy-service:policy-classifier" : "classifier1",
```

(continues on next page)

(continued from previous page)

```
                    "policy-service:direction" : "egress"
                  }
                }
              ]
            }
          }
        ]
      }
    }
  }
```

Create underlay networks

URL: restconf/config/policy-service:underlay-networks

The following REST will create multiple underlay networks with different access types

```
{
  "underlay-networks": {
    "underlay-network": [
      {
        "network-name": "MPLS",
        "network-access-type": "policy-service:mpls-access-network"
      },
      {
        "network-name": "DLS1",
        "network-access-type": "policy-service:dsl-access-network"
      },
      {
        "network-name": "DSL2",
        "network-access-type": "policy-service:dsl-access-network"
      }
    ]
  }
}
```

Create underlay group

URL: restconf/config/policy-service:underlay-network-groups

The following REST will create group for the DSL underlay networks

```
{
  "underlay-network-groups": {
    "underlay-network-group": [
      {
        "group-name": "DSL",
        "underlay-network": [
          {
            "network-name": "DSL1",
            "weight": 75
          },
          {
            "network-name": "DSL2",
```

(continues on next page)

(continued from previous page)

```
            "weight": 25
          }
        ]
      }
    ]
  }
}
```

Create policy profile

URL: restconf/config/policy-service:policy-profiles

The following REST will create profile for classifier1 with multiple policy-routes

```
{
  "policy-profiles": {
    "policy-profile": [
      {
        "policy-classifier": "classifier1",
        "policy-route": [
          {
            "route-name": "primary",
            "network-name": "MPLS"
          },
          {
            "route-name": "backup",
            "group-name": "DSL"
          }
        ]
      }
    ]
  }
}
```

CLI

None

Implementation

Assignee(s)

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

Other contributors: Yair Zinger <yair.zinger@hpe.com>

Work Items

Trello card: https://trello.com/c/Uk3yrjUG/25-multiple-vxlan-endpoints-for-compute

- Transport-zone creation for multiple tunnels based on underlay network definitions
- Extract ACL flow programming to common location so it can be used by the policy service
- Create policy OF groups based on underlay network/group definitions
- Create policy classifier table based on ACL rules
- Create policy routing table
- Bind policy service to logical tunnels
- Maintain policy-route-counters per dpn/dpn-path

Dependencies

None

Testing

Unit Tests

Integration Tests

The test plan defined for CSIT below could be reused for integration tests.

CSIT

Adding multiple ports to the CSIT setups is challenging due to rackspace limitations. As a result, the test plan defined for this feature uses white-box methodology and not verifying actual traffic was sent over the tunnels.

Policy routing with single tunnel per access network type

- Set `local_ips` to contain tep ips for networks `underlay1` and `underlay2`
- Each underlay network will be defined with different `access-network-type`
- Create the following policy profiles
  - Profile1: `policy-classifier=clf1, policy-routes=underlay1, underlay2`
  - Profile2: `policy-classifier=clf2, policy-routes=underlay2, underlay1`
- Create the following policy rules
  - Policy rule 1: `dst_ip=vm2_ip,dst_port=8080 set_policy_classifier=clf1`
  - Policy rule 2: `src_ip=vm1_ip set_policy_classifier=clf2`
  - Policy rule 3: `service-type=l2vpn service-name=elan-name set_policy_classifier=clf1`

  - Policy rule 4: `service-type=l3vpn service-name=router-name set_policy_classifier=clf2`
  - Policy rule 5: `ingress-port=vm3_port set_policy_classifier=clf1`
  - Policy rule 6: `ingress-port=vm4_port vlan=vlan-id set_policy_classifier=clf2`
- Verify policy service flows/groups for all policy rules
- Verify flows/groups removal after the profiles were deleted

Policy routing with multiple tunnels per access network type

- Set `local_ips` to contain tep ips for networks `underlay1`..``underlay4``
- `underlay1`, `underlay2` and `underlay3`, `underlay4` are from the same `access-network-type`
- Create the following policy profiles where each route can be either group or basic route
  - Profile1: `policy-classifier=clf1, policy-routes={underlay1, underlay2}, {underlay3,underlay4}`
  - Profile2: `policy-classifier=clf2, policy-routes={underlay3,underlay4}, {underlay1, underlay2}`
  - Profile3: `policy-classifier=clf3, policy-routes=underlay1, {underlay3, underlay4}`
  - Profile4: `policy-classifier=clf4, policy-routes={underlay1, underlay2}, underlay3`
  - Profile5: `policy-classifier=clf5, policy-routes={underlay1, underlay2}`
  - Profile6: `policy-classifier=clf6, policy-routes=underlay4`
- Create the following policy rules
  - Policy rule 1: `dst_ip=vm2_ip,dst_port=8080 set_policy_classifier=clf1`
  - Policy rule 2: `src_ip=vm1_ip set_policy_classifier=clf2`
  - Policy rule 3: `service-type=l2vpn service-name=elan-name set_policy_classifier=clf3`
  - Policy rule 4: `service-type=l3vpn service-name=router-name set_policy_classifier=clf4`
  - Policy rule 5: `ingress-port=vm3_port set_policy_classifier=clf5`
  - Policy rule 6: `ingress-port=vm4_port vlan=vlan-id set_policy_classifier=clf6`
- Verify policy service flows/groups for all policy rules
- Verify flows/groups removal after the profiles were deleted

Documentation Impact

Netvirt documentation needs to be updated with description and examples of policy service configuration

References

[1] OpenDaylight Documentation Guide

[2] Load balancing and high availability of multiple VxLAN tunnels

[3] Service Binding On Tunnels

[4] Network Access Control List (ACL) YANG Data Model

Table of Contents

Support for QoS Alert

https://git.opendaylight.org/gerrit/#/q/topic:qos-alert

This feature adds support to monitor the per port packet drop counts when QoS rate limit rule is applied.

Problem description

If QoS bandwidth policy is applied on a neutron port, all packets exceeding the rate limit are dropped by the switch. This spec proposes a new service to monitor the packet drop ratio and log the alert message if packet drop ratio is greater than the configured threshold value.

Use Cases

Periodically monitor the port statistics of neutron ports having bandwidth limit rule and log an alert message in a log file if packet drop ratio cross the threshold value. Log file can be analyzed offline later to check the health/diagnostics of the network.

Proposed change

Proposed new service will use the RPC `/operations/opendaylight-direct-statistics:get-node-connector-sta` provided by openflowplugin to retrieve port statistics directly from switch by polling at regular interval. Polling interval is configurable with default value of 2 minutes.

Port packet drop ratio is calculated using delta of two port statistics counters `rx_dropped` and `rx_received` between the sample interval.

```
packet drop ratio = 100 * (rx_dropped / (rx_received + rx_dropped ))
```

An message is logged if packet drop ratio is greater than the configured threshold value.

Existing logging framework `log4j` shall be used to log the alert messages in the log file. A new appender `qosalertmsg` shall be added in `org.ops4j.pax.logging.cfg` to define the logging properties.

Log file format

```
2017-01-17 01:17:49,550 Packet drop threshold hit for qos policy qospolicy1 with qos-
↪id qos-2dbf02f6-dcd1-4c13-90ee-6f727e21fe8d for port port-3afde68d-1103-4b8a-a38d-
↪9cae631f7d67 on network network-563f9610-dd91-4524-ae23-8ec3c32f328e rx_received␣
↪4831 rx_dropped 4969
2017-01-17 01:17:49,550 Packet drop threshold hit for qos policy qospolicy2 with qos-
↪id qos-cb7e5f67-2552-4d49-b534-0ce90ebc8d97 for port port-09d3a437-f4a4-43eb-8655-
↪85df8bbe4793 on network network-389532a1-2b48-4ba9-9bcd-c1705d9e28f9 rx_received␣
↪3021 rx_dropped 4768
```

(continues on next page)

(continued from previous page)

```
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 43: 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";
```

(continues on next page)

(continued from previous page)

```
        }
        default 2;
      }

    }
}
```

Configuration impact

Following new parameters shall be made available as configuration. Initial or default configuration is specified in `netvirt-qosservice-config.xml`

Sl No.	configuration	Description
1.	`qos-alert-enabled`	configuration parameter to enable/disable the alerts
2.	`qos-drop-packet-threshold`	Drop percentage threshold configuration.
3.	`qos-alert-poll-interval`	Polling interval in minutes

Logging properties like log file name, location, size and maximum number of backup files are configured in file `org.ops4j.pax.logging.cfg`

Clustering considerations

In cluster setup, only one instance of qosalert service shall poll for port statistics. Entity owner service (EOS) shall be used to determine the owner of service.

Other Infra considerations

N.A.

Security considerations

None.

Scale and Performance Impact

QoS Alert Service minimizes scale and performance impact by following:

- Proposed service uses the direct-statistics RPC instead of OpenflowPlugin statistics-manager. This is lightweight because only node-connector statistics are queried instead of all statistics.
- Polling frequency is quite slow. Default polling interval is **two minutes** and minimum allowed value is 1 minute.

Targeted Release

Carbon.

Alternatives

N.A.

Usage

Features to Install

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

REST API

Put Qos Alert Config

Following API puts Qos Alert Config.

Method: POST

URI: /config/qosalert-config:qosalert-config

Parameters:

Parameter	Type	Value range	Comments
`qos-alert-enabled`	Boolean	true/false	Optional (default false)
`qos-drop-packet-threshold`	Uint16	1..100	Optional (default 5)
`qos-alert-poll-interval`	Uint16	1..65535	Optional time interval in minute(s) (default 2)

Example: .. code-block:: json

```
{
        "input": {
            "qos-alert-enabled": true,
            "qos-drop-packet-threshold": 35,
            "qos-alert-poll-interval": 5
        }
}
```

CLI

Following new karaf CLIs are added

```
qos:enable-qos-alert <true|false>

qos:drop-packet-threshold <threshold value in %>

qos:alert-poll-interval <polling interval in minutes>
```

Implementation

Assignee(s)

Primary assignee:

- Arun Sharma (arun.e.sharma@ericsson.com)

Other contributors:

- Ravi Sundareswaran (ravi.sundareswaran@ericsson.com)
- Mukta Rani (mukta.rani@tcs.com)

Work Items

Trello Link <https://trello.com/c/780v28Yw/148-netvirt-qos-alert>

1. Adding new yang file and listener.
2. Adding new `log4j appender` in odlparent `org.ops4j.pax.logging.cfg` file.
3. Retrieval of port statistics data using the openflowplugin RPC.
4. Logging alert message into the log file.
5. UT and CSIT

Dependencies

This doesn't add any new dependencies.

Testing

Capture details of testing that will need to be added.

Unit Tests

Standard UTs will be added.

Integration Tests

N.A.

CSIT

Following new CSIT tests shall be added

1. Verify that alerts are generated if drop packets percentage is more than the configured threshold value.
2. Verify that alerts are not generated if drop packets percentage is less than threshold value.
3. Verify that alerts are not generated when `qos-alert-enabled` if false irrespective of drop packet percentage.

Documentation Impact

This will require changes to User Guide.

User Guide will need to add information on how qosalert service can be used.

References

[1] Neutron QoS

[2] Spec for NetVirt QoS

[3] Openflowplugin port statistics

Table of Contents

Neutron Quality of Service API Enhancements for NetVirt

QoS patches: https://git.opendaylight.org/gerrit/#/q/topic:qos

The Carbon release will enhance the initial implementation of Neutron QoS API[1] support for NetVirt which was released in Boron. The Boron released added support for Neutron QoS policies and the Egress bandwidth rate limiting rule. The Carbon release will update the QoS feature set of NetVirt by providing support for the DSCP Marking rule and QoS Rule capability reporting.

Problem description

It is important to be able to configure QoS attributes of workloads on virtual networks. The Neutron QoS API provides a method for defining QoS policies and associated rules which can be applied to Neutron Ports and Networks. These rules include:

- Egress Bandwidth Rate Limiting
- DSCP Marking

(Note that for the Neutron API, the direction of traffic flow (ingress, egress) is from the perspective of the OpenStack instance.)

As a Neutron provider for ODL, NetVirt will provide the ability to report back to Neutron its QoS rule capabilties and provide the ability to configure and manage the supported QoS rules on supported backends (e.g. OVS, ...). The key changes in the Carbon release will be the addition of support for the DSCP Marking rule.

[1] Neutron QoS http://docs.openstack.org/developer/neutron/devref/quality_of_service.html

Use Cases

Neutron QoS API support, including:

- Egress rate limiting - Drop traffic that exceeeds the specified rate parameters for a Neutron Port or Network.
- DSCP Marking - Set the DSCP field for IP packets arriving from Neutron Ports or Networks.
- Reporting of QoS capabilities - Report to Neutron which QoS Rules are supported.

Proposed change

To handle DSCP marking, listener support will be added to the *neutronvpn* service to respond to changes in DSCP Marking Rules in QoS Policies in the Neutron Northbound QoS models[2][3] .

To implement DSCP marking support, a new ingress (from vswitch perspective) QoS Service is defined in Genius. When DSCP Marking rule changes are detected, a rule in a new OpenFlow table for QoS DSCP marking rules will be updated.

The QoS service will be bound to an interface when a DSCP Marking rule is added and removed when the DSCP Marking rule is deleted. The QoS service follows the DHCP service and precedes the IPV6 service in the sequence of Genius ingress services.

Some use cases for DSCP marking require that the DSCP mark set on the inner packet be replicated to the DSCP marking in the outer packet. Therefore, for packets egressing out of OVS through vxlan/gre tunnels the option to copy the DSCP bits from the inner IP header to the outer IP header is needed. Marking of the inner header is done via OpenFlow rules configured on the corresponding Neutron port as described above. For cases where the outer tunnel header should have a copy of the inner header DSCP marking, the `tos` option on the tunnel interface in OVSDB must be configured to the value `inherit`. The setting of the `tos` option is done with a configurable parameter defined in the ITM module. By default the `tos` option is set to *0* as specified in the OVSDB specification[4] .

On the creation of new tunnels, the `tos` field will be set to either the user provided value or to the default value, which may be controlled via configuration. This will result in the tunnel-options field in the IFM (Interface Manager) to be set which will in turn cause the `options` field for the tunnel interface on the OVSDB node to be configured.

To implement QoS rule capability reporting back towards Neutron, code will be added to the *neutronvpn* service to populate the operational qos-rule-types list in the Neutron Northbound Qos model[3] with a list of the supported QoS rules - which will be the bandwidth limit rule and DSCP marking rule for the Carbon release.

Pipeline changes

A new QoS DSCP table is added to support the new QoS Service:

Table	Match	Action
QoS DSCP [90]	Ethtype == IPv4 or IPv6 AND LPort tag	Mark packet with DSCP value

[2] Neutron Northbound QoS Model Extensions https://github.com/opendaylight/neutron/blob/master/model/src/main/yang/neutron-qos-ext.yang
[3] Neutron Northbound QoS Model https://github.com/opendaylight/neutron/blob/master/model/src/main/yang/neutron-qos.yang
[4] OVSDB Schema http://openvswitch.org/ovs-vswitchd.conf.db.5.pdf

Yang changes

A new leaf `option-tunnel-tos` is added to `tunnel-end-points` in *itm-state.yang* and to `vteps` in *itm.yang*.

Listing 44: itm-state.yang

```
list tunnel-end-points {
    ordered-by user;
    key "portname VLAN-ID ip-address tunnel-type";

    leaf portname {
        type string;
    }
    leaf VLAN-ID {
        type uint16;
    }
    leaf ip-address {
        type inet:ip-address;
    }
    leaf subnet-mask {
        type inet:ip-prefix;
    }
    leaf gw-ip-address {
        type inet:ip-address;
    }
    list tz-membership {
        key "zone-name";
        leaf zone-name {
            type string;
        }
    }
    leaf interface-name {
        type string;
    }
    leaf tunnel-type {
        type identityref {
            base odlif:tunnel-type-base;
        }
    }
    leaf option-of-tunnel {
        description "Use flow based tunnels for remote-ip";
        type boolean;
        default false;
    }
    leaf option-tunnel-tos {
        description "Value of ToS bits to be set on the encapsulating
            packet.  The value of 'inherit' will copy the DSCP value
            from inner IPv4 or IPv6 packets.  When ToS is given as
            and numberic value, the least significant two bits will
            be ignored. ";
        type string;
    }
}
```

Listing 45: itm.yang

```
list vteps {
    key "dpn-id portname";
    leaf dpn-id {
        type uint64;
    }
    leaf portname {
        type string;
    }
    leaf ip-address {
        type inet:ip-address;
    }
    leaf option-of-tunnel {
        description "Use flow based tunnels for remote-ip";
        type boolean;
        default false;
    }
    leaf option-tunnel-tos {
        description "Value of ToS bits to be set on the encapsulating
            packet.  The value of 'inherit' will copy the DSCP value
            from inner IPv4 or IPv6 packets.  When ToS is given as
            and numberic value, the least significant two bits will
            be ignored. ";
        type string;
    }
}
```

A configurable parameter `default-tunnel-tos` is added to *itm-config.yang* which defines the default ToS value to be applied to tunnel ports.

Listing 46: itm-config.yang

```
container itm-config {
    config true;

    leaf default-tunnel-tos {
        description "Default value of ToS bits to be set on the encapsulating
            packet.  The value of 'inherit' will copy the DSCP value
            from inner IPv4 or IPv6 packets.  When ToS is given as
            and numberic value, the least significant two bits will
            be ignored. ";
        type string;
        default 0;
    }
}
```

Configuration impact

A configurable parameter `default-tunnel-tos` is added to *genius-itm-config.xml* which specifies the default ToS to use on a tunnel if it is not specified by the user when a tunnel is created. This value may be set to `inherit` for some DSCP Marking use cases.

Listing 47: genius-itm-config.xml

```
<itm-config xmlns="urn:opendaylight:genius:itm:config">
    <default-tunnel-tos>0</default-tunnel-tos>
</itm-config>
```

Clustering considerations

None.

Other Infra considerations

None.

Security considerations

None.

Scale and Performance Impact

Additional OpenFlow packets will be generated to configure DSCP marking rules in response to QoS Policy changes coming from Neutron.

Targeted Release

Carbon

Alternatives

Use of OpenFlow meters was desired, but the OpenvSwitch datapath implementation does not support meters (although the OpenvSwitch OpenFlow protocol implementation does support meters).

Usage

The user will use the QoS support by enabling and configuring the QoS extension driver for networking-odl. This will allow QoS Policies and Rules to be configured for Neuetron Ports and Networks using Neutron.

Perform the following configuration steps:

- In *neutron.conf* enable the QoS service by appending `qos` to the `service_plugins` configuration:

Listing 48: /etc/neutron/neutron.conf

```
service_plugins = odl-router, qos
```

- Add the QoS notification driver to the *neutron.conf* file as follows:

Listing 49: /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 50: /etc/neutron/plugins/ml2/ml2.conf.ini

```
[ml2]
extensions_drivers = port_security,qos
```

Features to Install

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

- odl-netvirt-openstack

REST API

None.

CLI

Refer to the Neutron CLI Reference[5] for the Neutron CLI command syntax for managing QoS policies and rules for Neutron networks and ports.

[5] Neutron CLI Reference http://docs.openstack.org/cli-reference/neutron.html#neutron-qos-available-rule-types

Implementation

Assignee(s)

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

Primary assignee:

- Poovizhi Pugazh <poovizhi.p@ericsson.com>

Other contributors:

- Ravindra Nath Thakur <ravindra.nath.thakur@ericsson.com>
- Eric Multanen <eric.w.multanen@intel.com>
- Praveen Mala <praveen.mala@intel.com> (including CSIT)

Work Items

Task list in Carbon Trello: https://trello.com/c/bLE2n2B1/14-qos

Dependencies

Genius project - Code[6] to support QoS Service needs to be added.

Neutron Northbound - provides the Neutron QoS models for policies and rules (already done).

Following projects currently depend on NetVirt: Unimgr

Testing

Capture details of testing that will need to be added.

Unit Tests

Integration Tests

CSIT

Documentation Impact

Documentation to describe use of Neutron QoS support with NetVirt will be added.

OpenFlow pipeline documentation updated to show QoS service table.

[6] 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

https://git.opendaylight.org/gerrit/#/q/topic:SMAC_virt_endpoints

All L3 Routed packets destined to virtual endpoints in the datacenter managed by ODL do not carry a proper source-mac address in such frames put out to virtual endpoints.

This spec makes sure a proper source-mac is updated in the packet at the point where the packet is delivered to the VM, regardless of the tenant network type. On the actual datapath, there will be no change in the source mac-addresses and packets continue to use the same mechanism that is used today.

Addressing the datapath requires unique MAC allocation per OVS Datapath, so that it can be used as the source MAC for all distributively routed packets of an ODL enabled cloud. It would be handled in some future spec.

Problem description

Today all L3 Routed packets destined to virtual endpoints in the datacenter either

- Incorrectly carry the source mac-address of the originator (regardless of which network the originator is in)
- Incorrectly carry sometimes the reserved source mac address of 00:00:00:00:00:00

This spec is intended to setup a source-mac-address in the frame of L3 Routed packets just before such frames are directed into the virtual endpoints themselves. This enables use-cases where certain virtual endpoints which are VNFs in the datacenter that are source-mac conscious (or mandate that src-mac in frames be valid) can become functional on their instantiation in an OpenDaylight enabled cloud.

Use Cases

- Intra-Datacenter L3 forwarded packets within a hypervisor.
- Intra-Datacenter L3 forwarded packets over Internal VXLAN Tunnels between two hypervisors in the datacenter.
- Inter-Datacenter L3 forwarded packets :
  - Destined to VMs associated floating IP over External VLAN Provider Networks.
  - Destined to VMs associated floating IP over External MPLSOverGRE Tunnels.
  - SNAT traffic from VMs over External MPLSOverGRE Tunnels.
  - SNAT traffic from VMS over External VLAN Provider Networks.

Proposed change

All the L3 Forwarded traffic today reaches the VM via a LocalNextHopGroup managed by the VPN Engine (including FIBManager).

Currently the LocalNextHopGroup sets-up the destination MAC Address of the VM and forwards the traffic to EGRESS_LPORT_DISPATCHER_TABLE (Table 220). In that LocalNextHopGroup we will additionally setup source-mac-address for the frame. There are two cases to decide what source-mac-address should go into the frame:

- If the VM is on a subnet (on a network) for which a subnet gatewayip port exists, then the source-mac address of that subnet gateway port will be setup as the frame's source-mac inside the LocalNextHop group.This is typical of the case when a subnet is added to a router, as the router interface port created by neutron will be representing the subnet's gateway-ip address.

- If the VM is on a subnet (on a network), for which there is no subnet gatewayip port but that network is part of a BGPVPN , then the source-mac address would be that of the connected mac-address of the VM itself. The connected mac-address is nothing but the mac-address on the ovs-datapath for the VMs tapxxx/vhuxxx port on that hypervisor itself.

The implementation also applies to Extra-Routes (on a router) and Discovered Routes as they both use the LocalNextHopGroup in their last mile to send packets into their Nexthop VM.

We need to note that when a network is already part of a BGPVPN, adding a subnet on such a network to a router is disallowed currently by NeutronVPN. And so the need to swap the mac-addresses inside the LocalNextHopGroup to reflect the subnet gatewayip port here does not arise.

For all the use-cases listed in the USE-CASES section above, proper source mac address will be filled-up in the frame before it enters the virtual endpoint.

Pipeline changes

There are no pipeline changes.

The only change is in the NextHopGroup created by VPN Engine (i.e., VRFEntryListener). In the NextHopGroup we will additionally fill up the ethernet source mac address field with proper mac-address as outlined in the 'Proposed change' section.

Currently the LocalNextHopGroup is used in the following tables of VPN Pipeline:

- L3_LFIB_TABLE (Table 20) - Lands all routed packets from MPLSOverGRE tunnel into the virtual endpoint.
- INTERNAL_TUNNEL_TABLE (Table 36) - Lands all routed packets on Internal VXLAN Tunnel within the DC into the virtual end point.
- L3_FIB_TABLE (Table 21) - Lands all routed packets within a specific hypervisor into the virtual endpoint.

```
cookie=0x8000002, duration=50.676s, table=20, n_packets=0, n_bytes=0, priority=10,
↪mpls,mpls_label=70006 actions=write_actions(pop_mpls:0x0800,group:150000)
cookie=0x8000003, duration=50.676s, table=21, n_packets=0, n_bytes=0, priority=42,ip,
↪metadata=0x222f2/0xfffffffe,nw_dst=10.1.1.3 actions=write_actions(group:150000)
cookie=0x9011176, duration=50.676s, table=36, n_packets=0, n_bytes=0, priority=5,tun_
↪id=0x11176 actions=write_actions(group:150000)

NEXTHOP GROUP:
group_id=150000,type=all,bucket=actions=set_field:fa:16:3e:01:1a:40->eth_src,set_
↪field:fa:16:3e:8b:c5:51->eth_dst,load:0x300->NXM_NX_REG6[],resubmit(,220)
```

Yang changes

None.

Configuration impact

None.

Clustering considerations

None.

Other Infra considerations

None.

Security considerations

None.

Scale and Performance Impact

None

Targeted Release

Carbon/Boron

Alternatives

None.

Usage

N/A.

Features to Install

odl-netvirt-openstack

REST API

N/A.

CLI

N/A.

Implementation

Assignee(s)

Primary assignee:

- Achuth Maniyedath (achuth.m@altencalsoftlabs.com)

Other contributors:

- Karthik Prasad (karthik.p@altencalsoftlabs.com)
- Vivekanandan Narasimhan (n.vivekanandan@ericsson.com)

Work Items

https://trello.com/c/IfAmnFFr/110-add-source-macs-in-frames-for-l3-routed-packets-before-such-frames-get-to-the-virtual-endpoint

- Determine the smac address to be used for L3 packets forwarded to VMs.
- Update the LocalNextHopGroup table with proper ethernet source-mac parameter.

Dependencies

No new dependencies.

Testing

Verify the Source-MAC-Address setting on frames forwarded to Virtual endpoints in following cases.

Intra-Datacenter traffic to VMs (Intra/Inter subnet).

- VM to VM traffic within a hypervisor.
- VM to VM traffic across hypervisor over Internal VXLAN tunnel.

Inter-Datacenter traffic to/from VMs.

- External access to VMs using Floating IPs on MPLSOverGRE tunnels.
- External access to VMs using Floating IPs over VLAN provider networks.
- External access from VMs using SNAT over VLAN provider networks.
- External access from VMs using SNAT on MPLSOverGRE tunnels.

Unit Tests

N/A.

Integration Tests

N/A.

CSIT

- Validate that router-interface src-mac is available on received frames within the VM when that VM is on a router-arm.
- Validate that connected-mac as src-mac available on received frames within the VM when that VM is on a network-driven L3 BGPVPN.

Documentation Impact

N/A

References

N/A

Table of Contents

SR-IOV Hardware Offload for OVS

https://git.opendaylight.org/gerrit/#/q/topic:sriov-hardware-offload

This feature aims to support OVS hardware offload using SR-IOV technology. In Kernel 4.12 we introduced Traffic Control (TC see [1]) hardware offloads framework for SR-IOV VFs which allows us to configure the NIC [2]. Subsequent OVS patches [3] allows us to use the TC framework to offload OVS datapath rules. This feature is supported in OVS 2.8.0.

Problem description

Current ODL implementation supports bind vnic_type normal and keep track on the other neutron port types for DHCP usage. To support the OVS offload using SR-IOV ODL should support binding the direct vnic_type with hardware offloading ("switchdev") support.

Use Cases

As a cloud operator, I would like to leverage the OVS hardware offload feature to gain performance improvement.

Proposed change

In addition to the `normal` and `direct` ports we are introducing a port that supports hardware offloading. We refer to a port supporting HW offloading as a direct port with "switchdev" capability, as the NIC functions as an embedded switch device, routing packages according to the offloaded rules. A port is considered a "switchdev" port if the following applies:

1. The port's vnic_type is `direct`
2. The port contains additional information in `binding profile: '{"capabilities": ["switchdev"]}'`

3. The port is bound to a host that supports vnic_type direct This is validated by retrieving the hostconfig by the port's host id. This is done as extra validation. An example of A hostconfig supporting direct ports:

```
{"allowed_network_types": ["local", "flat", "vlan", "vxlan", "gre"],
 "bridge_mappings": {},
 "datapath_type": "system",
 "supported_vnic_types": [{"vif_type": "ovs",
                           "vnic_type": "normal",
                           "vif_details": {"support_vhost_user": false,
                                           "has_datapath_type_netdev": false,
                                           "uuid": "d8190c22-f6df-4236-964c-
↪9aa3544d1e4c",
                                           "host_addresses": ["my_host_name"]}},
                          {"vif_type": "ovs",
                           "vnic_type": "direct",
                           "vif_details": {"support_vhost_user": false,
                                           "has_datapath_type_netdev": false,
                                           "uuid": "d8190c22-f6df-4236-964c-
↪9aa3544d1e4c",
                                           "host_addresses": ["my_host_name"]}}]
}
```

Note that in order to validate the hostconfig the port must be bound to a hypervisor, this lead to workflow changes when creating ports of type switchdev.

Workflow

For SR-IOV legacy port - all workflows remain unchanged: On Neutron port create event we call handleNeutronPortCreate and update the subnet map to hold the port's IP in the relevant subnet id.

For Switchdev port:

- On Neutron port create event we ignore switchdev ports.
- On Neutron port update event we check if the port is bound to a host, if so we call handleNeutronPortCreate and update the subnet, in addition, we create an Openflow and Elan interfaces (just like a "normal" type port).

Pipeline changes

None

Yang changes

None

Configuration impact

None

Clustering considerations

None

Other Infra considerations

None

Security considerations

This feature does not support security groups.

Scale and Performance Impact

For every newly bound switchdev port, A DS read is executed to retrieve the host config.

Targeted Release

Oxygen.

Alternatives

None

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

None

CLI

The openstack port should be created as:

```
openstack port create --network private --vnic-type=direct --binding-profile '{
↪"capabilities": ["switchdev"]}' port1
```

Implementation

Assignee(s)

Primary assignee:

- Edan David (edand@mellanox.com)
- Moshe Levi (moshele@mellanox.com)

Work Items

Update ODL's NeutronPortChangeListener methods: handleNeutronPortCreated and handleNeutronPortDelete to allow adding/removing VF representor from the ovs pipeline in the following case: check that neutron port is vnic_type is direct and with binding:profile '{"capabilities": ["switchdev"]}'. Also, check the hostconfig allows binding the direct port see example:

```
{"vif_type": "ovs",
 "vnic_type": "direct",
 "vif_details": {"support_vhost_user": false,
                 "has_datapath_type_netdev": false,
                 "uuid": "d8190c22-f6df-4236-964c-9aa3544d1e4c",
                 "host_addresses": ["my_host_name"]}}
```

Dependencies

This feature has dependency on the v2 driver and pseudoagent port binding, And on commit: https://git.opendaylight.org/gerrit/#/c/65551/ fixing profile attribute handling in odl-neutton.

Testing

Unit Tests

Add test case for creating swichdev port.

Integration Tests

CSIT

Will be added in the future.

Documentation Impact

Update the documentation to provide explanation on the feature dependencies and hostconfig configuration.

References

[1] http://netdevconf.org/1.2/papers/efraim-gerlitz-sriov-ovs-final.pdf [2] https://patchwork.ozlabs.org/patch/738176/ [3] https://mail.openvswitch.org/pipermail/ovs-dev/2017-April/330606.html [4] https://specs.openstack.org/openstack/neutron-specs/specs/api/ports_binding_extended_attributes__ports_.html

Table of Contents

Support for compute node scale in and scale out

https://git.opendaylight.org/gerrit/#/q/topic:compute-scalein-scaleout

Add support for adding a new compute node into the existing topology and for removing or decomissioning existing compute node from topology.

Problem description

Support for adding a new compute node is already available. But when we scale in a compute node, we have to cleanup its relevant flows from openflow tables and cleanup the vxlan tunnel endpoints from other compute nodes. Also if the scaled in compute node is the designated compute for a particular service like nat or subnetroute etc , then those services have to choose a new compute node.

Use Cases

- Scale out of compute nodes.
- Scale in of single compute node
- Scale in of a bunch of compute nodes

Proposed change

The following are steps taken by administrator to achieve compute node scale in.

- The Nova Compute(s) shall be set into maintenance mode (nova service-disable <hostname> nova-compute).

This to avoid VM's to be scheduled to these Compute Hosts.

- Call a new rpc scalein-computes-start <list of scaledin compute node ids> to mark them as tombstoned.
- VMs still residing on the Compute Host(s), shall be migrated from the Compute Host(s).
- Disconnect the compute node from opendaylight controller node.
- Call a new rpc scalein-computes-end <list of scaledin compute node ids>

This is to signal the end of scale in process the following rpc will be invoked.

Incase vm migration or deletion from some of these compute nodes fails

The following recovery rpc will be invoked

scalein-compute-recover <list of not scaled in compute node ids which were passed as arg in scalein-computes-start>

Following is the typical sequence of operations.

scalein-computes-start A,B,C delete/migrate vms of A (success) delete/migrate vms of B (fail) delete/migrate vms of C (success) scalein-computes-end A,C scalein-computes-recover B

Typically When a single compute node gets scaled in as it gets disconnected from controller all the services who designated this compute as their designated compute would re-elect another compute node.

But when multiple compute nodes are getting scaled in during that window some of these computes should not be elected as designated compute.

To achieve that these scaled in computes are marked as tombstoned and they should be avoided when doing designated switch election or programming new services.

When we receive scalein-computes-end rpc call then corresponding computes config inventory and topology database also can be deleted.

When we receive scalein-computes-recover rpc call then corresponding computes tombstoned flag is set to false. If there are any services that do not have any compute node designated then they should start election of computes and possibly choose from these recovered computes.

Pipeline changes

None.

Yang changes

The following rpcs will be added.

Listing 51: scalein-api.yang

```
    rpc scalein-computes-start {
        description "To trigger start of scale in the given dpns";
        input {
            leaf-list scalein-node-ids {
                type string;
            }
        }
    }

    rpc scalein-computes-end {
        description "To end the scale in of the given dpns";
        input {
            leaf-list scalein-node-ids {
                type string;
            }
        }
    }

    rpc scalein-computes-recover {
        description "To recover the dpns which are marked for scale in";
        input {
            leaf-list recover-node-ids {
                type string;
            }
        }
    }
```

Topology node bridge-external-ids will be updated with additional key called “tombstoned”.

Configuration impact

None.

Clustering considerations

None.

Other Infra considerations

None.

Security considerations

None.

Scale and Performance Impact

None

Targeted Release

Oxygen

Alternatives

None.

Usage

N/A.

Features to Install

odl-netvirt-openstack

REST API

N/A.

CLI

N/A.

Implementation

Assignee(s)

Primary assignee:

- suneelu varma (k.v.suneelu.verma@ericsson.com)

Other contributors:

- Hanmanth (hanamantagoud.v.kandagal@ericsson.com)
- Chetan (chetan.arakere@altencalsoftlabs.com)

Work Items

TODO

Dependencies

No new dependencies.

Testing

- Verify that scaled out compute vms should be able to communicate with inter and intra compute vms.
- Verify that scale in compute flows be removed and existing service continue work.
- Verify that scale in compute nodes config inventory and topology datastores are cleaned.
- Identify a compute node which is designated for NAT/subnetroute functionality , scale in that compute, verify that NAT/subnetroute functionality continues to work. Verify that its relevant flows are reprogrammed.
- While the scale in work flow is going on for few computes, create a new NAT/subnetroute resource, make sure that one of these compute nodes are not chosen.
- Verify the recovery procedure of scale in workflow, make sure that the recovered compute gets its relevant flows.
- Scale in a compute which is designated and no other compute has presence of that service (vpn) to be designated, make sure that all its flows and datastores are deleted.
- Start scale in for a compute which is designated and no other compute has presence of that service (vpn) to be designated, recover the compute and make sure that all its flows and datastores are recovered.

Unit Tests

N/A.

Integration Tests

N/A.

CSIT

- Verify that scale out compute vms should be able to communicate with inter and intra compute vms.
- Verify that scale in compute flows be removed and existing service continue work.
- Identify a compute node which is designated for NAT/subnetroute functionality , scale in that compute, verify that NAT/subnetroute functionality continues to work. Verify that its relevant flows are reprogrammed.
- Verify the recovery procedure of scale in workflow, make sure that the recovered compute gets its relevant flows.

Documentation Impact

N/A

References

N/A

Table of Contents

Support for TCP MD5 Signature Option configuration of Quagga BGP

https://git.opendaylight.org/gerrit/#/q/topic:qbgp-tcp-md5-signature-option

This functionality adds support to odl-netvirt-impl feature to configure the TCP MD5 Signature Option [RFC2385] password in Quagga BGPs [QBGP].

Problem description

Quagga [QBGP] supports TCP MD5 Signature Option [RFC2385] in BGP traffic but current odl-netvirt-impl feature implementation lacks support to configure the required passwords.

Use Cases

UC1: Protect (Quagga [QBGP]) BGP and DC gateway BGP interface using TCP MD5 Signature Option [RFC2385].

Proposed change

The following components need to be enhanced:

- BGP Manager

Pipeline changes

No pipeline changes.

API changes

Changes will be needed in `ebgp.yang`, and `qbgp.thrift`.

YANG changes

A new optional leaf with the TCP MD5 Signature Option [RFC2385] password is added (by means of a choice) to list `neighbors`.

Listing 52: 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 53: ebgp.yang modifications

```
    list neighbors {
      key "address";
      leaf address {
        type inet:ipv4-address;
        mandatory "true";
      }
      leaf remote-as {
```

(continues on next page)

(continued from previous page)

```
        type uint32;
        mandatory "true";
      }
 +    use tcp-security-option-grouping;
```

Thrift changes

A new function `setPeerSecret` is added to the service `BgpConfigurator`.

Listing 54: 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).
```

(continues on next page)

(continued from previous page)

```
+     *
+     *   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 55: 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 56: etc/shiro.ini example

```
#
# DISCOURAGED since Carbon
#
/config/ebgp:bgp/neighbors/** = authBasic, roles[admin]
```

Listing 57: AAA MDSALDynamicAuthorizationFilter example

```
{ "aaa:policies":
   {  "aaa:policies": [
      {  "aaa:resource": "/restconf/config/ebgp:bgp/neighbors/**",
         "aaa:permissions": [
         {  "aaa:role": "admin",
            "aaa:actions": [ "get","post","put","patch","delete" ]
         } ]
      } ]
   }
}
```

If `BgpConfigurator` thrift service is not secured then `tcp-md5-signature-password` goes clear on the wire.

Quagga [QBGP] (up to version 1.0) keeps the password in memory in clear. The password can be retrieved through Quagga's configuration interface.

Scale and Performance Impact

Negligible scale or performance impacts.

- datastore: A bounded (<=80) string per configured neighbor.
- Traffic (thrift `BgpConfigurator` service): A bounded (<=80) string field per neighbor addition operation.

Targeted Release

Carbon

Alternatives

Three alternatives have been considered in order to avoid storing the plain password in datastore: RPC, post-update, and transparent encryption. They are briefly described below.

The best alternative is transparent encryption, but in Carbon time-frame is not feasible.

The post-update alternative does not actually solve the limitation.

The RPC alternative is feasible in Carbon time-frame but, given that currently `BgpConfigurator` thrift service is not secured, to add an RPC does not pull its weight.

RPC encryption

A new RPC `add-neighbor(address, as-number[, tcp-md5-signature-password])` is in charge of create `neighbors` elements. The password is salted and encrypted with `aaa-encryption-service`. Both the salt and the encrypted password are stored in the `neighbors` element.

Post-update encryption

The `neighbors` element contains both a `plain-password` leaf and a `encrypted-password-with-salt` leaf. The listener `BgpConfigurationManager.NeighborsReactor` is in charge of encrypt and remove the `plain-password` leaf when it is present (and the encrypted one is not).

This alternative does not really solve the limitation because during a brief period the password is stored in plain.

Transparent encryption

A plain value is provided in REST write operations but it is *automagically* encrypted before it reaches MD-SAL. Read operations never decrypts the encrypted values.

This alternative impacts at least `aaa`, `yangtools`, and `netconf` projects. It can not possibly be done in Carbon.

Usage

Features to Install

odl-netvirt-openstack

REST API

The RESTful API for neighbors creation (`/restconf/config/ebgp:bgp/neighbors/{address}`) will be enhanced to accept an additional `tcp-md5-signature-password` attribute:

```
{ "neighbors": {
   "address": "192.168.50.2",
   "remote-as": "2791",
   "tcp-md5-signature-password": "password"
}}
```

CLI

A new option `--tcp-md5-password` will be added to commands `odl:configure-bgp` and `odl:bgp-nbr`.

```
opendaylight-user@root> odl:configure-bgp -op add-neighbor --ip 192.168.50.2 --as-num␣
↪2791 --tcp-md5-password password
opendaylight-user@root> odl:bgp-nbr --ip-address 192.168.50.2 --as-number 2791 --tcp-
↪md5-password password add
```

Implementation

Assignee(s)

Primary assignee: Jose-Santos Pulido, JoseSantos, jose.santos.pulido.garcia@ericsson.com

Other contributors: TBD

Work Items

- https://trello.com/c/87MAFjRf

1. Spec
2. `ebgp.yang`
3. `BgpConfigurator` thrift service (both idl and client)
4. `BgpConfigurationManager.NeighborsReactor`
5. `ConfigureBgpCli`

Dependencies

Internal

No internal dependencies are added or removed.

External

To enable TCP MD5 Signature Option [RFC2385] in a BGP the following conditions need to be met:

- `BgpConfigurator` thrift service provider (e.g. `Zebra Remote Procedure Call` [ZRPC]) must support the new function `setPeerSecret`.
- BGP's TCP stack must support TCP MD5 Signature Option (e.g. in linux the kernel option CONFIG_TCP_MD5SIG must be set).

Testing

Unit Tests

Currently `bgpmanager` has no unit tests related to configuration.

Integration Tests

Currently `bgpmanager` has no integration tests.

CSIT

Currently there is no CSIT test exercising `bgpmanager`.

Documentation Impact

Currently there is no documentation related to `bgpmanager`.

References

Table of Contents

Support of VXLAN based L2 connectivity across Datacenters

https://git.opendaylight.org/gerrit/#/q/topic:EVPN_RT2

Enable realization of L2 connectivity over VXLAN tunnels using L2 BGPVPNs, internally taking advantage of EVPN as the BGP Control Plane mechanism.

Problem description

OpenDaylight NetVirt service today supports L3VPN connectivity over VXLAN tunnels. L2DCI communication is not possible so far.

This spec attempts to enhance the BGPVPN service in NetVirt to embrace inter-DC L2 connectivity over external VXLAN tunnels.

In scope

The scope primarily includes providing ability to support intra-subnet connectivity across DataCenters over VXLAN tunnels using BGP EVPN with type L2.

When we mention that we are using EVPN BGP Control plane, this spec proposes using the RouteType 2 as the primary means to provision the control plane to enable inter-DC connectivity over external VXLAN tunnels.

With this inplace we will be able to support the following.

- Intra-subnet connectivity across dataCenters over VXLAN tunnels.

The following are already supported as part of the other spec(RT5) and will continue to function.

- Intra-subnet connectivity within a DataCenter over VXLAN tunnels.
- Inter-subnet connectivity within a DataCenter over VXLAN tunnels.
- Inter-subnet connectivity across dataCenters over VXLAN tunnels.

Out of scope

Use Cases

The following high level use-cases will be realized by the implementation of this Spec.

Datacenter access from another Datacenter over WAN via respective DC-Gateways (L2 DCI)

This use-case involves providing intra-subnet connectivity between two DataCenters. Tenant VMs in one datacenter will be able to communicate with tenant VMs on the other datacenter provided they are part of the same BGP EVPN and they are on same subnets.

The dataplane between the tenant VMs themselves and between the tenant VMs towards the DC-Gateway will be over VXLAN Tunnels.

The dataplane between the DC-Gateway to its other WAN-based BGP Peers is transparent to this spec. It is usually MPLS-based EPVPN.

The BGP Control plane between the ODL Controller and the DC-Gateway will be via EVPN RouteType 2 as defined in EVPN_RT2.

The control plane between the DC-Gateway and it other BGP Peers in the WAN is transparent to this spec, but can be EVPN IP-MPLS.

In this use-case:

1. We will have only a single DCGW for WAN connectivity
2. MAC IP prefix exchange between ODL controller and DC-GW (iBGP) using EVPN RT2
3. WAN control plane may use EVPN IP-MPLS for route exchange.
4. On the DC-Gateway, the VRF instance will be configured with two sets of import/export targets. One set of import/export route targets belong to EVPN inside DataCenter (realized using EVPN RT2) and the second set of import/export route target belongs to WAN control plane.
5. EVPN single homing to be used in all RT2 exchanges inside the DataCenter i.e., ESI=0 for all prefixes sent from DataCenter to the DC-Gateway.

Proposed change

The following components of an Openstack-ODL-based solution need to be enhanced to provide intra-subnet and inter-subnet connectivity across DCs using EVPN MAC IP Advertisement (Route Type 2) mechanism (refer EVPN_RT2):

- Openstack Neutron BGPVPN Driver
- OpenDaylight Controller (NetVirt)
- BGP Quagga Stack to support EVPN with RouteType 2 NLRI
- DC-Gateway BGP Neighbour that supports EVPN with RouteType 2 NLRI

The changes required in Openstack Neutron BGPVPN Driver and BGP Quagga Stack are captured in the Solution considerations section down below.

Pipeline changes

INTRA DC

Intra Subnet, Local DPN: VMs on the same subnet, same VPN, same DPN

There are no explicit pipeline changes for this use-case.

Intra Subnet, Remote DPN: VMs on two different DPNs, both VMs on the same subnet and same VPN

There are no explicit pipeline changes for this use-case.

Inter Subnet, Local DPN: VMs on different subnet, same VPN, same DPN

There are no explicit pipeline changes for this use-case.

Inter Subnet, Local DPN: VMs on different subnet, same VPN, same DPN

There are no explicit pipeline changes for this use-case.

INTER DC

Intra subnet Traffic from DC-Gateway to Local DPN

Classifier table (0) =>
Dispatcher table (17) `match: tunnel-type=vxlan` =>
L2VNI_EXTERNAL_TUNNEL_DEMUX_TABLE (24) => `match tunnel-id=l2vni, set elan-tag`
ELAN DMAC table (51) `match: elan-tag=vxlan-net-tag,dst-mac=vm2-mac set reg6=vm-lport-tag` =>
Egress table (220) `match: reg6=vm-lport-tag output to vm port`

Intra subnet Traffic from Local DPN to DC-Gateway

Classifier table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
GW Mac table (19) =>
Dispatcher table (17) `l2vpn service: set elan-tag=vxlan-net-tag` =>
ELAN base table (48) =>
ELAN SMAC table (50) `match: elan-tag=vxlan-net-tag,src-mac=vm1-mac` =>
ELAN DMAC table (51) `match: elan-tag=vxlan-net-tag,dst-mac=external-vm-mac set tun-id=vxlan-net-tag group=next-hop-group`
Next Hop Group `bucket0 :set reg6=tunnel-lport-tag bucket1 :set reg6=tunnel2-lport-tag`

Egress table (220) `match: reg6=tunnel2-lport-tag` output to `tunnel2`

Inter subnet Traffic from Local DPN to DC-Gateway (Symmetric IRB)

Classifier Table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set tun-id=l3vni output to nexthopgroup` =>
NextHopGroup: `set-eth-dst router-gw-vm, reg6=tunnel-lport-tag` =>
Lport Egress Table (220) `Output to tunnel port`

Inter subnet Traffic from DC-Gateway to Local DPN (Symmetric IRB)

Classifier table (0) =>
Dispatcher table (17) `match: tunnel-type=vxlan` =>
L3VNI_EXTERNAL_TUNNEL_DEMUX_TABLE (23) => `match tunnel-id=l3vni, set l3vpn-id` =>
L3 Gateway MAC Table (19) => `match dst-mac=vpn-subnet-gateway-mac-address` =>
FIB table (21) `match: l3vpn-tag=l3vpn-id,dst-ip=vm2-ip set reg6=vm-lport-tag goto=local-nexthop-group` =>
local nexthop group `set dst-mac=vm2-mac table=220` =>
Egress table (220) `match: reg6=vm-lport-tag output to vm port`

Inter subnet Traffic from Local DPN to DC-Gateway (ASymmetric IRB)

Classifier Table (0) =>
Dispatcher table (17) `l3vpn service: set vpn-id=router-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set tun-id=l2vni output to nexthopgroup` =>
NextHopGroup: `set-eth-dst dst-vm-mac, reg6=tunnel-lport-tag` =>
Lport Egress Table (220) `Output to tunnel port`

Intra subnet Traffic from DC-Gateway to Local DPN (ASymmetric IRB)

Classifier table (0) =>
Dispatcher table (17) `match: tunnel-type=vxlan` =>
L2VNI_EXTERNAL_TUNNEL_DEMUX_TABLE (24) => `match tunnel-id=l2vni, set elan-tag`
ELAN DMAC table (51) `match: elan-tag=vxlan-net-tag,dst-mac=vm2-mac set reg6=vm-lport-tag` =>
Egress table (220) `match: reg6=vm-lport-tag output to vm port`

ARP Pipeline changes

Local DPN: VMs on the same subnet, same DPN

a. Introducing a new Table aka ELAN_ARP_SERVICE_TABLE (Table 81). This table will be the first table in elan pipeline.

Classifier table (0) =>
Dispatcher table (17) `elan service: set elan-id=vxlan-net-tag` =>
Arp Service table (81) => `match: arp-op=req, dst-ip=vm-ip, ela-id=vxlan-net-tag inline arp reply`

Intra Subnet, Local DPN: VMs on the same subnet, on remote DC

Classifier table (0) =>
Dispatcher table (17) `elan service: set elan-id=vxlan-net-tag` =>
Arp Service table (81) => `match: arp-op=req, dst-ip=vm-ip, ela-id=vxlan-net-tag inline arp reply`

Yang changes

Changes will be needed in `l3vpn.yang` , `odl-l3vpn.yang` , `odl-fib.yang` and `neutronvpn.yang` to start supporting EVPN functionality.

ODL-L3VPN YANG changes

A new container evpn-rd-to-networks is added This holds the rd to networks mapping This will be useful to extract in which elan the received RT2 route can be injected into.

Listing 58: 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 59: 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{
```

(continues on next page)

(continued from previous page)

```
    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 60: 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;
```

(continues on next page)

(continued from previous page)

```
            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 61: 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;
            }
        }
```

(continues on next page)

(continued from previous page)

```
    }
}

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

https://git.opendaylight.org/gerrit/#/q/topic:EVPN_RT5

Enable realization of L3 connectivity over VXLAN tunnels using L3 BGPVPNs, internally taking advantage of EVPN as the BGP Control Plane mechanism.

Problem description

OpenDaylight NetVirt service today supports VLAN-based, VXLAN-based connectivity and MPLSOverGRE-based overlays.

In this VXLAN-based underlay is supported only for traffic within the DataCenter. For all the traffic that need to go via the DC-Gateway the only supported underlay is MPLSOverGRE.

Though there is a way to provision an external VXLAN tunnel via the ITM service in Genius, the BGPVPN service in NetVirt does not have the ability to take advantage of such a tunnel to provide inter-DC connectivity.

This spec attempts to enhance the BGPVPN service (runs on top of the current L3 Forwarding service) in NetVirt to embrace inter-DC L3 connectivity over external VXLAN tunnels.

In scope

The scope primarily includes providing ability to support Inter-subnet connectivity across DataCenters over VXLAN tunnels by modeling a new type of L3VPN which will realize this connectivity using EVPN BGP Control plane semantics.

When we mention that we are using EVPN BGP Control plane, this spec proposes using the RouteType 5 explained in EVPN_RT5 as the primary means to provision the control plane en enable inter-DC connectivity over external VXLAN tunnels.

This new type of L3VPN will also inclusively support:

- Intra-subnet connectivity within a DataCenter over VXLAN tunnels.
- Inter-subnet connectivity within a DataCenter over VXLAN tunnels.

Out of scope

- Does not cover providing VXLAN connectivity between hypervisors (with OVS Datapath) and Top-Of-Rack switches that might be positioned within such DataCenters.
- Does not cover providing intra-subnet connectivity across DCs.

Both the points above will be covered by another spec that will be Phase 2 of realizing intra-subnet inter-DC connectivity.

Use Cases

The following high level use-cases will be realized by the implementation of this Spec.

DataCenter access from a WAN-client via DC-Gateway (Single Homing)

This use case involves communication within the DataCenter by tenant VMs and also communication between the tenant VMs to a remote WAN-based client via DC-Gateway. The dataplane between the tenant VMs themselves and between the tenant VMs towards the DC-Gateway will be over VXLAN Tunnels.

The dataplane between the DC-Gateway to its other WAN-based BGP Peers is transparent to this spec. It is usually MPLS-based IPVPN.

The BGP Control plane between the ODL Controller and the DC-Gateway will be via EVPN RouteType 5 as defined in EVPN_RT5.

The control plane between the DC-Gateway and it other BGP Peers in the WAN is transparent to this spec, but can be IP-MPLS.

In this use-case:

1. We will have only a single DCGW for WAN connectivity
2. IP prefix exchange between ODL controller and DC-GW (iBGP) using EVPN RT5
3. WAN control plane will use L3VPN IP-MPLS route exchange.
4. On the DC-Gateway, the VRF instance will be configured with two sets of import/export targets. One set of import/export route targets belong to L3VPN inside DataCenter (realized using EVPN RT5) and the second set of import/export route target belongs to WAN control plane.
5. EVPN single homing to be used in all RT5 exchanges inside the DataCenter i.e., ESI=0 for all prefixes sent from DataCenter to the DC-Gateway.
6. Inter AS option B is used at DCGW, route regeneration at DCGW

Datacenter access from another Datacenter over WAN via respective DC-Gateways (L3 DCI)

This use-case involves providing inter-subnet connectivity between two DataCenters. Tenant VMs in one datacenter will be able to communicate with tenant VMs on the other datacenter provided they are part of the same L3VPN and they are on different subnets.

Both the Datacenters can be managed by different ODL Controllers, but the L3VPN configured on both ODL Controllers will use identical RDs and RTs.

Proposed change

The following components of an Openstack-ODL-based solution need to be enhanced to provide intra-subnet and inter-subnet connectivity across DCs using EVPN IP Prefix Advertisement (Route Type 5) mechanism (refer EVPN_RT5):

- Openstack Neutron BGPVPN Driver
- OpenDaylight Controller (NetVirt)
- BGP Quagga Stack to support EVPN with RouteType 5 NLRI
- DC-Gateway BGP Neighbour that supports EVPN with RouteType 5 NLRI

The changes required in Openstack Neutron BGPVPN Driver and BGP Quagga Stack are captured in the Solution considerations section down below.

Pipeline changes

For both the use-cases above, we have put together the required pipeline changes here. For ease of understanding, we have made subsections that talk about Intra-DC traffic and Inter-DC traffic.

INTRA DC

Intra Subnet, Local DPN: VMs on the same subnet, same VPN, same DPN

There are no explicit pipeline changes for this use-case. However the tables that a packet will traverse through is shown below for understanding purposes.

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `tablemiss: goto_table=17` =>
Lport Dispatcher Table (17) `elan service: set elan-id=elan-tag` =>
ELAN Source MAC Table (50) `match: elan-id=elan-tag, src-mac=source-vm-mac` =>
ELAN Destination MAC Table (51) `match: elan-id=elan-tag, dst-mac=dst-vm-mac set output to port-of-dst-vm`

Intra Subnet, Remote DPN: VMs on two different DPNs, both VMs on the same subnet and same VPN

There are no explicit pipeline changes for this use-case. However the tables that a packet will traverse through is shown below for understanding purposes.

VM sourcing the traffic (Ingress DPN)

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `l3vpn service: tablemiss: goto_table=17` =>
Lport Dispatcher Table (17) `elan service: set elan-id=elan-tag` =>
ELAN Source MAC Table (50) `match: elan-id=elan-tag, src-mac=source-vm-mac` =>
ELAN Destination MAC Table (51) `match: elan-id=elan-tag, dst-mac=dst-vm-mac set tun-id=dst-vm-lport-tag, output to vxlan-tun-port`

VM receiving the traffic (Egress DPN)

Classifier Table (0) =>
Internal Tunnel Table (36) `match: tun-id=lport-tag set reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

Inter Subnet, Local DPN: VMs on different subnet, same VPN, same DPN

There are no explicit pipeline changes for this use-case. However the tables that a packet will traverse through is shown below for understanding purposes.

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set output to nexthopgroup-dst-vm` =>
NextHopGroup-dst-vm: `set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

```
cookie=0x8000000, table=0, priority=4,in_port=1 actions=write_metadata:0x10000000000/
↪0xffffff0000000001,goto_table:17
cookie=0x8000001, table=17, priority=5,metadata=0x5000010000000000/0xffffff0000000000␣
↪actions=write_metadata:0x60000100000222e0/0xfffffffffffffffe,goto_table:19
cookie=0x8000009, table=19, priority=20,metadata=0x222e0/0xfffffffe,dl_
↪dst=de:ad:be:ef:00:01 actions=goto_table:21
cookie=0x8000003, table=21, priority=42,ip,metadata=0x222e0/0xfffffffe,nw_dst=10.0.0.
↪2 actions=apply_actions(group:150001)
```

Inter Subnet, Remote DPN: VMs on two different DPNs, both VMs on different subnet, but same VPN

For this use-case there is a change in the remote flow rule to L3 Forward the traffic to the remote VM. The flow-rule will use the LPortTag as the vxlan-tunnel-id, in addition to setting the destination mac address of the remote destination vm.

VM sourcing the traffic (Ingress DPN)

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set eth-dst-mac=dst-vm-mac, tun-id=dst-vm-lport-tag, output to vxlan-tun-port`

```
cookie=0x8000000, table=0, priority=4,in_port=1 actions=write_metadata:0x10000000000/
↪0xffffff0000000001,goto_table:17
cookie=0x8000001, table=17, priority=5,metadata=0x5000010000000000/0xffffff0000000000␣
↪actions=write_metadata:0x60000100000222e0/0xfffffffffffffffe,goto_table:19
cookie=0x8000009, table=19, priority=20,metadata=0x222e0/0xfffffffe,dl_
↪dst=de:ad:be:ef:00:01 actions=goto_table:21
cookie=0x8000003, table=21, priority=42,ip,metadata=0x222e0/0xfffffffe,nw_dst=10.0.0.
↪2 actions=apply_actions(group:150001)
cookie=0x8000003, table=21, priority=42,ip,metadata=0x222e0/0xfffffffe,nw_dst=10.0.0.
↪3 actions=apply_actions(set_field:fa:16:3e:f8:59:af->eth_dst,set_field:0x2->tun_id,
↪output:2)
```

As you can notice 0x2 set in the above flow-rule as tunnel-id is the LPortTag assigned to VM holding IP Address 10.0.0.3.

VM receiving the traffic (Egress DPN)

Classifier Table (0) =>
Internal Tunnel Table (36) `match: tun-id=lport-tag set reg6=lport-tag-dst-vm` =>
Lport Egress Table (220) `Output to dst vm port`

```
cookie=0x8000001, table=0, priority=5,in_port=2 actions=write_metadata:0x40000000001/
↪0xffffff0000000001,goto_table:36
cookie=0x9000001, table=36, priority=5,tun_id=0x2 actions=load:0x400->NXM_NX_REG6[],
↪resubmit(,220)
```

As you notice, 0x2 tunnel-id match in the above flow-rule in INTERNAL_TUNNEL_TABLE (Table 36), is the LPort-Tag assigned to VM holding IP Address 10.0.0.3.

INTER DC

Intra Subnet

Not supported in this Phase

Inter Subnet

For this use-case we are doing a couple of pipeline changes:

a. Introducing a new Table aka L3VNI_EXTERNAL_TUNNEL_DEMUX_TABLE (Table 23). **L3VNI_EXTERNAL_TUNNEL_DEMUX_TABLE (Table 23)** - This table is a new table in the L3VPN pipeline and will be responsible only to process VXLAN packets coming from External VXLAN tunnels.

The packets coming from External VXLAN Tunnels (note: not Internal VXLAN Tunnels), would be directly punted to this new table from the CLASSIFIER TABLE (Table 0) itself. Today when multiple services bind to a tunnel port on GENIUS, the service with highest priority binds directly to Table 0 entry for the tunnel port. So such a service should make sure to provide a fallback to Dispatcher Table so that subsequent service interested in that tunnel traffic would be given the chance.

The new table L3VNI_EXTERNAL_TUNNEL_DEMUX_TABLE will have flows to match on VXLAN VNIs that are L3VNIs. On a match, their action is to fill the metadata with the VPNID, so that further tables in the L3VPN pipeline would be able to continue and operate with the VPNID metadata seamlessly. After filling the metadata, the packets are resubmitted from this new table to the L3_GW_MAC_TABLE (Table 19). The TableMiss in L3VNI_EXTERNAL_TUNNEL_DEMUX_TABLE will resubmit the packet to LPORT_DISPATCHER_TABLE to enable next service if any to process the packet ingressing from the external VXLAN tunnel.

b. For all packets going from VMs within the DC, towards the external gateway device via the External VXLAN Tunnel, we are setting the VXLAN Tunnel ID to the L3VNI value of VPNInstance to which the VM belongs to.

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

Classifier Table (0) =>
L3VNI External Tunnel Demux Table (23) `match: tun-id=l3vni set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set output to nexthopgroup-dst-vm` =>
NextHopGroup-dst-vm: `set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

```
cookie=0x8000001, table=0, priority=5,in_port=9 actions=write_metadata:0x70000000001/
↪0x1fffff0000000001,goto_table:23
cookie=0x8000001, table=19, priority=20,metadata=0x222e0/0xffffffff,dl_
↪dst=de:ad:be:ef:00:06 actions=goto_table:21
cookie=0x8000001, table=23, priority=5,tun_id=0x16 actions= write_metadata:0x222e0/
↪0xfffffffe,resubmit(19)
cookie=0x8000001, table=23, priority=0,resubmit(17)
cookie=0x8000003, table=21, priority=42,ip,metadata=0x222e0/0xfffffffe,nw_dst=10.0.0.
↪2 actions=apply_actions(group:150001)
cookie=0x8000003, table=21, priority=42,ip,metadata=0x222e0/0xfffffffe,nw_dst=10.0.0.
↪3 actions=apply_actions(set_field:fa:16:3e:f8:59:af->eth_dst,set_field:0x2->tun_id,
↪output:2)
```

(continues on next page)

(continued from previous page)

In the above flow rules, Table 23 is the new L3VNI_EXTERNAL_TUNNEL_DEMUX_TABLE. The in_port=9 reprsents an external VXLAN Tunnel port.

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

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=ext-ip-address set eth-dst-mac=dst-mac-address, tun-id=l3vni, output to ext-vxlan-tun-port`

```
cookie=0x7000001, table=0, priority=5,in_port=8, actions=write_metadata:0x60000000001/
↪0x1fffff0000000001,goto_table:17
cookie=0x7000001, table=17, priority=5,metadata=0x60000000001/0x1fffff0000000001␣
↪actions=goto_table:19
cookie=0x7000001, table=19, priority=20,metadata=0x222e0/0xffffffff,dl_
↪dst=de:ad:be:ef:00:06 actions=goto_table:21
cookie=0x7000001, table=23, priority=5,tun_id=0x16 actions= write_metadata:0x222e0/
↪0xfffffffe,resubmit(19)
cookie=0x7000001, table=23, priority=0,resubmit(17)
cookie=0x7000003, table=21, priority=42,ip,metadata=0x222e0/0xfffffffe,nw_dst=10.0.0.
↪2 actions=apply_actions(group:150001)
cookie=0x7000003, table=21, priority=42,ip,metadata=0x222e0/0xfffffffe,nw_dst=10.0.0.
↪3 actions=apply_actions(set_field:fa:16:3e:f8:59:af->eth_dst,set_field:0x2->tun_id,
↪output:2)
```

SNAT pipeline (Access to External Network Access over VXLAN)

SNAT Traffic from Local DPN to External IP (assuming this DPN is NAPT Switch)

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id` =>
PSNAT Table (26) `match: vpn-id=l3vpn-id` =>
Outbound NAPT Table (46) `match: nw-src=vm-ip,port=int-port set src-ip=router-gateway-ip,src-mac=external-router-gateway-mac-address, vpn-id=external-vpn-id,port=ext-port`
=>
NAPT PFIB Table (47) `match: vpn-id=external-vpn-id` =>
L3 FIB Table (21) `match: vpn-id=external-vpn-id nw-dst=external-entity-ip set eth-dst=external-entity-mac tun-id=external-l3vni, output to ext-vxlan-tun-port`

SNAT Reverse Traffic from External IP to Local DPN (assuming this DPN is NAPT Switch)

Classifier Table (0) =>
L3VNI External Tunnel Demux Table (23) `match: tun-id=external-l3vni set vpn-id=external-vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=external-vpn-id, dst-mac=external-router-gateway-mac-address` =>
Inbound NAPT Table (44) `match: vpn-id=external-vpn-id nw-dst=router-gateway-ip port=ext-port set vpn-id=l3vpn-id, dst-ip=vm-ip`
NAPT PFIB Table (47) `match: vpn-id=l3vpn-id` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip-address set output to nexthopgroup-dst-vm` =>
NextHopGroup-dst-vm: `set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

DNAT pipeline (Access from External Network over VXLAN)

DNAT Traffic from External IP to Local DPN

Classifier Table (0) =>
L3VNI External Tunnel Demux Table (23) `match: tun-id=external-l3vni set vpn-id=external-vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=external-vpn-id, eth-dst=floating-ip-dst-vm-mac-address` =>
PDNAT Table (25) `match: nw-dst=floating-ip,eth-dst=floating-ip-dst-vm-mac-address set ip-dst=dst-vm-ip, vpn-id=l3vpn-id` =>
DNAT Table (27) `match: vpn-id=l3vpn-id,nw-dst=dst-vm-ip` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id, nw-dst=dst-vm-ip set output to nexthopgroup-dst-vm` =>
NextHopGroup-dst-vm: `set-eth-dst dst-mac-vm, reg6=dst-vm-lport-tag` =>
Lport Egress Table (220) `Output to dst vm port`

DNAT Reverse Traffic from Local DPN to External IP

Classifier Table (0) =>
Lport Dispatcher Table (17) `l3vpn service: set vpn-id=l3vpn-id` =>
L3 Gateway MAC Table (19) `match: vpn-id=l3vpn-id, dst-mac=vpn-subnet-gateway-mac-address` =>
L3 FIB Table (21) `match: vpn-id=l3vpn-id` =>
PSNAT Table (26) `match: vpn-id=l3vpn-id nw-src=src-vm-ip set ip-src=floating-ip-src-vm, vpn-id=external-vpn-id` =>
SNAT Table (28) `match: vpn-id=external-vpn-id nw-src=floating-ip-src-vm set eth-src=floating-ip-src-vm-mac-address` =>
L3 FIB Table (21) `match: vpn-id=external-vpn-id nw-dst=external-floating-ip set eth-dst=external-mac-address tun-id=external-l3vni, output to ext-vxlan-tun-port`

DNAT to DNAT Traffic (Intra DC)

a) 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`.

b) 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 62: 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
```

(continues on next page)

(continued from previous page)

```
            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 63: 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;
}
```

(continues on next page)

(continued from previous page)

```
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 64: 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;
```

(continues on next page)

(continued from previous page)

```
}
Augment:parent_rd {
    type string;
    mandatory false;
}
```

The encaptype indicates whether an MPLSOverGre or VXLAN encapsulation should be used for this route. If the encapType is MPLSOverGre then the usual label field will carry the MPLS Label to be used in datapath for traffic to/from this VRFEntry IP prefix.

If the encaptype is VXLAN, the VRFEntry implicitly refers that this route is reachable via a VXLAN tunnel. The L3VNI will carry the VRF VNI and there will also be an L2VNI which represents the VNI of the network to which the VRFEntry belongs to.

Based on whether Symmetric IRB (or) Asymmetric IRB is configured to be used by the CSC (see section on Configuration Impact below). If Symmetric IRB is configured, then the L3VNI should be used to program the flows rules. If Asymmetric IRB is configured, then L2VNI should be used in the flow rules.

The mac_address field must be filled for every route in an EVPN. This mac_address field will be used for support intra-DC communication for both inter-subnet and intra-subnet routing.

The gateway_mac_address must always be filled for every route in an EVPN.[AKMA7] [NV8] This gateway_mac_address will be used for all packet exchanges between DC-GW and the DPN in the DC to support L3 based forwarding with Symmetric IRB.

NEUTRONVPN YANG changes

One new leaf l3vni will be added to container grouping `vpn-instance`

Listing 65: 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 66: 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;
    }
```

(continues on next page)

(continued from previous page)

```
    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'],
```

(continues on next page)

(continued from previous page)

```
        'import-RT': ['50:2'],
        'id': '4ae8cd92-48ca-49b5-94e1-b2921a260007',
        'l3vni': '200',
        'tenant-id': 'a565b3ed854247f795c0840b0481c699'
}]}}
```

There is no change in the REST API for associating networks, associating routers (or) deleting the L3VPN.

On the Openstack-side configuration, the vni_ranges configured in Openstack Neutron ml2_conf.ini should not overlap with the L3VNI provided in the ODL RESTful API. In an inter-DC case, where both the DCs are managed by two different Openstack Controller Instances, the workflow will be to do the following:

1. Configure the DC-GW2 facing OSC2 and DC-GW1 facing OSC1 with the same BGP configuration parameters.
2. On first Openstack Controller (OSC1) create an L3VPN1 with RD1 and L3VNI1
3. Create a network Net1 and Associate that Network Net1 to L3VPN1
4. On second Openstack Controller (OSC2) create an L3VPN2 with RD1 with L3VNI2
5. Create a network Net2 on OSC2 and associate that Network Net2 to L3VPN2.
6. Spin-off VM1 on Net1 in OSC1.
7. Spin-off VM2 on Net2 in OSC2.
8. Now VM1 and VM2 should be able to communicate.

Implementation

Assignee(s)

Primary assignee: Kiran N Upadhyaya (kiran.n.upadhyaya@ericsson.com)

Sumanth MS (sumanth.ms@ericsson.com)

Basavaraju Chickmath (basavaraju.chickmath@ericsson.com)

Other contributors: Vivekanandan Narasimhan (n.vivekanandan@ericsson.com)

Work Items

The Trello cards have already been raised for this feature under the EVPN_RT5.

Here is the link for the Trello Card: https://trello.com/c/Tfpr3ezF/33-evpn-evpn-rt5

New tasks into this will be added to cover Java UT and CSIT.

Dependencies

Requires a DC-GW that is supporting EVPN RT5 on BGP Control plane.

Testing

Capture details of testing that will need to be added.

Unit Tests

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

Integration Tests

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

CSIT

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

Documentation Impact

This will require changes to User Guide and Developer Guide.

User Guide will need to add information on how OpenDaylight can be used to deploy L3 BGPVPNs and enable communication across datacenters between virtual endpoints in such L3 BGPVPN.

Developer Guide will capture the ODL L3VPN API changes to enable management of an L3VPN that can use VXLAN overlay to enable communication across datacenters.

References

[1] EVPN_RT5

[2] Network Virtualization using EVPN

[3] Integrated Routing and Bridging in EVPN

[4] VXLAN DCI using EVPN

[5] BGP MPLS-Based Ethernet VPN

- http://docs.opendaylight.org/en/latest/documentation.html
- https://wiki.opendaylight.org/view/Genius:Carbon_Release_Plan

Temporary Source MAC Learning

https://git.opendaylight.org/gerrit/#/q/topic:temp-smac-learning

Temporary source MAC learning introduces two new tables to the ELAN service, for OVS-based source MAC learning using a learn action, to reduce a large scale of packets punted to the controller for an unlearned source MAC.

Problem description

Currently any packet originating from an unknown source MAC address is punted to the controller from the ELAN service (L2 SMAC table 50).

This behavior continues for each packet from this source MAC until ODL properly processes this packet and adds an explicit source MAC rule to this table.

During the time that is required to punt a packet, process it by the ODL and create an appropriate flow, it is not necessary to punt any other packet from this source MAC, as it causes an unnecessary load.

Use Cases

Any L2 traffic from unknown source MACs passing through the ELAN service.

Proposed change

A preliminary logic will be added prior to the SMAC learning table, that will use OpenFlow learn action to add a temporary rule for each source MAC after the first packet is punted.

Pipeline changes

Two new tables will be introduced to the ELAN service:

Table 48 for resubmitting to tables 49 and 50 (trick required to use the learned flows, similar to the ACL implementation).
Table 49 for setting a register value to mark that this SMAC was already punted to the ODL for learning. The flows in this table will be generated automatically by OVS.
Table 50 will be modified, with a new flow, which has a lower priority than the existing known SMAC flows but a higher priority than the default flow. This flow passes packets marked with the register directly to the DMAC table 51 without punting to the controller, as it is already being processed. In addition, the default flow that punts packets to the controller, will also have a new learn action, temporarily adding a flow matching this source MAC to table 49.

Example of flows after change:

```
cookie=0x8040000, duration=1575.755s, table=17, n_packets=7865, n_
 ↪bytes=1451576, priority=6,metadata=0x6000020000000000/0xffffff0000000000␣
 ↪actions=write_metadata:0x7000021389000000/0xfffffffffffffffe,goto_table:48
cookie=0x8500000, duration=1129.530s, table=48, n_packets=4149, n_
 ↪bytes=729778, priority=0 actions=resubmit(,49),resubmit(,50)
cookie=0x8600000, duration=6.875s, table=49, n_packets=0, n_bytes=0, hard_
 ↪timeout=60, priority=0,dl_src=fa:16:3e:2f:73:61 actions=load:0x1->NXM_NX_
 ↪REG4[0..7]
```

(continues on next page)

(continued from previous page)

```
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 assigne and other contributors.

Primary assignee: Olga Schukin (olga.schukin@hpe.com)

Other contributors: Alon Kochba (alonko@hpe.com)

Work Items

N/A.

Dependencies

No new dependencies. Learn action is already in use in netvirt pipeline and has been available in OVS since early versions. However this is a non-standard OpenFlow feature.

Testing

Existing source MAC learning functionality should be verified.

Unit Tests

N/A.

Integration Tests

N/A.

CSIT

N/A.

Documentation Impact

Pipeline documentation should be updated accordingly to reflect the changes to the ELAN service.

Table of Contents

Enhancement to VLAN Provider Network Support

https://git.opendaylight.org/gerrit/#/q/topic:vlan-provider-network

This feature aims to enhance the support for VLAN provider networks that are not of type external.As part of this enhancement, ELAN pipeline processing for the network will be done on the switch only if there is at least one VM port in the network on the switch. The behavior of VLAN provider networks of type external and flat networks will remain unchanged as of now. The optimization for external network is out of scope of this spec and will be handled as part of future releases.

Problem description

Current ODL implementation supports all configured VLAN segments corresponding to VLAN provider networks on a particular patch port on all Open vSwitch which are part of the network. This could have adverse performance impacts because every provider patch port will receive and processes broadcast traffic for all configured VLAN segments even in cases when the switch doesn't have a VM port in the network. Furthermore, for unknown SMACs it leads to unnecessary punts from ELAN pipeline to controller for source MAC learning from all the switches.

Use Cases

L2 forwarding between OVS switches using provider type VLAN over L2 segment of the underlay fabric

Proposed change

Instead of creating the VLAN member interface on the patch port at the time of network creation, VLAN member interface creation will be deferred until a VM port comes up in the switch in the VLAN provider network. Switch pipeline will not process broadcast traffic on this switch in a VLAN provider network until VM port is added to the network. This will be applicable to VLAN provider network without external router attribute set.

Elan service binding will also be done at the time of VLAN member interface creation. Since many neutron ports on same switch can belong to a single VLAN provider network, the flow rule should be created only once when first VM comes up and should be deleted when there are no more neutron ports in the switch for the VLAN provider network.

Pipeline changes

None.

Yang changes

`elan:elan-instances` container will be enhanced with information whether an external router is attached to VLAN provider network.

Listing 67: elan.yang

```
container elan-instances {
     description
        "elan instances configuration parameters. Elan instances support both the␣
↪VLAN and VNI based elans.";

     list elan-instance {
         max-elements "unbounded";
         min-elements "0";
         key "elan-instance-name";
         description
             "Specifies the name of the elan instance. It is a string of 1 to 31
              case-sensitive characters.";
         leaf elan-instance-name {
             type string;
             description "The name of the elan-instance.";
         }
         ...

         leaf external {
             description "indicates whether the network has external router attached␣
↪to it";
             type boolean;
             default "false";
         }
     }
}
```

Configuration impact

None

Clustering considerations

None

Other Infra considerations

N.A.

Security considerations

None.

Scale and Performance Impact

Performance will improve because of the following:

1. Switch will drop packets if it doesn't have a VM port in the VLAN on which packet is received.
2. Unnecessary punts to the controller from ELAN pipeline for source mac learning will be prevented.

Targeted Release

Carbon.

Alternatives

N.A.

Usage

Features to Install

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

REST API

CLI

Implementation

Assignee(s)

Primary assignee:

- Ravindra Nath Thakur (ravindra.nath.thakur@ericsson.com)
- Naveen Kumar Verma (naveen.kumar.verma@ericsson.com)

Other contributors:

- Ravi Sundareswaran (ravi.sundareswaran@ericsson.com)

Work Items

N.A.

Dependencies

This doesn't add any new dependencies.

Testing

Capture details of testing that will need to be added.

Unit Tests

Integration Tests

CSIT

Documentation Impact

This feature will not require any change in User Guide.

References

[1] https://trello.com/c/A6Km6J3D/110-flat-and-vlan-network-type

Table of Contents

VNI based L2 switching, L3 forwarding and NATing

https://git.opendaylight.org/gerrit/#/q/topic:vni-based-l2-l3-nat

Important: All gerrit links raised for this feature will have topic name as **vni-based-l2-l3-nat**

This feature attempts to realize the use of VxLAN VNI (Virtual Network Identifier) for VxLAN tenant traffic flowing on the cloud data-network. This is applicable to L2 switching, L3 forwarding and NATing for all VxLAN based provider networks. In doing so, it eliminates the presence of `LPort tags`, `ELAN tags` and `MPLS labels` on the wire and instead, replaces them with VNIs supplied by the tenant's OpenStack.

This will be selectively done for the use-cases covered by this spec and hence, its implementation won't completely remove the usage of the above entities. The usage of `LPort tags` and `ELAN tags` within an OVS datapath (not on the wire) of the hypervisor will be retained, as eliminating it completely is a large redesign and can be pursued incrementally later.

This spec is the first step in the direction of enforcing datapath semantics that uses tenant supplied VNI values on VxLAN Type networks created by tenants in OpenStack Neutron.

Note: The existing L3 BGPVPN control-path and data-path semantics will continue to use L3 labels on the wire as well as inside the OVS datapaths of the hypervisor to realize both intra-dc and inter-dc connectivity.

Problem description

OpenDaylight NetVirt service today supports the following types of networks:

- Flat
- VLAN
- VxLAN
- GRE

Amongst these, VxLAN-based overlay is supported only for traffic within the DataCenter. External network accesses over the DC-Gateway are supported via VLAN or GRE type external networks. For rest of the traffic over the DC-Gateway, the only supported overlay is GRE.

Today, for VxLAN enabled networks by the tenant, the labels are generated by L3 forwarding service and used. Such labels are re-used for inter-DC use-cases with BGPVPN as well. This does not honor and is not in accordance with the datapath semantics from an orchestration point of view.

This spec attempts to change the datapath semantics by enforcing the VNIs (unique for every VxLAN enabled network in the cloud) **as dictated by the tenant's OpenStack configuration for L2 switching, L3 forwarding and NATing**.

This implementation will remove the reliance on using the following (on the wire) within the DataCenter:

- Labels for L3 forwarding
- LPort tags for L2 switching

More specifically, the traffic from source VM will be routed in source OVS by the L3VPN / ELAN pipeline. After that, the packet will travel as a switched packet in the VxLAN underlay within the DC, containing the VNI in the VxLAN header instead of MPLS label / LPort tag. In the destination OVS, the packet will be collected and sent to the destination VM through the existing ELAN pipeline.

In the nodes themselves, the LPort tag will continue to be used when pushing the packet from ELAN / L3VPN pipeline towards the VM as ACLService continues to use `LPort tags`.

Simiarly `ELAN tags` will continue to be used for handling L2 broadcast packets:

- locally generated in the OVS datapath
- remotely received from another OVS datapath via internal VxLAN tunnels

LPort tag uses 8 bits and ELAN tag uses 21 bits in the metadata. The existing use of both in the metadata will remain unaffected.

In Scope

Since VNIs are provisioned only for VxLAN based underlays, this feature has in its scope the use-cases pertaining to **intra-DC connectivity over internal VxLAN tunnels only**.

On the cloud data network wire, all the VxLAN traffic for basic L2 switching within a VxLAN network and L3 forwarding across VxLAN-type networks using routers will use tenant supplied VNI values for such VXLAN networks.

Inter-DC connectivity over external VxLAN tunnels is covered by the EVPN_RT5 spec.

Out of Scope

- Complete removal of use of `LPort tags` everywhere in ODL: Use of `LPort tags` within the OVS Datapath of a hypervisor, for streaming traffic to the right virtual endpoint on that hypervisor (note: not on the wire) will be retained
- Complete removal of use of `ELAN tags` everywhere in ODL: Use of `ELAN tags` within the OVS Datapath to handle local/remote L2 broadcasts (note: not on the wire) will be retained
- Complete removal of use of `MPLS labels` everywhere in ODL: Use of `MPLS labels` for realizing an L3 BGPVPN (regardless of type of networks put into such BGPVPN that may include networks of type VxLAN) both on the wire and within the OVS Datapaths will be retained.
- Addressing or testing IPv6 use-cases
- Intra DC NAT usecase where no explicit Internet VPN is created for VxLAN based external provider networks: Detailed further in Intra DC subsection in NAT section below.

Complete removal of use of `LPort tags`, `ELAN tags` and `MPLS labels` for VxLAN-type networks has large scale design/pipeline implications and thus need to be attempted as future initiatives via respective specs.

Use Cases

This feature involves amendments/testing pertaining to the following:

L2 switching use cases

1. L2 Unicast frames exchanged within an OVS datapath
2. L2 Unicast frames exchanged over OVS datapaths that are on different hypervisors
3. L2 Broadcast frames transmitted within an OVS datapath
4. L2 Broadcast frames received from remote OVS datapaths

L3 forwarding use cases

1. Router realized using VNIs for networks attached to a new router (with network having pre-created VMs)
2. Router realized using VNIs for networks attached to a new router (with new VMs booted later on the network)
3. Router updated with one or more extra route(s) to an existing VM.
4. Router updated to remove previously added one/more extra routes.

NAT use cases

The provider network types for external networks supported today are:

- External VLAN Provider Networks (transparent Internet VPN)
- External Flat Networks (transparent Internet VPN)
- Tenant-orchestrated Internet VPN of type GRE (actually MPLSOverGRE)

Following are the SNAT/DNAT use-cases applicable to the network types listed above:

1. SNAT functionality.
2. DNAT functionality.
3. DNAT to DNAT functionality (Intra DC)
    - FIP VM to FIP VM on same hypervisor
    - FIP VM to FIP VM on different hypervisors
4. SNAT to DNAT functionality (Intra DC)
    - Non-FIP VM to FIP VM on the same NAPT hypervisor
    - Non-FIP VM to FIP VM on the same hypervisor, but NAPT on different hypervisor
    - Non-FIP VM to FIP VM on different hypervisors (with NAPT on FIP VM hypervisor)
    - Non-FIP VM to FIP VM on different hypervisors (with NAPT on Non-FIP VM hypervisor)

Proposed change

The following components within OpenDaylight Controller needs to be enhanced:

- NeutronVPN Manager
- ELAN Manager
- VPN Engine (VPN Manager, VPN Interface Manager and VPN Subnet Route Handler)
- FIB Manager
- NAT Service

Pipeline changes

L2 Switching

Unicast

Within hypervisor

There are no explicit pipeline changes for this use-case.

Across hypervisors

- *Ingress OVS*

  Instead of setting the destination LPort tag, destination network VNI will be set in the `tun_id` field in `L2_DMAC_FILTER_TABLE` (table 51) while egressing the packet on the tunnel port.

  The modifications in flows and groups on the ingress OVS are illustrated below:

```
cookie=0x8000000, duration=65.484s, table=0, n_packets=23, n_bytes=2016,␣
↪priority=4,in_port=6actions=write_metadata:0x30000000000/0xffffff0000000001,
↪goto_table:17
cookie=0x6900000, duration=63.106s, table=17, n_packets=23, n_bytes=2016,␣
↪priority=1,metadata=0x30000000000/0xffffff0000000000 actions=write_
↪metadata:0x2000030000000000/0xfffffffffffffffe,goto_table:40
cookie=0x6900000, duration=64.135s, table=40, n_packets=4, n_bytes=392,␣
↪priority=61010,ip,dl_src=fa:16:3e:86:59:fd,nw_src=12.1.0.4 actions=ct(table=41,
↪zone=5002)
cookie=0x6900000, duration=5112.542s, table=41, n_packets=21, n_bytes=2058,␣
↪priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,17)
cookie=0x8040000, duration=62.125s, table=17, n_packets=15, n_bytes=854,␣
↪priority=6,metadata=0x6000030000000000/0xffffff0000000000 actions=write_
↪metadata:0x700003138a000000/0xfffffffffffffffe,goto_table:48
cookie=0x8500000, duration=5113.124s, table=48, n_packets=24, n_bytes=3044,␣
↪priority=0 actions=resubmit(,49),resubmit(,50)
cookie=0x805138a, duration=62.163s, table=50, n_packets=15, n_bytes=854,␣
↪priority=20,metadata=0x3138a000000/0xffffffffff000000,dl_src=fa:16:3e:86:59:fd␣
↪actions=goto_table:51
cookie=0x803138a, duration=62.163s, table=51, n_packets=6, n_bytes=476,␣
↪priority=20,metadata=0x138a000000/0xffff000000,dl_dst=fa:16:3e:31:fb:91␣
↪actions=set_field:**0x710**->tun_id,output:1
```

- *Egress OVS*

  On the egress OVS, for the packets coming in via the internal VxLAN tunnel (OVS - OVS), `INTERNAL_TUNNEL_TABLE` currently matches on destination LPort tag for unicast packets. Since the incoming packets will now contain the network VNI in the VxLAN header, the `INTERNAL_TUNNEL_TABLE` will match on this VNI, set the ELAN tag in the metadata and forward the packet to `L2_DMAC_FILTER_TABLE` so as to reach the destination VM via the ELAN pipeline.

  The modifications in flows and groups on the egress OVS are illustrated below:

```
cookie=0x8000001, duration=5136.996s, table=0, n_packets=12601, n_bytes=899766,␣
↪priority=5,in_port=1,actions=write_metadata:0x10000000001/0xfffff0000000001,
↪goto_table:36
cookie=0x9000004, duration=1145.594s, table=36, n_packets=15, n_bytes=476,␣
↪priority=5,**tun_id=0x710,actions=write_metadata:0x138a000001/0xffffffff000000,
↪goto_table:51**
cookie=0x803138a, duration=62.163s, table=51, n_packets=9, n_bytes=576,␣
↪priority=20,metadata=0x138a000001/0xffff000000,dl_dst=fa:16:3e:86:59:fd␣
↪actions=load:0x300->NXM_NX_REG6[],resubmit(,220)
cookie=0x6900000, duration=63.122s, table=220, n_packets=9, n_bytes=1160,␣
↪priority=6,reg6=0x300actions=load:0x70000300->NXM_NX_REG6[],write_
↪metadata:0x7000030000000000/0xfffffffffffffffe,goto_table:251
cookie=0x6900000, duration=65.479s, table=251, n_packets=8, n_bytes=392,␣
↪priority=61010,ip,dl_dst=fa:16:3e:86:59:fd,nw_dst=12.1.0.4 actions=ct(table=252,
↪zone=5002)
cookie=0x6900000, duration=5112.299s, table=252, n_packets=19, n_bytes=1862,␣
↪priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,220)
cookie=0x8000007, duration=63.123s, table=220, n_packets=8, n_bytes=1160,␣
↪priority=7,reg6=0x70000300actions=output:6
```

Broadcast

Across hypervisors

The ARP broadcast by the VM will be a (local + remote) broadcast.

For the local broadcast on the VM's OVS itself, the packet will continue to get flooded to all the VM ports by setting the destination LPort tag in the local broadcast group. Hence, there are no explicit pipeline changes for when a packet is transmitted within the source OVS via a local broadcast.

The changes in pipeline for the remote broadcast are illustrated below:

- *Ingress OVS*

  Instead of setting the ELAN tag, network VNI will be set in the `tun_id` field as part of bucket actions in remote broadcast group while egressing the packet on the tunnel port.

  The modifications in flows and groups on the ingress OVS are illustrated below:

```
cookie=0x8000000, duration=65.484s, table=0, n_packets=23, n_bytes=2016,␣
↪priority=4,in_port=6actions=write_metadata:0x30000000000/0xfffff0000000001,
↪goto_table:17
cookie=0x6900000, duration=63.106s, table=17, n_packets=23, n_bytes=2016,␣
↪priority=1,metadata=0x30000000000/0xfffff0000000000 actions=write_
↪metadata:0x2000030000000000/0xfffffffffffffffe,goto_table:40
cookie=0x6900000, duration=64.135s, table=40, n_packets=4, n_bytes=392,␣
↪priority=61010,ip,dl_src=fa:16:3e:86:59:fd,nw_src=12.1.0.4 actions=ct(table=41,
↪zone=5002)
```

(continues on next page)

(continued from previous page)

```
cookie=0x6900000, duration=5112.542s, table=41, n_packets=21, n_bytes=2058,␣
↪priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,17)
cookie=0x8040000, duration=62.125s, table=17, n_packets=15, n_bytes=854,␣
↪priority=6,metadata=0x6000030000000000/0xffffff0000000000 actions=write_
↪metadata:0x700003138a000000/0xfffffffffffffffe,goto_table:48
cookie=0x8500000, duration=5113.124s, table=48, n_packets=24, n_bytes=3044,␣
↪priority=0 actions=resubmit(,49),resubmit(,50)
cookie=0x805138a, duration=62.163s, table=50, n_packets=15, n_bytes=854,␣
↪priority=20,metadata=0x3138a000000/0xffffffffff000000,dl_src=fa:16:3e:86:59:fd␣
↪actions=goto_table:51
cookie=0x8030000, duration=5112.911s, table=51, n_packets=18, n_bytes=2568,␣
↪priority=0 actions=goto_table:52
cookie=0x870138a, duration=62.163s, table=52, n_packets=9, n_bytes=378,␣
↪priority=5,metadata=0x138a000000/0xffff000001 actions=write_
↪actions(group:210004)

group_id=210004,type=all,bucket=actions=group:210003,bucket=actions=set_
↪field:**0x710**->tun_id,output:1
```

- *Egress OVS*

  On the egress OVS, for the packets coming in via the internal VxLAN tunnel (OVS - OVS), `INTERNAL_TUNNEL_TABLE` currently matches on ELAN tag for broadcast packets. Since the incoming packets will now contain the network VNI in the VxLAN header, the `INTERNAL_TUNNEL_TABLE` will match on this VNI, set the ELAN tag in the metadata and forward the packet to `L2_DMAC_FILTER_TABLE` to be broadcasted via the local broadcast groups traversing the ELAN pipeline.

  The `TUNNEL_INGRESS_BIT` being set in the `CLASSIFIER_TABLE` (table 0) ensures that the packet is always sent to the local broadcast group only and hence, remains within the OVS. This is necessary to avoid switching loop back to the source OVS.

  The modifications in flows and groups on the egress OVS are illustrated below:

```
cookie=0x8000001, duration=5136.996s, table=0, n_packets=12601, n_bytes=899766,␣
↪priority=5,in_port=1,actions=write_metadata:0x10000000001/0xfffff0000000001,
↪goto_table:36
cookie=0x9000004, duration=1145.594s, table=36, n_packets=15, n_bytes=476,␣
↪priority=5,**tun_id=0x710,actions=write_metadata:0x138a000001/0xffffffffff000000,
↪goto_table:51**
cookie=0x8030000, duration=5137.609s, table=51, n_packets=9, n_bytes=1293,␣
↪priority=0 actions=goto_table:52
cookie=0x870138a, duration=1145.592s, table=52, n_packets=0, n_bytes=0,␣
↪priority=5,metadata=0x138a000001/0xffff000001 actions=apply_
↪actions(group:210003)

group_id=210003,type=all,bucket=actions=set_field:0x4->tun_id,resubmit(,55)

cookie=0x8800004, duration=1145.594s, table=55, n_packets=9, n_bytes=378,␣
↪priority=9,tun_id=0x4,actions=load:0x400->NXM_NX_REG6[],resubmit(,220)
cookie=0x6900000, duration=63.122s, table=220, n_packets=9, n_bytes=1160,␣
↪priority=6,reg6=0x300actions=load:0x70000300->NXM_NX_REG6[],write_
↪metadata:0x7000030000000000/0xfffffffffffffffe,goto_table:251
cookie=0x6900000, duration=65.479s, table=251, n_packets=8, n_bytes=392,␣
↪priority=61010,ip,dl_dst=fa:16:3e:86:59:fd,nw_dst=12.1.0.4 actions=ct(table=252,
↪zone=5002)
cookie=0x6900000, duration=5112.299s, table=252, n_packets=19, n_bytes=1862,␣
↪priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,220)
```

(continues on next page)

(continued from previous page)

```
cookie=0x8000007, duration=63.123s, table=220, n_packets=8, n_bytes=1160,␣
↪priority=7,reg6=0x70000300actions=output:6
```

The ARP response will be a unicast packet, and as indicated above, for unicast packets, there are no explicit pipeline changes.

L3 Forwarding

Between VMs on a single OVS

There are no explicit pipeline changes for this use-case. The destination LPort tag will continue to be set in the nexthop group since when `The EGRESS_DISPATCHER_TABLE` sends the packet to `EGRESS_ACL_TABLE`, it is used by the ACL service.

Between VMs on two different OVS

L3 forwarding between VMs on two different hypervisors is asymmetric forwarding since the traffic is routed in the source OVS datapath while it is switched over the wire and then all the way to the destination VM on the destination OVS datapath.

VM sourcing the traffic (Ingress OVS)

`L3_FIB_TABLE` will set the destination network VNI in the `tun_id` field instead of the MPLS label.

```
CLASSIFIER_TABLE => DISPATCHER_TABLE => INGRESS_ACL_TABLE =>
DISPATCHER_TABLE => L3_GW_MAC_TABLE =>
L3_FIB_TABLE (set destination MAC, **set tunnel-ID as destination network VNI**)
=> Output to tunnel port
```

The modifications in flows and groups on the ingress OVS are illustrated below:

```
cookie=0x8000000, duration=128.140s, table=0, n_packets=25, n_bytes=2716, priority=4,
↪in_port=5 actions=write_metadata:0x50000000000/0xffffff0000000001,goto_table:17
cookie=0x8000000, duration=4876.599s, table=17, n_packets=0, n_bytes=0, priority=0,
↪metadata=0x5000000000000000/0xf000000000000000 actions=write_
↪metadata:0x6000000000000000/0xf000000000000000,goto_table:80
cookie=0x1030000, duration=4876.563s, table=80, n_packets=0, n_bytes=0, priority=0␣
↪actions=resubmit(,17)
cookie=0x6900000, duration=123.870s, table=17, n_packets=25, n_bytes=2716, priority=1,
↪metadata=0x50000000000/0xffffff0000000000 actions=write_metadata:0x2000050000000000/
↪0xfffffffffffffffe,goto_table:40
cookie=0x6900000, duration=126.056s, table=40, n_packets=15, n_bytes=1470,␣
↪priority=61010,ip,dl_src=fa:16:3e:63:ea:0c,nw_src=10.1.0.4 actions=ct(table=41,
↪zone=5001)
cookie=0x6900000, duration=4877.057s, table=41, n_packets=17, n_bytes=1666,␣
↪priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,17)
cookie=0x6800001, duration=123.485s, table=17, n_packets=28, n_bytes=3584, priority=2,
↪metadata=0x2000050000000000/0xffffff0000000000 actions=write_
↪metadata:0x5000050000000000/0xfffffffffffffffe,goto_table:60
cookie=0x6800000, duration=3566.900s, table=60, n_packets=24, n_bytes=2184,␣
↪priority=0 actions=resubmit(,17)
```

(continues on next page)

(continued from previous page)

```
cookie=0x8000001, duration=123.456s, table=17, n_packets=17, n_bytes=1554, priority=5,
↪metadata=0x5000050000000000/0xffffff0000000000 actions=write_
↪metadata:0x60000500000222e0/0xfffffffffffffffe,goto_table:19
cookie=0x8000009, duration=124.815s, table=19, n_packets=15, n_bytes=1470,␣
↪priority=20,metadata=0x222e0/0xffffffff,dl_dst=fa:16:3e:51:da:ee actions=goto_
↪table:21
cookie=0x8000003, duration=125.568s, table=21, n_packets=9, n_bytes=882, priority=42,
↪ip,metadata=0x222e0/0xffffffff,nw_dst=12.1.0.3 actions=**set_field:0x710->tun_id**,
↪set_field:fa:16:3e:31:fb:91->eth_dst,output:1
```

VM receiving the traffic (Egress OVS)

On the egress OVS, for the packets coming in via the VxLAN tunnel, `INTERNAL_TUNNEL_TABLE` currently matches on MPLS label and sends it to the nexthop group to be taken to the destination VM via `EGRESS_ACL_TABLE`. Since the incoming packets will now contain network VNI in the VxLAN header, the `INTERNAL_TUNNEL_TABLE` will match on the VNI, set the ELAN tag in the metadata and forward the packet to `L2_DMAC_FILTER_TABLE`, from where it will be taken to the destination VM via the ELAN pipeline.

```
CLASSIFIER_TABLE => INTERNAL_TUNNEL_TABLE (Match on network VNI, set ELAN tag in the␣
↪metadata)
=> L2_DMAC_FILTER_TABLE (Match on destination MAC) => EGRESS_DISPATCHER_TABLE
=> EGRESS_ACL_TABLE => Output to destination VM port
```

The modifications in flows and groups on the egress OVS are illustrated below:

```
cookie=0x8000001, duration=4918.647s, table=0, n_packets=12292, n_bytes=877616,␣
↪priority=5,in_port=1actions=write_metadata:0x10000000001/0xfffff0000000001,goto_
↪table:36
cookie=0x9000004, duration=927.245s, table=36, n_packets=8234, n_bytes=52679,␣
↪priority=5,**tun_id=0x710,actions=write_metadata:0x138a000001/0xfffffffff000000,
↪goto_table:51**
cookie=0x803138a, duration=62.163s, table=51, n_packets=9, n_bytes=576, priority=20,
↪metadata=0x138a000001/0xffff000000,dl_dst=fa:16:3e:86:59:fd actions=load:0x300->NXM_
↪NX_REG6[],resubmit(,220)
cookie=0x6900000, duration=63.122s, table=220, n_packets=9, n_bytes=1160, priority=6,
↪reg6=0x300actions=load:0x70000300->NXM_NX_REG6[],write_metadata:0x7000030000000000/
↪0xfffffffffffffffe,goto_table:251
cookie=0x6900000, duration=65.479s, table=251, n_packets=8, n_bytes=392,␣
↪priority=61010,ip,dl_dst=fa:16:3e:86:59:fd,nw_dst=12.1.0.4 actions=ct(table=252,
↪zone=5002)
cookie=0x6900000, duration=5112.299s, table=252, n_packets=19, n_bytes=1862,␣
↪priority=62020,ct_state=-new+est-rel-inv+trk actions=resubmit(,220)
cookie=0x8000007, duration=63.123s, table=220, n_packets=8, n_bytes=1160, priority=7,
↪reg6=0x70000300actions=output:6
```

NAT Service

For NAT, we need VNIs to be used in two scenarios:

- When packet is forwarded from non-NAPT to NAPT hypervisor (VNI per router)
- Between hypervisors (intra DC) over Internet VPN (VNI per Internet VPN)

Hence, a pool titled `opendaylight-vni-ranges`, non-overlapping with the OpenStack Neutron vni_ranges configuration, needs to be configured by the OpenDaylight Controller Administrator.

This `opendaylight-vni-ranges` pool will be used to carve out a unique VNI per router to be then used in the datapath for traffic forwarding from non-NAPT to NAPT switch for this router.

Similarly, for MPLSOverGRE based external networks, the `opendaylight-vni-ranges` pool will be used to carve out a unique VNI per Internet VPN (GRE-provider-type) to be then used in the datapath for traffic forwarding for `SNAT-to-DNAT` and `DNAT-to-DNAT` cases within the DataCenter. Only one external network can be associated to Internet VPN today and this spec doesn't attempt to address that limitation.

A NeutronVPN configuration API will be exposed to the administrator to configure the lower and higher limit for this pool. If the administrator doesn't configure this explicitly, then the pool will be created with default values of lower limit set to 70000 and upper limit set to 100000, during the first NAT session configuration.

FIB Manager changes: For external network of type GRE, it is required to use `Internet VPN VNI` for intra-DC communication, but we still require `MPLS labels` to reach SNAT/DNAT VMs from external entities via MPLSOverGRE. Hence, we will make use of the `l3vni` attribute added to fibEntries container as part of EVPN_RT5 spec. NAT will populate both `label` and `l3vni` values for fibEntries created for floating-ips and external-fixed-ips with external network of type GRE. This `l3vni` value will be used while programming remote FIB flow entries (on all the switches which are part of the same VRF). But still, MPLS label will be used to advertise prefixes and in `L3_LFIB_TABLE` taking the packet to `INBOUND_NAPT_TABLE` and `PDNAT_TABLE`.

For SNAT/DNAT use-cases, we have following provider network types for External Networks:

1. VLAN - not VNI based
2. Flat - not VNI based
3. VxLAN - VNI based (covered by the EVPN_RT5 spec)
4. GRE - not VNI based (will continue to use `MPLS labels`)

Inter DC

SNAT

- From a VM on a NAPT switch to reach Internet, and reverse traffic reaching back to the VM

  There are no explicit pipeline changes.

- From a VM on a non-NAPT switch to reach Internet, and reverse traffic reaching back to the VM

  On the non-NAPT switch, `PSNAT_TABLE` (table 26) will be set with `tun_id` field as `Router Based VNI` allocated from the pool and send to group to reach NAPT switch.

  On the NAPT switch, `INTERNAL_TUNNEL_TABLE` (table 36) will match on the `tun_id` field which will be `Router Based VNI` and send the packet to `OUTBOUND_NAPT_TABLE` (table 46) for SNAT Translation and to be taken to Internet.

  – *Non-NAPT switch*

```
cookie=0x8000006, duration=2797.179s, table=26, n_packets=47, n_bytes=3196,␣
↪priority=5,ip,metadata=0x23a50/0xfffffffe actions=**set_field:0x710->tun_
↪id**,group:202501

group_id=202501,type=all,bucket=actions=output:1
```

– *NAPT switch*

```
cookie=0x8000001, duration=4918.647s, table=0, n_packets=12292, n_
↪bytes=877616, priority=5,in_port=1,actions=write_metadata:0x10000000001/
↪0xfffff0000000001,goto_table:36
cookie=0x9000004, duration=927.245s, table=36, n_packets=8234, n_bytes=52679,␣
↪priority=10,ip,**tun_id=0x710**,actions=write_metadata:0x23a50/0xfffffffe,
↪goto_table:46
```

As part of the response from NAPT switch, the packet will be taken to the Non-NAPT switch after SNAT reverse translation using destination VMs Network VNI.

DNAT

There is no NAT specific explicit pipeline change for DNAT traffic to DC-gateway.

Intra DC

- VLAN Provider External Networks: VNI is not applicable on the external VLAN Provider network. However, the Router VNI will be used for datapath traffic from non-NAPT switch to NAPT-switch over the internal VxLAN tunnel.
- VxLAN Provider External Networks:
  - **Explicit creation of Internet VPN**: An L3VNI, mandatorily falling within the `opendaylight-vni-ranges`, will be provided by the Cloud admin (or tenant). This VNI will be used uniformly for all packet transfer over the VxLAN wire for this Internet VPN (uniformly meaning all the traffic on Internal or External VXLAN Tunnel, except the non-NAPT to NAPT communication). This usecase is covered by EVPN_RT5 spec
  - **No explicit creation of Internet VPN**: A transparent Internet VPN having UUID same as that of the corresponding external network UUID is created implicitly and the VNI configured for this external network should be used on the VxLAN wire. This usecase is **out of scope** from the perspective of this spec, and the same is indicated in *Out of Scope* section.
- GRE Provider External Networks: `Internet VPN VNI` will be carved per Internet VPN using `opendaylight-vni-ranges` to be used on the wire.

DNAT to DNAT

- FIP VM to FIP VM on different hypervisors

  After DNAT translation on the first hypervisor `DNAT-OVS-1`, the traffic will be sent to the `L3_FIB_TABLE` (table=21) in order to reach the floating IP VM on the second hypervisor `DNAT-OVS-2`. Here, the `tun_id` action field will be set as the `INTERNET VPN VNI` value.

  – *DNAT-OVS-1*

```
cookie=0x8000003, duration=518.567s, table=21, n_packets=0, n_bytes=0,␣
↪priority=42,ip,metadata=0x222e8/0xfffffffe,nw_dst=172.160.0.200␣
↪actions=**set_field:0x11178->tun_id**,output:9
```

- *DNAT-OVS-2*

```
cookie=0x9011177, duration=411685.075s, table=36, n_packets=2, n_bytes=196,␣
↪priority=**6**,**tun_id=0x11178**actions=resubmit(,25)
cookie=0x9011179, duration=478573.171s, table=36, n_packets=2, n_bytes=140,␣
↪priority=5,**tun_id=0x11178**,actions=goto_table:44

cookie=0x8000004, duration=408145.805s, table=25, n_packets=600, n_
↪bytes=58064, priority=10,ip,nw_dst=172.160.0.100,**eth_
↪dst=fa:16:3e:e6:e3:c6** actions=set_field:10.0.0.5->ip_dst,write_
↪metadata:0x222e0/0xfffffffe,goto_table:27
cookie=0x8000004, duration=408145.805s, table=25, n_packets=600, n_
↪bytes=58064, priority=10,ipactions=goto_table:44
```

First, the `INTERNAL_TUNNEL_TABLE` (table=36) will take the packet to the `PDNAT_TABLE` (table 25) for an exact FIP match in `PDNAT_TABLE`.

- In case of a successful FIP match, `PDNAT_TABLE` will further match on floating IP MAC. This is done as a security prerogative since in DNAT usecases, the packet can land to the hypervisor directly from the external world. Hence, better to have a second match criteria.
- In case of no match, the packet will be redirected to the SNAT pipeline towards the `INBOUND_NAPT_TABLE` (table=44). This is the use-case where `DNAT-OVS-2` also acts as the NAPT switch.

In summary, on an given NAPT switch, if both DNAT and SNAT are configured, the incoming traffic will first be sent to the `PDNAT_TABLE` and if there is no FIP match found, then it will be forwarded to `INBOUND_NAPT_TABLE` for SNAT translation.

As part of the response, the `Internet VPN VNI` will be used as `tun_id` to reach floating IP VM on `DNAT-OVS-1`.

- FIP VM to FIP VM on same hypervisor

  The pipeline changes will be similar as are for different hypervisors, the only difference being that `INTERNAL_TUNNEL_TABLE` will never be hit in this case.

SNAT to DNAT

- Non-FIP VM to FIP VM on different hypervisors (with NAPT elected as the FIP VM hypervisor)

  The packet will be sent to the NAPT hypervisor from non-FIP VM (for SNAT translation) using `Router VNI` (similar to as described in *SNAT* section). As part of the response from the NAPT switch after SNAT reverse translation, the packet is forwarded to non-FIP VM using destination VM's Network VNI.

- Non-FIP VM to FIP VM on the same NAPT hypervisor

  There are no explicit pipeline changes for this use-case.

- Non-FIP VM to FIP VM on the same hypervisor, but a different hypervisor elected as NAPT switch

  - *NAPT hypervisor*

    The packet will be sent to the NAPT hypervisor from non-FIP VM (for SNAT translation) using `Router VNI` (similar to as described in *SNAT* section). On the NAPT switch, the `INTERNAL_TUNNEL_TABLE`

will match on the `Router VNI` in the `tun_id` field and send the packet to `OUTBOUND_NAPT_TABLE` for SNAT translation (similar to as described in *SNAT* section).

```
cookie=0x8000005, duration=5073.829s, table=36, n_packets=61, n_bytes=4610,␣
↪priority=10,ip,**tun_id=0x11170**,actions=write_metadata:0x222e0/0xfffffffe,
↪goto_table:46
```

The packet will later be sent back to the FIP VM hypervisor from L3_FIB_TABLE with `tun_id` field set as the `Internet VPN VNI`.

```
cookie=0x8000003, duration=518.567s, table=21, n_packets=0, n_bytes=0,␣
↪priority=42,ip,metadata=0x222e8/0xfffffffe,nw_dst=172.160.0.200␣
↪actions=**set_field:0x11178->tun_id**,output:9
```

– *FIP VM hypervisor*

On reaching the FIP VM Hypervisor, the packet will be sent for DNAT translation. The `INTERNAL_TUNNEL_TABLE` will match on the `Internet VPN VNI` in the `tun_id` field and send the packet to `PDNAT_TABLE`.

```
cookie=0x9011177, duration=411685.075s, table=36, n_packets=2, n_bytes=196,␣
↪priority=**6**,**tun_id=0x11178**,actions=resubmit(,25)
cookie=0x8000004, duration=408145.805s, table=25, n_packets=600, n_
↪bytes=58064, priority=10,ip,nw_dst=172.160.0.100,**eth_
↪dst=fa:16:3e:e6:e3:c6** actions=set_field:10.0.0.5->ip_dst,write_
↪metadata:0x222e0/0xfffffffe,goto_table:27
```

Upon FIP VM response, DNAT reverse translation happens and traffic is sent back to the NAPT switch for SNAT translation. The `L3_FIB_TABLE` will be set with `Internet VPN VNI` in the `tun_id` field.

```
cookie=0x8000003, duration=95.300s, table=21, n_packets=2, n_bytes=140,␣
↪priority=42,ip,metadata=0x222ea/0xfffffffe,nw_dst=172.160.0.3 actions=**set_
↪field:0x11178->tun_id**,output:5
```

– *NAPT hypervisor*

On NAPT hypervisor, the `INTERNAL_TUNNEL_TABLE` will match on the `Internet VPN VNI` in the `tun_id` field and send the packet to `` INBOUND_NAPT_TABLE`` for SNAT reverse translation (external fixed IP to VM IP). The packet will then be sent back to the non-FIP VM using destination VM's Network VNI.

- Non-FIP VM to FIP VM on different hypervisors (with NAPT elected as the non-FIP VM hypervisor)

  After SNAT Translation, `Internet VPN VNI` will be used to reach FIP VM. On FIP VM hypervisor, the `INTERNAL_TUNNEL_TABLE` will take the packet to the `PDNAT_TABLE` to match on `Internet VPN VNI` in the `tun_id` field for DNAT translation.

  Upon response from FIP, DNAT reverse translation happens and uses `Internet VPN VNI` to reach back to the non-FIP VM.

YANG changes

- `opendaylight-vni-ranges` and `enforce-openstack-semantics` leaf elements will be added to neutronvpn-config container in `neutronvpn-config.yang`:
  - `opendaylight-vni-ranges` will be introduced to accept inputs for the VNI range pool from the configurator via the corresponding exposed REST API. In case this is not defined, the default value defined in `netvirt-neutronvpn-config.xml` will be used to create this pool.
  - `enforce-openstack-semantics` will be introduced to have the flexibility to enable or disable OpenStack semantics in the dataplane for this feature. It will be defaulted to true, meaning these semantics will be enforced by default. In case it is set to false, the dataplane will continue to be programmed with LPort tags / ELAN tags for switching and with labels for routing use-cases. Once this feature gets stabilized and the semantics are in place to use VNIs on the wire for BGPVPN based forwarding too, this config can be permanently removed if deemed fit.

Listing 68: neutronvpn-config.yang

```
container neutronvpn-config {
    config true;
    ...
    ...
    leaf opendaylight-vni-ranges {
        type string;
        default "70000:99999";
    }
    leaf enforce-openstack-semantics {
        type boolean;
        default true;
    }
}
```

- Provider network-type and provider segmentation-ID need to be propagated to FIB Manager to manipulate flows based on the same. Hence:
  - A new grouping `network-attributes` will be introduced in `neutronvpn.yang` to hold network type and segmentation ID. This grouping will replace the leaf-node `network-id` in `subnetmaps` MD-SAL configuration datastore:

Listing 69: 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.
```

(continues on next page)

(continued from previous page)

```
            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 70: neutronvpn.yang

```
notification subnet-added-to-vpn {
    description "new subnet added to vpn";
    ...
    ...
    uses network-attributes;
}
```

– VpnSubnetRouteHandler will act on these notifications and store these attributes in `subnet-op-data` MD-SAL operational datastore as described below. FIB Manager will get to retrieve the `subnetID` from the primary adjacency of the concerned VPN interface. This `subnetID` will be used as the key to retrieve `network-attributes` from `subnet-op-data` datastore.

Listing 71: 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 72: odl-l3vpn.yang

```
container prefix-to-interface {
    config false;
    list vpn-ids {
        key vpn-id;
        leaf vpn-id {type uint32;}
        list prefixes {
            key ip_address;
            ...
            ...
            leaf subnet-id {
                type yang:uuid;
            }
            leaf nat-prefix {
                type boolean;
                default false;
            }
        }
    }
}
```

Configuration impact

- We have to make sure that we do not accept configuration of VxLAN type provider networks without the `segmentation-ID` available in them since we are using it to represent the VNI on the wire and in the flows/groups.

Clustering considerations

No specific additional clustering considerations to be adhered to.

Other Infra considerations

None.

Security considerations

None.

Scale and Performance Impact

None.

Targeted Release(s)

Carbon.

Known Limitations

None.

Alternatives

N.A.

Usage

Features to Install

odl-netvirt-openstack

REST API

No new changes to the existing REST APIs.

CLI

No new CLI is being added.

Implementation

Assignee(s)

Primary assignee: Abhinav Gupta <abhinav.gupta@ericsson.com> Vivekanandan Narasimhan <n.vivekanandan@ericsson.com>

Other contributors: Chetan Arakere Gowdru <chetan.arakere@altencalsoftlabs.com> Karthikeyan Krishnan <karthikeyan.k@altencalsoftlabs.com> Yugandhar Sarraju <yugandhar.s@altencalsoftlabs.com>

Work Items

Trello card: https://trello.com/c/PfARbEmU/84-enforce-vni-on-the-wire-for-l2-switching-l3-forwarding-and-nating-on-vxlan-overlay-

1. Code changes to alter the pipeline and e2e testing of the use-cases mentioned.
2. Add Documentation

Dependencies

This doesn't add any new dependencies.

Testing

Unit Tests

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

Integration Tests

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

CSIT

No new testcases to be added, existing ones should continue to succeed.

Documentation Impact

This will require changes to the Developer Guide.

Developer Guide needs to capture how this feature modifies the existing Netvirt L3 forwarding service implementation.

References

- http://docs.opendaylight.org/en/latest/documentation.html
- https://wiki.opendaylight.org/view/Genius:Carbon_Release_Plan
- EVPN_RT5

Table of Contents

Weighted NAPT Selection

https://git.opendaylight.org/gerrit/#/q/topic:weighted-napt-selection

Brief introduction of the feature.

Problem description

NATService needs to select a Designated NAPT switch amongst different switches hosting VMs in a given VRF. Currently this selection uses a round robin algorithm which treats all switches as equals. A switch that is selected as Designated NAPT gets all external traffic from VMs in that VRF, thus putting extra load on that switch. Since selection of NAPT is runtime decision it is not possible to scale-up such switches.

To solve this problem we need a mechanism to make NAPT selection logic pick particular switches which can handle extra traffic, more over ones that can't. Administrator should be able to mark such switches at deployment.

Use Cases

Main use case is to allow admin to specify configuration for specific computes so they are more likely to be selected as Designated NAPT Switches. Following use cases will be supported as part of initial feature.

- Configuration of switch with weight to pin more VRFs to it
- Add a switch with more weight than existing switches in VRF
- Remove a switch with higher weight and add it back in

Note: No changes will be made to triggers for NAPT Selection for initial release This means following use cases will not be supported yet.

- Addition of switch with higher weight will not result in re-distribution of VRFs.
- Decreasing/Increasing weight of a switch will not trigger r-distribution of VRFs.

Proposed change

We'll introduce a new configuration parameter called `odl_base_weight` which will be configured in `external_ids` parameter of `Open_vSwitch` in specific switches. This will be part of 0-day orchestration. Value for this will be a number. If nothing is configured bas weight will be considered to be `0`.

Higher the `odl_base_weight`, greater the number of VRFs designated on a given switch.

`NAPTSwitchSelector` will be modified to factor in this parameter when selecting a designate NAPT switch. Currently weight of a given switch is only number of VRFs hosted on it with base weight of 0. Weight of switch is incremented by *1* for each VRF hosted on it. Switch with least weight at time of selection ends up being selected as designated Switch. `odl_base_weight` of *X* will translate to weight *-X* in `NAPTSwitchSelector`.

Pipeline changes

None.

Yang changes

None.

Configuration impact

A new configuration parameter called `odl_base_weight` which can be configured in `external_ids` parameter of `Open_vSwitch` for specific switches.

Refer section [Proposed Change] for more details.

Clustering considerations

N.A.

Other Infra considerations

Requires new API in Genius.

Security considerations

N.A.

Scale and Performance Impact

Selecting switches that can handle extra traffic that SNAT brings in will help with dataplane performance

Targeted Release

Oxygen

Alternatives

None.

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

REST API

None.

CLI

None.

Implementation

Assignee(s)

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

Primary assignee: Vishal Thapar, vthapar, <vishal.thapar@ericsson.com>

Other contributors: <developer-b>, <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

None.

Testing

TBD.

Unit Tests

Any existing UTs will be enhanced accordingly.

Integration Tests

Any existing UTs will be enhanced accordingly.

CSIT

Following test cases will be added to NAT suite:

- Test NAPT base weight 0 * This should work like existing logic
- Test NAPT base weight 2 * Create 4 VMs in 2 VRFs on both computes * Compute with base weight 2 is designated for both VRFs

Documentation Impact

Refer section [Configuration Changes]

References

None.

1.2 NetVirt CSIT

1.2.1 OpenStack Upgrades

Contents

The NetVirt CSIT relies heavily on DevStack which changes heavily with each OpenStack release. This section will detail how the NetVirt CSIT should be adapted to these changes.

Note: This section should be used only as a guide. Each new release brings in a wide variety of changes that may or may not be captured below.

The table below lists the tasks to adapt CSIT to the new OpenStack versions.

Table 2: CSIT OpenStack Upgrade Tasks

Task	Notes
Tempest exclusions	https://git.opendaylight.org/gerrit/69001
Cirros flavors	https://git.opendaylight.org/gerrit/68971
Update run-test.sh	https://git.opendaylight.org/gerrit/68800
Update networking-odl plugins in local.conf	https://git.opendaylight.org/gerrit/68800
Update networking-l2gw plugins in local.conf	https://git.opendaylight.org/gerrit/68441
Update OpenStack CLI	https://git.opendaylight.org/gerrit/68687
Update workarounds	Look for places where master and other recent branches are used
Update CSIT jobs	https://git.opendaylight.org/gerrit/68909

Table of Contents

1.2.2 Troubleshooting Netvirt Datapath

Opendaylight Netvirt programs specific flows to OVS, for the various VM connectivity usecases to work. The purpose of this document is to give a detailed picture of the various flows that happen on OVS when a packet arrives.

Openflow Table Ownership

TABLE NUMBER	TABLE NAME	OWNERSHIP
0	INTERFACE INGRESS TABLE	GENIUS - INTERFACEMANAGER
17	INGRESS DISPATCHER TABLE	GENIUS - INTERFACEMANAGER
18	EXTERNAL TUNNEL DHCP TABLE	NETVIRT - L2GW SERVICE
19	GATEWAY MAC TABLE	NETVIRT - L3VPN
20	L3 LFIB TABLE	NETVIRT - L3VPN
21	L3 FIB TABLE	NETVIRT - L3VPN
22	L3 SUBNET ROUTE TABLE	NETVIRT - L3VPN
25	Floating IP to Internal IP Translation Table	NETVIRT - NAT
26	Internal IP to FIP/ External IP Translation Table	NETVIRT - NAT
27	Intermediate Pre-FIB Table after Reverse Translation	NETVIRT - NAT
28	Intermediate Pre-FIB Table after Forward Translation	NETVIRT - NAT
36	Internal Terminating Service Table	ALL SERVICES(which require communication over v
38	External Terminating Service Table	NETVIRT - L2GW SERVICE
43	ARP Check Table	NETVIRT - ELAN
44	Inbound Translation in NAPT vSwitch	NETVIRT - NAT
45	IPv6 Table	NETVIRT - IPV6
46	Outbound Translation in NAPT vSwitch	NETVIRT - NAT
47	NAPT vSwitch Pre-FIB Table	NETVIRT - NAT
48	ELAN DestIpToDMac Table	NETVIRT - ELAN

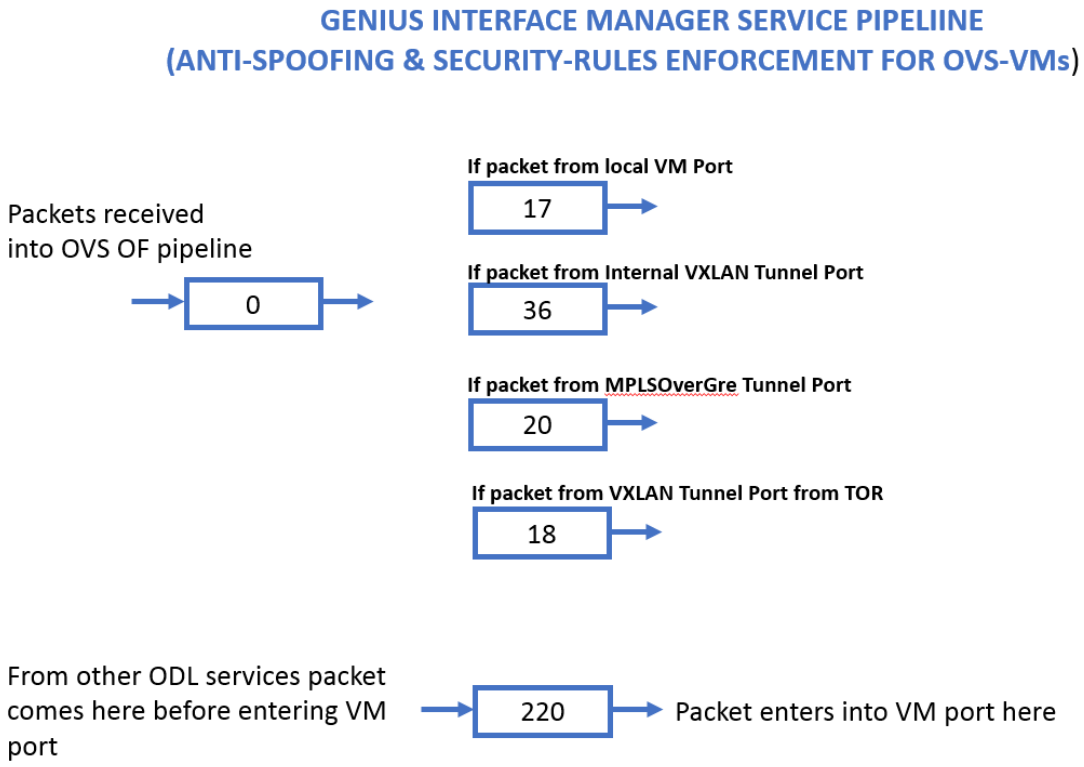
Continued on next p

Table 3 – continued from previous page

TABLE NUMBER	TABLE NAME	OWNERSHIP
49	Temporary Source MAC Learned Table	NETVIRT - ELAN
50	ELAN SMAC Table	NETVIRT - ELAN
51	ELAN DMAC Table	NETVIRT - ELAN
52	ELAN Unknown DMAC Table	NETVIRT - ELAN
55	ELAN Filter Equals Table	NETVIRT - ELAN
60	DHCP Table	NETVIRT - DHCP
80	L3 Interface Table	NETVIRT - L3VPN
81	ARP Responder Table	NETVIRT - L3VPN
210	Ingress ACL Anti-spoofing table	NETVIRT - ACL
211	Ingress ACL Conntrack classifier table	NETVIRT - ACL
212	Ingress ACL Conntrack sender table	NETVIRT - ACL
213	Applying ACL for existing Ingress traffic table	NETVIRT - ACL
214	Ingress ACL Filter cum dispatcher table	NETVIRT - ACL
215	Ingress ACL filter table	NETVIRT - ACL
216	Ingress Remote ACL filter table	NETVIRT - ACL
217	Ingress ACL committer table	NETVIRT - ACL
220	Interface Egress Dispatcher Table	GENIUS - INTERFACEMANAGER
239	Clear Egress conntrack state table	NETVIRT - ACL
240	Egress ACL Anti-spoofing table	NETVIRT - ACL
241	Egress ACL Conntrack classifier table	NETVIRT - ACL
242	Egress ACL Conntrack sender table	NETVIRT - ACL
243	Applying ACL for existing Egress traffic table	NETVIRT - ACL
244	Egress ACL Filter cum dispatcher table	NETVIRT - ACL
245	Egress ACL filter table	NETVIRT - ACL
246	Egress Remote ACL filter table	NETVIRT - ACL
247	Egress ACL committer table	NETVIRT - ACL

Genius InterfaceManager Pipeline

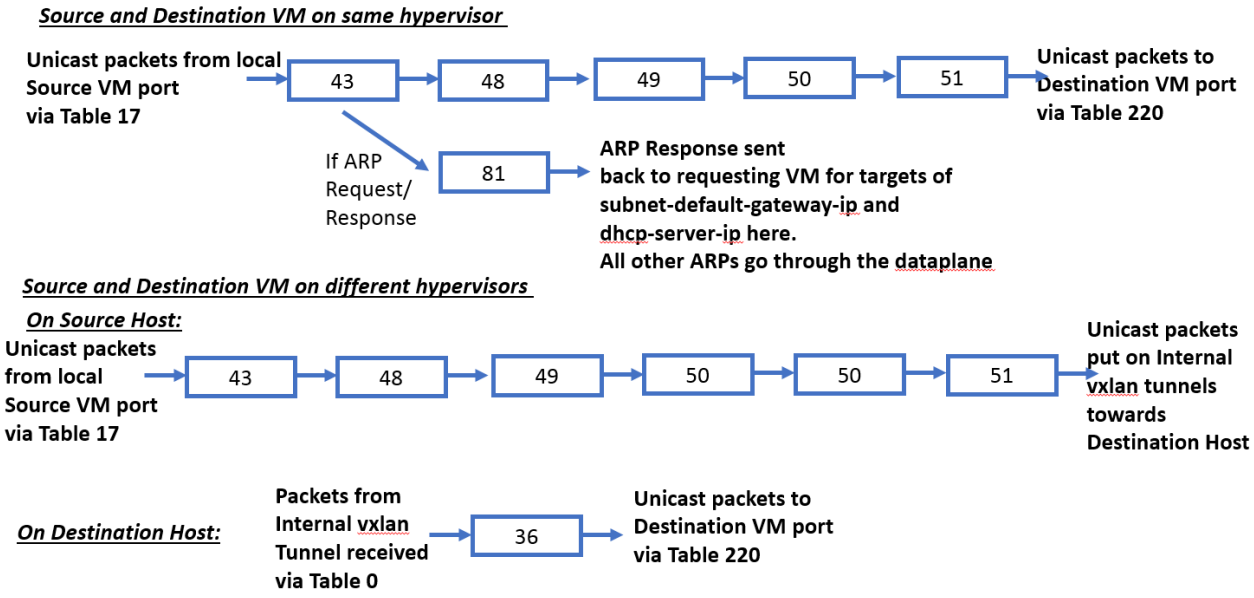
Netvirt uses Genius interface-manager to program ingress and egress flows for VMs as well as Tunnel ports. interface-manager is also used for binding multiple services on the same interface. A high level overview of the pipeline for ingress/egress is shown below in the diagram. This will be applicable for all service traffic flows explained in the subsequent sections.



Traffic Flows in Netvirt

ELAN Traffic Flow

Traffic Type	FLOW
Known unicast traffic flow(both direction)	Table 0 => Table 17 => Table 43 => Table 48 => Table 49 => Table 50 => Table 51 => Table 220 => Output Port
Unknown unicast/ multicast/broadcast traffic	Table 0 => Table 17 => Table 43 => Table 50 => Table 51 => Table 52 => Remote BC Group => Local BC Group => Table 55 => Table 220 => Output Port



L3VPN Traffic Flow

Traffic Type	FLOW
L3VPN Traffic Flow within same DPN	Table 0 => Table 17 => Table 19 => Table 21 => Local nexthop Group => Table 220 => output VM port
L3VPN Traffic Flow across DPNs within Data Center(source DPN)	Table 0 => Table 17 => Table 19 => Table 21 => Table 220 => Output tunnel port
L3VPN Traffic Flow across DPNs within DC(destination)	Table 0 => Table 36 => Table 220 => Output VM port
L3VPN Traffic Flow across DC(towards DC)	Table 0 => Table 17 => Table 19 => Table 21 => push MPLS, => Table 220 => output tunnel port
L3VPN Traffic Flow across DC(from DC)	Table 0 => Table 20 => Local nexthop group => Table 220 => output tunnel port

NAT Traffic Flow

DNAT Traffic Flow

Traffic Type	FLOW
DNAT Traffic Flow on source DPN	Table 0 => Table 20 => Table 25 => Table 27 => Table 21 => Local nexthop Group => Table 220 =>Output port
DNAT Traffic Flow on destination DPN	Table 0 => Table 17 => Table 21 => Table 26 => Table 28 => Table 21 => External Tunnel Groups

SNAT Traffic Flow

- SNAT VM Residing on the NAPT vSwitch

Traffic Type	FLOW
DPN (source traffic)	Table 0 => Table 17 => Table 21 => Table 26 => Table 46 => Table 47 => Table 21 => External Tunnel Groups
DPN (reverse traffic)	Table 0 => Table 20 => Table 44 => Table 47 => Table 21 => Local nexthop Group => Table 220 => output port

- SNAT VM Residing on non-NAPT vSwitch (Source Traffic)

Traffic Type	FLOW
DPN (source traffic)	Table 0 => Table 17 => Table 21 => Table 26 => Internal Tunnel Group => Table 220 => output tunnel port
NAPT DPN (reverse traffic)	Table 0 => Table 36 => Table 46 => Table 47 => Table 21 => External Tunnel Group => Table 220 => Output port

- SNAT VM Residing on non-NAPT vSwitch (Reverse Traffic)

Traffic Type	FLOW
NAPT DPN (source traffic)	Table 0 => Table 20 => Table 44 => Table 47 => Table 21 => Internal Tunnel Group => Table 220 => output port
DPN (reverse traffic)	Table 0 => Table 36 => Local nexthop Group => Table 220 => output port

- Conntrack Based SNAT Traffic Flow

<TBD>

SubnetRoutes Traffic flow

SubnetRoute Traffic Flow with ITM – EGRESS FROM A DPN

SubnetRoute Traffic Flow with ITM – LEARNING INVISIBLE IP

SubnetRoute Traffic Flow with ITM – Subsequent Packet from/to INVISIBLE IP

ACL/Security Groups Traffic Flow

VM Egress

VM Ingress with ITM

VM Ingress with ITM

VM Egress anti-spoofing

VM Ingress anti-spoofing with ITM

Inputs given by

- Akash Sahu
- Chetan Arakere Gowdru
- Faseela K
- Kiran N Upadhyaya
- Manu B
- N Vivekanandan
- Shashidhar Raja

CHAPTER

TWO

NETVIRT DEVELOPER GUIDE

CHAPTER

THREE

NETVIRT INSTALLATION GUIDE

CHAPTER

FOUR

OPENSTACK WITH NETVIRT

4.1 OpenStack with NetVirt

Table of Contents

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 and their dependencies:

```
./bin/client # Connect to OpenDaylight with the client
opendaylight-user@root> feature:install odl-netvirt-openstack odl-mdsal-apidocs
```

Optional - Advanced OpenDaylight Installation - Configurations and Clustering

- ACL Implementation - Security Groups - Stateful:
  - Default implementation used is stateful, requiring OVS compiled with conntrack modules.
  - This requires using a linux kernel that is >= 4.3
  - To check if OVS is running with conntrack support:

```
root@devstack:~/# lsmod | grep conntrack | grep openvswitch
  nf_conntrack          106496  9 xt_CT,openvswitch,nf_nat,nf_nat_ipv4,xt_
↪conntrack,nf_conntrack_netlink,xt_connmark,nf_conntrack_ipv4,nf_conntrack_
↪ipv6
```

  - If the conntrack modules are not installed for OVS, either recompile/install an OVS version with conntrack support, or alternatively configure OpenDaylight to use a non-stateful implementation.
  - OpenvSwitch 2.5 with conntrack support can be acquired from this repository for yum based linux distributions:

```
yum install -y http://rdoproject.org/repos/openstack-newton/rdo-release-
↪newton.rpm
yum install -y --nogpgcheck openvswitch
```

  - To spwan a VM with security groups disabled:

```
crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 extension_drivers␣
↪port_security

openstack port create --network=net1 --disable-port-security port1
openstack server create --flavor m1.tiny --image cirros --port port1 vm1
```

- 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:172.16.21.56:6653"
             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 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 8181 (which will be used by OpenDaylight to listen for REST calls) is available. 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="8181" />
</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}:8181/controller/nb/v2/neutron
password = admin
username = admin
EOT
```

- Configure Neutron to use OpenDaylight's odl-router service plugin for L3 connectivity:

```
crudini --set /etc/neutron/plugins/neutron.conf DEFAULT service_plugins odl-router
```

- Configure Neutron DHCP agent to provide metadata services:

```
crudini --set /etc/neutron/plugins/dhcp_agent.ini DEFAULT force_metadata True
```

Note:

If the OpenStack version being used is Newton, this workaround should be applied,
configuring the Neutron DHCP agent to use vsctl as the OVSDB interface:

```
crudini --set /etc/neutron/plugins/dhcp_agent.ini OVS ovsdb_interface vsctl
```

- Reset Neutron's database

```
mysql -e "DROP DATABASE IF EXISTS neutron;"
mysql -e "CREATE DATABASE neutron CHARACTER SET utf8;"
/usr/local/bin/neutron-db-manage --config-file /etc/neutron/neutron.conf --config-
↪file /etc/neutron/plugins/ml2/ml2_conf.ini upgrade head
```

- Restart neutron-server:

```
systemctl start neutron-server
```

Verifying it works

- Verify that OpenDaylight's ML2 interface is working:

```
curl -u admin:admin http://{CONTROL_HOST}:8181/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
```

(continues on next page)

(continued from previous page)

```
neutron router-gateway-set router1 public-net

neutron floatingip-create public-net
nova floating-ip-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`

```
udhcpc (v1.20.1) started
Sending discover...
Sending select for 10.0.123.3...
Lease of 10.0.123.3 obtained, lease time 86400 # This only happens when DHCP␣
↪is properly obtained.
```

- Check if the DHCP requests are reaching the qdhcp agent using the following commands on the OpenStack controller:

```
sudo ip netns
sudo ip netns exec qdhcp-xxxxx ifconfig # xxxx is the neutron network id
sudo ip netns exec qdhcp-xxxxx tcpdump -nei tapxxxxx # xxxxx is the neutron port␣
↪id

# Valid request and response:
```

(continues on next page)

(continued from previous page)

```
15:08:41.684932 fa:16:3e:02:14:bb > ff:ff:ff:ff:ff:ff, ethertype IPv4 (0x0800),␣
↪length 329: 0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from␣
↪fa:16:3e:02:14:bb, length 287
15:08:41.685152 fa:16:3e:79:07:98 > fa:16:3e:02:14:bb, ethertype IPv4 (0x0800),␣
↪length 354: 10.0.123.2.67 > 10.0.123.3.68: BOOTP/DHCP, Reply, length 312
```

- If the requests aren't reaching qdhcp:
  - Verify VXLAN tunnels exist between compute and control nodes by using `ovs-vsctl show`
  - Run the following commands to debug the OVS processing of the DHCP request packet:
    `ovs-ofctl -OOpenFlow13 dump-ports-desc br-int` # retrieve VMs ofport and MAC
    `ovs-appctl ofproto/trace br-int in_port=<ofport>,dl_src=<mac>, dl_dst=ff:ff:ff:ff:ff:ff,udp,ip_src=0.0.0.0,ip_dst=255.255.255.255 | grep "Rule\|action"`

```
root@devstack:~# ovs-appctl ofproto/trace br-int in_port=1,dl_
↪src=fe:16:3e:33:8b:d8,dl_dst=ff:ff:ff:ff:ff:ff,udp,ip_src=0.0.0.0,ip_
↪dst=255.255.255.255 | grep "Rule\|action"
    Rule: table=0 cookie=0x8000000 priority=1,in_port=1
    OpenFlow actions=write_metadata:0x20000000001/0xffffff0000000001,goto_
↪table:17
        Rule: table=17 cookie=0x8000001 priority=5,metadata=0x20000000000/
↪0xffffff0000000000
        OpenFlow actions=write_metadata:0xc0000200000222e2/0xfffffffffffffffe,
↪goto_table:19
            Rule: table=19 cookie=0x1080000 priority=0
            OpenFlow actions=resubmit(,17)
                Rule: table=17 cookie=0x8040000 priority=6,
↪metadata=0xc000020000000000/0xffffff0000000000
                OpenFlow actions=write_metadata:0xe00002138a000000/
↪0xfffffffffffffffe,goto_table:50
                    Rule: table=50 cookie=0x8050000 priority=0
                    OpenFlow actions=CONTROLLER:65535,goto_table:51
                        Rule: table=51 cookie=0x8030000 priority=0
                        OpenFlow actions=goto_table:52
                            Rule: table=52 cookie=0x870138a priority=5,
↪metadata=0x138a000001/0xffff000001
                            OpenFlow actions=write_actions(group:210003)
    Datapath actions: drop

root@devstack:~# ovs-ofctl -OOpenFlow13 dump-groups br-int | grep 'group_
↪id=210003'
    group_id=210003,type=all
```

- If the requests are reaching qdhcp, but the response isn't arriving to the VM:
  - Locate the compute the VM is residing on (can use `nova show <vm>`).
    - If the VM is on the same node as the qdhcp namespace, `ofproto/trace` can be used to track the packet:
      `ovs-appctl ofproto/trace br-int in_port=<dhcp_ofport>,dl_src=<dhcp_port_mac>,dl_dst=<vm_port_mac>, udp,ip_src=<dhcp_port_ip>,ip_dst=<vm_port_ip> | grep "Rule\|action"`

```
root@devstack:~# ovs-appctl ofproto/trace br-int in_port=2,dl_
↪src=fa:16:3e:79:07:98,dl_dst=fa:16:3e:02:14:bb,udp,ip_src=10.0.123.2,ip_
↪dst=10.0.123.3 | grep "Rule\|action"
    Rule: table=0 cookie=0x8000000 priority=4,in_port=2
    OpenFlow actions=write_metadata:0x10000000000/0xffffff0000000001,goto_
↪table:17
        Rule: table=17 cookie=0x8000001 priority=5,metadata=0x10000000000/
↪0xffffff0000000000
        OpenFlow actions=write_metadata:0x60000100000222e0/
↪0xfffffffffffffffe,goto_table:19
            Rule: table=19 cookie=0x1080000 priority=0
            OpenFlow actions=resubmit(,17)
                Rule: table=17 cookie=0x8040000 priority=6,
↪metadata=0x6000010000000000/0xffffff0000000000
                OpenFlow actions=write_metadata:0x7000011389000000/
↪0xfffffffffffffffe,goto_table:50
                    Rule: table=50 cookie=0x8051389 priority=20,
↪metadata=0x11389000000/0xffffffffff000000,dl_src=fa:16:3e:79:07:98
                    OpenFlow actions=goto_table:51
                        Rule: table=51 cookie=0x8031389 priority=20,
↪metadata=0x1389000000/0xffff000000,dl_dst=fa:16:3e:02:14:bb
                        OpenFlow actions=load:0x300->NXM_NX_REG6[],
↪resubmit(,220)
                            Rule: table=220 cookie=0x8000007 priority=7,
↪reg6=0x300
                            OpenFlow actions=output:3
```

* If the VM isn't on the same node as the qdhcp namepsace:
  - Check if the packet is arriving via VXLAN by running `tcpdump -nei <vxlan_port> port 4789`
  - If it is arriving via VXLAN, the packet can be tracked on the compute node rules, using `ofproto/trace` in a similiar manner to the previous section. Note that packets arriving from a tunnels have a unique tunnel_id (VNI) that should be used as well in the trace, due to the special processing of packets arriving from a VXLAN tunnel.

Floating IP Issues

- If you have assigned an external network and associated a floating IP to a VM but there is still no connectivity:
  - Verify the external gateway IP is reachable through the provided provider network port.
  - Verify OpenDaylight has successfully resolved the MAC address of the external gateway IP. This can be verified by searching for the line "Installing ext-net group" in the karaf.log.
  - Locate the compute the VM is residing on (can use `nova show <vm>`).
  - Run a ping to the VM floating IP.
  - If the ping fails, execute a flow dump of br-int, and search for the flows that are relevant to the VM's floating IP address: `ovs-ofctl -OOpenFlow13 dump-flows br-int | grep "<floating_ip>"`
    * Are there packets on the incoming flow (matching dst_ip=<floating_ip>)?
      If not this probably means the provider network has not been set up properly, verify provider_mappings configuration and the configured external network physical_network value match. Also verify that the Flat/VLAN network configured is actually reachable via the configured port.

* Are there packets on the outgoing flow (matching src_ip=<floating_ip>)?
  If not, this probably means that OpenDaylight is failing to resolve the MAC of the provided external gateway, required for forwarding packets to the external network.
* Are there packets being sent on the external network port?
  This can be checked using `tcpdump <port>` or by viewing the appropriate OpenFlow rules. The mapping between the OpenFlow port number and the linux interface can be acquired using `ovs-ofctl dump-ports-desc br-int`

```
ovs-ofctl -OOpenFlow13 dump-flows br-int | grep "<floating_ip>"
cookie=0x8000003, duration=436.710s, table=21, n_packets=190, n_
 ↪bytes=22602, priority=42,ip,metadata=0x222e2/0xfffffffe,nw_dst=10.64.98.
 ↪17 actions=goto_table:25
cookie=0x8000004, duration=436.739s, table=25, n_packets=190, n_
 ↪bytes=22602, priority=10,ip,nw_dst=10.64.98.17 actions=set_field:10.0.
 ↪123.3->ip_dst,write_metadata:0x222e0/0xfffffffe,goto_table:27
cookie=0x8000004, duration=436.730s, table=26, n_packets=120, n_
 ↪bytes=15960, priority=10,ip,metadata=0x222e0/0xfffffffe,nw_src=10.0.123.
 ↪3 actions=set_field:10.64.98.17->ip_src,write_metadata:0x222e2/
 ↪0xfffffffe,goto_table:28
cookie=0x8000004, duration=436.728s, table=28, n_packets=120, n_
 ↪bytes=15960, priority=10,ip,metadata=0x222e2/0xfffffffe,nw_src=10.64.98.
 ↪17 actions=set_field:fa:16:3e:ec:a8:84->eth_src,group:200000
```

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

CHAPTER

FIVE

NETVIRT USER GUIDE

5.1 Carbon Features

Contents

Major Feature	Sub Feature	sub-category	Status	Comme
Security Groups	conntrack		Full	ovs 2.6 r
	learn		Full	alternativ
	transparent		Full	no-ops s
	allowed_address_pairs		Full	
L2	east-west	vlan provider	Full	
		flat provider	Full	
		vxlan provider	Full	
	north-south (external n/w)	vlan provider	Full	no BGP
		flat provider	Full	
		vxlan provider	WIP	using EV
	vlan aware vms	vxlan provider	Full	
	vlan transparency	vxlan provider	Full	None wi
	MAC learning	vlan, vxlan provider	Full	
ARP	ARP supression for Neutron Router		Full	
	distributed ARP responses for floating IPs	vlan provider	Full	
L3 - IPV4	east-west	vlan provider	None	
		vxlan provider	Full	DVR
		gre provider	None	
	east-west with ECMP	vxlan provider	Full	
	north-south (external n/w)	vlan provider	Full	no BGP
		vxlan provider	Full	with EVI
		gre provider	None	needs OV
	floating-ip	vlan, vxlan, gre provider	Full	gre need
	snat - centralized-Controller	vlan, vxlan, gre provider	Full	Using co
	snat - centralized- Conntrack	vlan, vxlan, gre provider	Full	Used OV

Major Feature	Sub Feature	sub-category	Status	Comme
	Extra routes	vxlan provider	Full	
	Invisible IP learning	vxlan provider	Full	Uses cor
Transport	auto-bridge creation		Full	br-int
	auto-tunnel creation		Full	vxlan tur
	pre-configured full mesh of tunnels between vswitches		Full	
	VxLAN tunnels with IPv6 endpoints (requires OVS 2.6)			
DHCP	Neutron-based	IPv4, IPv6	Full	
	Controller-based	IPv4	Full	Missing
	Metadata Server	Neutron-based	Full	Requires
IPv6 control	IPv6 IP SLAAC	vlan, vxlan	Full	Detailed
	IPv6 RAs	vlan, vxlan	Full	
IPV6 forwarding				
	Security Groups, allowed address pairs		Full	
IPv6 L2	east-west	vlan provider		
		flat provider		
		vxlan provider	Full	
	north-south (external n/w)	vlan provider	?	no BGP
		flat provider	?	
		vxlan provider		
	north-south VPN connectivity	vlan provider		
		vxlan provider		
		gre provider		
	vlan aware vms	vxlan provider	Full	no CSIT
	vlan transparency	vxlan provider	Full	None wi
	MAC learning	vlan, vxlan provider	Full	
ND	ND supression for Neutron Router		WIP	
IPv6 L3	east-west	vlan provider	?	
		vxlan provider	Full	DVR
		gre provider		
	north-south (external n/w)	vlan provider	Full	no BGP
		vxlan provider		
		gre provider	WIP	needs OV
	north-south VPN connectivity	vlan provider		
		vxlan provider		
		gre provider	Full	needs OV
	Extra routes	vxlan, vlan? provider		
	Invisible IP learning	vxlan, vlan? provider		
L2GW/HWVTEP	L2 connectivity with PNF/baremetal		Full	Vlan tran
	SR-IOV with multi-provider extension		Full	upstream
	L3		None	target for
	HA	active/active, active/backup	Full	
	DHCP service for SR-IOV endpts		Full	using a d

Major Feature	Sub Feature	sub-category	Status	Comme
VM Migration	all features		Full	
QoS	rate limiting		Full	
	DSCP marking		Full	
	meters		Full	needs O
	supported rule types		None	
	minimum bandwidth rule		None	
Federation	L2 Connectivity between OpenStack sites		Full	
	L3 Connectivity between OpenStack sites		Full	
	ACLs federation, micro-segmentation between sites		Full	
Statistics	REST based statistics for neutron elements		Full	
	Element to Element counters		Full	
Neutron APIs	network		Full	
	subnet		Full	
	port		Full	
	router		Full	
	security-groups		Full	
	floating-ips		Full	
	external-networks		Full	
	provider-network		Full	
	allowed address pairs		Full	
	L2GW		Full	v2 drive
	bgpvpn		Full	v2 drive
	vlan aware vms		Ocata only	Ocata on
	extra routes		Full	
	QoS		Full	
	pseudo-agent binding, host config		Full	Default
ETREE			WIP	
DPDK			Full	conn-tra
Datapath	NSH based	WIP		
		non-NSH based	WIP	
	Northbound	WIP		
		WIP		
LBaaS	Octavia implementation		Partial	L2 supp
	ODL implementation		Future	
PNFs	Connectivity to PNFs on Flat/VLAN networks	L2	Full	
		L3	Full	
		External Network	Full	

5.2 Version Compatibility

Contents

This section will detail version compatibilities between OpenDaylight and other components.

5.2.1 Open_vSwitch Kernel and DPDK Modes

The table below lists the Open_vSwitch requirements for the Carbon release.

Table 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 L3VPN Service: User Guide

5.3.1 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.

5.3.2 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/if:interfaces/interface*
   a. Fields :
      i. name
      ii. type
      iii. enabled
      iv. of-port-id
      v. tenant-id
      vi. base-interface
   b. type specific fields
      i. when type = *l2vlan*
         A. vlan-id
      ii. when type = *stacked_vlan*
         A. stacked-vlan-id
      iii. when type = *l3tunnel*
         A. tunnel-type
         B. local-ip
         C. remote-ip
         D. gateway-ip
      iv. when type = *mpls*
         A. list labelStack
         B. num-labels

c. REST Methods : GET, PUT, DELETE, POST

1. Data Node Path : */config/odl-fib:fibEntries/vrfTables*
   a. Fields :
      i. List of vrfTables
   b. REST Methods : GET, PUT, DELETE, POST
2. Data Node Path : */config/odl-fib:fibEntries/vrfTables/``{routeDistinguisher}``/*
   a. Fields :
      i. route-distinguisher
      ii. list vrfEntries
         A. destPrefix
         B. label
         C. nexthopAddress
   b. REST Methods : GET, PUT, DELETE, POST
3. Data Node Path : */config/odl-fib:fibEntries/ipv4Table*
   a. Fields :
      i. list ipv4Entry
         A. destPrefix
         B. nexthopAddress
   b. REST Methods : GET, PUT, DELETE, POST

5.3.3 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/if:interfaces/if:interfacce*

Sample JSON Data

```
{
    "interface": [
        {
            "name" : "GRE_192.168.57.101_192.168.57.102",
            "type" : "odl-interface:l3tunnel",
            "odl-interface:tunnel-type": "odl-interface:tunnel-type-gre",
            "odl-interface:local-ip" : "192.168.57.101",
            "odl-interface:remote-ip" : "192.168.57.102",
            "odl-interface:portId" : "openflow:1:3",
            "enabled" : "true"
        }
    ]
}
```

Following is expected as a result of these configurations

1. Unique If-index is generated
2. *Interface-state* operational DS is updated
3. Corresponding Nexthop Group Entry is created

Step 3 : OS Create Neutron Ports and attach VMs

At this step user creates VMs.

Step 4 : Create VM Interfaces

Create l2vlan interfaces corresponding to VM created in step 3

REST API : *PUT /config/if:interfaces/if:interface*

Sample JSON Data

```
{
    "interface": [
        {
            "name" : "dpn1-dp1.2",
            "type" : "l2vlan",
            "odl-interface:of-port-id" : "openflow:1:2",
            "odl-interface:vlan-id" : "1",
            "enabled" : "true"
        }
    ]
}
```

Step 5: Create VPN Instance

REST API : *PUT /config/l3vpn:vpn-instances/l3vpn:vpn-instance/*

Sample JSON Data

```
{
  "vpn-instance": [
    {
        "description": "Test VPN Instance 1",
        "vpn-instance-name": "testVpn1",
        "ipv4-family": {
            "route-distinguisher": "4000:1",
            "export-route-policy": "4000:1,5000:1",
            "import-route-policy": "4000:1,5000:1",
        }
    }
  ]
}
```

Following is expected as a result of these configurations

1. VPN ID is allocated and updated in data-store
2. Corresponding VRF is created in BGP
3. If there are vpn-interface configurations for this VPN, corresponding action is taken as defined in step 5

Step 5 : Create VPN-Interface and Local Adjacency

this can be done in two steps as well

1. Create vpn-interface

REST API : *PUT /config/l3vpn:vpn-interfaces/l3vpn:vpn-interface/*

Sample JSON Data

```
{
  "vpn-interface": [
    {
      "vpn-instance-name": "testVpn1",
      "name": "dpn1-dp1.2",
    }
  ]
}
```

Note: name here is the name of VM interface created in step 3, 4

2. Add Adjacencies on vpn-interafce

REST API : *PUT /config/l3vpn:vpn-interfaces/l3vpn:vpn-interface/dpn1-dp1.3/adjacency*

Sample JSON Data

```
{
   "adjacency" : [
           {
               "ip-address" : "169.144.42.168",
               "mac-address" : "11:22:33:44:55:66"
           }
     ]
 }
```

its a list, user can define more than one adjacency on a vpn_interface

Above steps can be carried out in a single step as following

```
{
    "vpn-interface": [
        {
            "vpn-instance-name": "testVpn1",
            "name": "dpn1-dp1.3",
```

(continues on next page)

(continued from previous page)

```
            "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

5.4 Support

Table of Contents

5.4.1 Verified Combinations

This section describes different combinations of OpenStack and Open vSwitch versions that are expected to work with with different OpenDaylight versions. Using combinations outside this list may work but have not been verified.

Note: A verified combination is defined as a combinations that is 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 3: Supported Version Matrix

OpenDaylight	OpenStack	Open vSwitch	Sync	Notes
Carbon	Ocata	2.7		Combination drops when Pike releases
Carbon	Pike	2.8	S	
Nitrogen	Ocata	2.7		Combination drops when Pike releases
Nitrogen	Pike	2.9	S	
Oxygen	Pike	2.8		Combination drops when Queens releases
Oxygen	Queens	2.9	S	
Fluorine	Queens	2.9		Combination drops when Rocky releases
Fluorine	Rocky	2.11	S	
Neon	Rocky	2.11		Combination drops when Stein releases
Neon	Stein	2.11	S	
Sodium	Rocky	2.11		Combination drops when Train releases
Sodium	Train	2.11	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.

5.4.2 Open vSwitch Kernel and DPDK Modes

The table below lists the Open vSwitch requirements for the Carbon release.

Table 4: 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

5.5 Bridge Configuration

Table of Contents

The following describes OVS bridge configurations supported by OpenDaylight.

5.5.1 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.

5.5.2 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`.

5.6 Debugging Netvirt/Networking-odl Port Status Update

Table of Contents

Recent versions of Opendaylight Netvirt support dynamic update of Neutron port status via websocket and the networking-odl ML2 plugin knows how to take advantage of this feature. The following is a basic description of the internals of this feature followed by a guide for how to debug it.

5.6.1 How Port Status Update Works

When Neutron/networking-odl boots it registers a websocket based subscription to all neutron ports in the ODL operational yang model. Once this websocket subscription is connected, networking-odl receives json based notifications for every time Netvirt changes the status of a port. Note that for now this only happens at the time of port binding, if a port should go down, e.g., the VM crashes, Netvirt will not update the port status.

Note that a failure of port status update is not the only error that can cause a port to not transition to ACTIVE. Neutron uses a mechanism called provisioning-blocks to make sure two things happen before a port is transitioned to ACTIVE:

- Port status update to ACTIVE
- DHCP provisioning by the DHCP agent

If the DHCP provisioning does not succeed, e.g., if the dhcp agent is down, the port will not transition to active. In order to determine if DHCP provisioning worked look for *either* of the following two lines in the neutron logs:

```
Provisioning for port <uuid of port> completed by entity DHCP.
```

```
Provisioning complete for port <uuid of port> triggered by entity
DHCP.
```

5.6.2 How to Debug Port Status

1) BEFORE YOU GET STARTED, SET THE LOG LEVELS

Neutron logs need to be set to debug. You can do this by having "debug = True" in your neutron.conf, generally under /etc/neutron.

The following ODL component log levels need to be set in <your karaf installation>/etc/org.ops4j.pax.logging.cfg:

```
log4j2.logger.npcl.level = TRACE
log4j2.logger.npcl.name =org.opendaylight.netvirt.neutronvpn.NeutronPortChangeListener
log4j2.logger.nu.level = DEBUG
log4j2.logger.nu.name =org.opendaylight.netvirt.neutronvpn.api.utils.NeutronUtils
log4j2.logger.oisah.level = INFO
log4j2.logger.oisah.name =org.opendaylight.genius.interfacemanager.renderer.ovs.
↪statehelpers.OvsInterfaceStateAddHelper
```

2) Check that the websocket is connected

Websocket connection status is logged in the neutron logs. Check that this line is the last websocket status logged before your port was created:

```
websocket transition to status ODL_WEBSOCKET_CONNECTED
```

Note: The connection can disconnect but should reconnect so you can have multiple lines like this in the log with different statuses. You can now follow the transitions in this case.

If the websocket is not connected, either something is wrong with your deployment or ODL is not running. It may be worth checking if a firewall is blocking the websocket port which is 8185 by default but may be custom configured in a file called 10-rest-connector.xml under your karaf installation.

3) Check whether networking-odl received the port status update

All port status notifications are logged in the neutron logs like this:

```
Update port for port id <uuid of port> <ACTIVE|DOWN>
```

Note that for VM ports Netvirt will initially report that the port is DOWN until the basic flows are configured.

If there is no log line like this reporting your port is ACTIVE it is best to...

4) Check whether Netvirt transitioned the port to ACTIVE

Look for the following in karaf.log:

```
writePortStatus:  operational port status for <uuid of port> set to
<ACTIVE|DOWN>
```

Again, remember that for VM ports the port is initially reported as DOWN and soon after as ACTIVE.

If this log line is missing…

5) Check whether the Neutron port was received by Netvirt

```
Adding Port :  key:  <iid of the port, including uuid>, value=<dump
of the port>
```

If this line is missing it means that something is wrong with the REST communication between networking-odl and ODL.

If this line is present but the line from (4) is not, it probably means that the southbound OpenFlow Port Status event was never received. Now…

6) Check whether the Genius operational port status was created

Check for this:

```
Adding Interface State to Oper DS for interface <tap interface name>
```

The tap interface name is the word "tap-" followed by the initial 7 characters of the neutron port's UUID. If this line is missing, you have confirmed that the southbound port was never received via openflowplugin. This could mean that the switch is not connected or perhaps the VM never booted.

7) Something deeper is wrong

Although unlikely, if you will have made it this far and still have no answer, something much deeper is wrong and a more serious debugging effort is required.

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