1 Purpose

This document describes the changes that are proposed in the L2 Plugin and the L2 Agent of OpenStack in order to enable distributed virtual router functionality at the L2 layer.

2 Design Goals for L2 in DVR

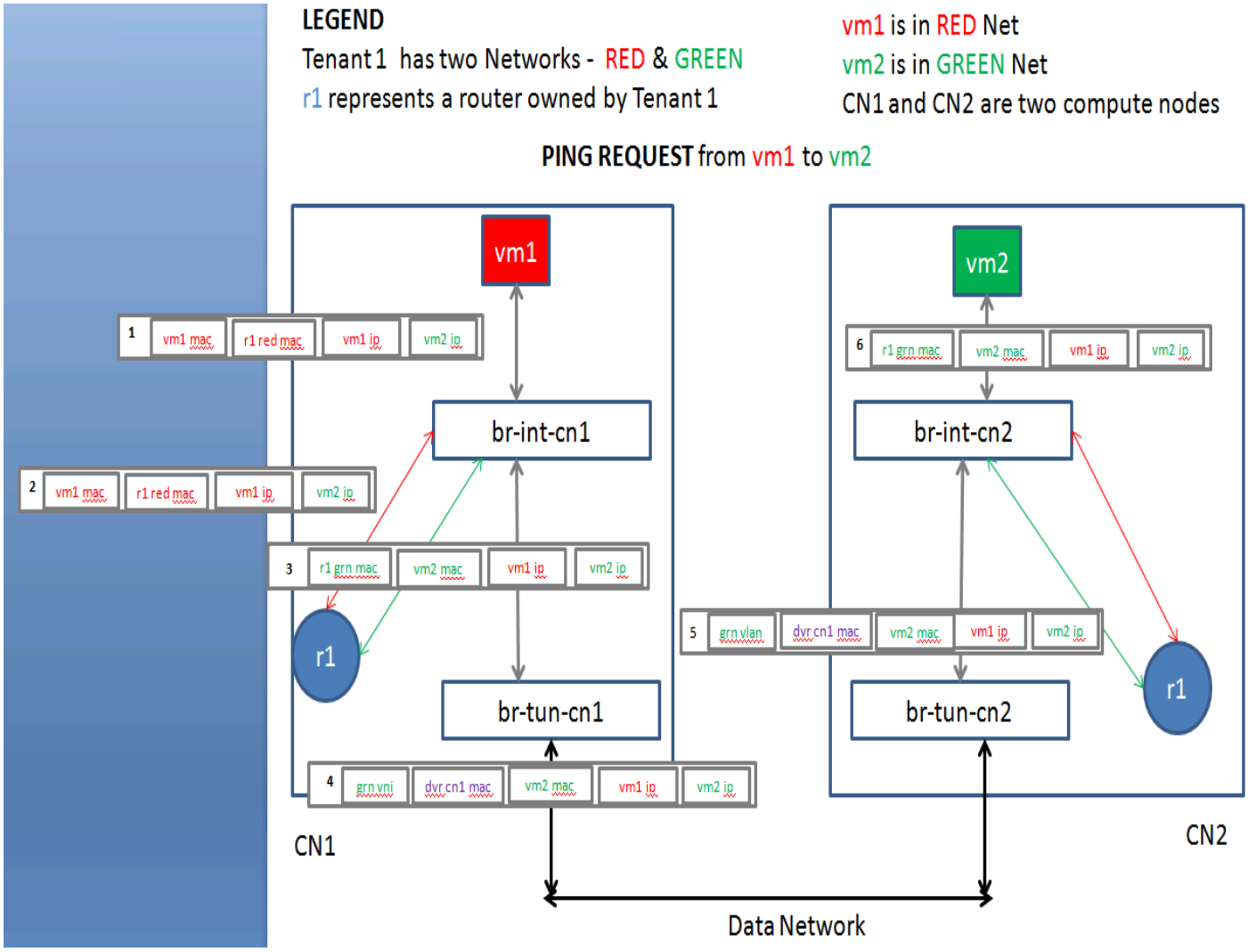
There are set of goals that went into the design of the set of rules in OVS to enable distributed virtual routing functionality at the L2 layer. They are the following:

1. The rules should enable communication between tenant VMs on different networks, via a distributed router hosted in the compute node.
2. Any given distributed router instance may reside on an all the Compute Nodes and will be responsible for routing packets generated by VMs co-residing on the same node.
3. A given distributed router instance is either hosted in the Compute Node as DVR (or) as normal router on the Network Node, but never simultaneously in both the nodes.
4. The rules should be such that the existing OpenVswitch infrastructure in OpenStack Neutron can be used to accomplish distributed routing.
5. Should use lesser MAC Address per compute node and continue to preserve L2 isolation for routed packets by distributed routers.
6. Should ensure enablement of North-South routing, which indicates access to external network by the VMs directly.
7. Should embrace FWaaS Service to be operational even with distributed routing enabled on DVR interfaces.
8. MAC timeouts and ARP timeouts of any element (bridges) inside the nodes in the cloud, should not affect operation of DVR
9. The rules must ensure that multi-tenant isolation is intact even when Overlapping IP Addresses is enabled in the cloud.
10. The rules must ensure that multi-tenant isolation is intact even when Overlapping MAC Addresses are enabled in the cloud (By overlapping MAC Addresses, what is meant here is that MAC Addresses across different networks could be the same)

3 How Distributed Routing works

In order to accomplish distributed routing, both the L3 and L2 Agent work will need to work hand-in-hand inside the Compute Node. Today the L3 Agent runs in Network Node, but with this DVR proposal, the L3 Agent will run in Compute Nodes as well. The L2 Agent on compute nodes will continue as is today but will work in an enhanced mode called the ‘DVR Mode’ where the L2 Agents will be additionally responsible for managing (adding/removing) OVS rules in

order to accomplish distributed routing.
The following is a sample topology used to illustrate how distributed routing is accomplished.



Terminology for the contents of packets shown in the above figure:

red1-L-vlan Represents the local vlan of the red network on Compute node CN1

grn1-L-vlan Represents the local vlan of the green network on Compute node CN1

red2-L-vlan Represents the local vlan of the red network on Compute node CN2

grn2-L-vlan Represents the local vlan of the green network on Compute node CN2

red-vni Represents the VXLAN ID used in the tunneled frame, for the tenant network represented by red-L-vlan local vlan.

grn-vni Represents the VXLAN ID used in the tunneled frame, for the tenant network represented by grn-L-vlan local vlan.

r1-red-ip Represents the IP Address of the red subnet interface of the DVR Router (r1). This IP Address for red subnet interface would remain the same on DVR Router (r1) instantiated in both CN1 and CN2.

r1-grn-ip Represents the IP Address of the green subnet interface of the DVR Router (r1). This IP Address for green subnet interface would remain the same on DVR Router (r1) instantiated in both CN1 and CN2.
r1-red-mac Represents the MAC Address of the red subnet interface of the DVR Router (r1). This MAC Address for the red subnet interface would remain the same on DVR Router (r1) instantiated in both CN1 and CN2.
r1-grn-mac à Represents the MAC Address of the green subnet interface of the DVR Router (r1). This MAC Address for the green subnet interface would remain the same on DVR Router (r1) instantiated in both CN1 and CN2.

dvr-cn1-mac Represents Mac Address allocated from DVR Base Mac pool by the controller to Compute Node CN1. This mac address will go as the source mac address in all the frames that are generated by the DVR subnet interfaces present in CN1.

dvr-cn2-mac Represents Mac Address allocated from DVR Base Mac pool by the controller to Compute Node CN2. This mac address will go as the source mac address in all the frames that are generated by the DVR subnet interfaces present in CN2.

In the above figure, a PING ECHO Request is initiated from vm1 on red network to vm2 on green network which are connected by a DVR router identified by r1. The DVR Router r1 will have the same IP Address and MAC Address on both the nodes CN1 and CN2. As we see, the DVR Router r1 has two interfaces: one interface is a subnet in the red network and another interface is a subnet in the green network.

The packet flow for **PING ECHO Request** from vm1 to vm2 is shown in the above figure with packets numbered from **'1' to '6'.**

1 The frame with destination ip as 'vm2 ip' is sent by vm1 towards its default gateway mac for red network which is r1-red-mac. The integration bridge forwards this frame to DVR router r1.
2 The DVR router r1's red subnet interface picks this frame and it then routes the IP packet in the frame.
3 After routing, the DVR router r1 puts the routed frame out of its green subnet interface. This frame is switched by the integration bridge towards the tunnel bridge along with tagging the frame with green network's local-vlan tag.
4 The tunnel bridge on CN1, replaces the frame's source mac address with a Unique DVR MAC Address of its node (one unique dvr mac address is assigned per compute node by the controller). The resulting frame is forwarded to CN2 by this tunnel bridge. Before forwarding, it also strips the local green-vlan tag and tunnels the frame with green-vni VXLAN id.
5 The tunnel bridge on CN2, picks up the tunneled frame, de-tunnels it and strips off the green-vni tag. It then adds its local green network vlan tag to the frame and forwards the frame to the integration bridge.

6 The integration bridge on CN2, identifies the incoming frame's source mac address is a DVR Unique MAC Address (every compute node l2-agent knows all dvr unique mac addresses used in the cloud). It then replaces the DVR Unique MAC Address with the green subnet interface MAC address and forwards the frame to vm2.

For the **PING RESPONSE** from vm2 to vm1, the same above sequence happens in reverse.

As you might notice, the frames are routed by the DVR router in the frame originating node itself and then they are just put towards the right destination. For all the routed frames, the DVR Unique MAC Address of the frame's originating node is used in the underlay, as the Source MAC Address of the frame (ie., in inner frame).

The ARP entry for vm2 will be pre-populated in DVR router r1 residing in CN1, by the L3-Agent running on CN1 (through information supplied from the L3 Plugin). Similarly the ARP entry for vm1 will be pre-populated in the DVR router r1 residing in CN2, again by the L3-Agent running on CN2 (through information supplied from the L3 Plugin).

On Compute Node CN1:

Integration Bridge Rules:

Table 0: (Local switching table)

table=0, priority=2, in_port=patch-tun, dl_src=dvr-cn2-mac actions: goto table 1

table=0, priority=1, actions: output->NORMAL

Table 1: (DVR_TO_LOCALMAC table)

table=1, priority=2, dl_vlan=red1-L-vlan, nw_dst=red-subnet actions: strip_vlan, mod_dl_src=r1-red-MAC,output->port-vm1

table=1, priority=1 actions: drop

***Tunnel Bridge Rules*:**

DEMUX TABLE Table 0 (existing):
table=0, priority=1,in_port=patch_int , actions=resubmit(,1)
table=0, priority=1,in_port=<tep-port> actions=resubmit(,4)
table=0,priority=0 actions=drop

DVR PROCESS Table 1 (New table for dvr):
table=1, priority=4, dl_vlan=red1-L-vlan, dl_type=arp, ar_tpa=r1-red-ip actions: drop
table=1, priority=4, dl_vlan=grn1-L-vlan, dl_type=arp, ar_tpa=r1-grn-ip actions: drop
table=1, priority=2,, dl_vlan=red1_L_vlan, dl_dst=r1-red-mac, actions: drop
table=1, priority=2, dl_vlan=grn1_L_vlan, dl_dst=r1-grn-mac, actions: drop
table=1, priority=1, dl_vlan=red1_L_vlan, dl_src=r1-red-mac, actions: mod_dl_src=dvr-cn1-mac, resubmit(,2)
table=1, priority=1, dl_vlan=grn1_L_vlan, dl_src=r1-grn-mac, actions: mod_dl_src=dvr-cn1-mac, resubmit (,2)
table=1, priority=0, actions: goto table 2

PATCH_LV_TO_TUN Table 2 (existing as Table 1):
table=2, priority=0,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,21)
table=2, priority=0,dl_dst=00:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,20)

GRE_TUN_TO_LV Table 3 (existing as Table 2):
table=3, priority=0 actions=drop

VXLAN_TUN_TO_LV Table 4 (existing as Table 3):
table=4, priority=1,tun_id=red-vni actions=mod_vlan_vid:red1-L-vlan,resubmit(,9)
table=4, priority=1,tun_id=grn-vni actions=mod_vlan_vid:grn1-L-vlan,resubmit(,9)
table=4, priority=0 actions=drop

DVR_LEARNING_BLOCKER Table 9 (New table for dvr):
table=9, priority=1,dl_src=dvr-cn2-mac actions=output->patch_int
table=9, priority=0 actions=resubmit(,10)

LEARN_FROM_TUN Table 10 (existing table 10):
table=10, priority=1 actions=learn(table=20,hard_timeout=300,priority=1,NXM_OF_VLAN_TCI[0..11],NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC
[],load:0->NXM_OF_VLAN_TCI[],load:NXM_NX_TUN_ID[]->NXM_NX_TUN_ID[], output:NXM_OF_IN_PORT[]), output:patch_int

UCAST_TO_TUN Table 20 (existing table 20):
table=20, priority=0 dl_vlan=green1-L-vlan, dl_dst=vm2-mac, actions=strip_vlan, set_tunnel:grn-vni, output->cn2-tep-port
table=20, priority=0 dl_vlan=red1-L-vlan, dl_dst=vm7-mac actions=actions=strip_vlan, set_tunnel:red-vni, output->cn2-tep-port
table=20, priority=0 actions=resubmit(,21)

FLOOD_TO_TUN Table 21 (existing table 21):
table=21, priority=1,dl_vlan=red1-L-vlan actions=strip_vlan, set_tunnel:red-vni, output->cn2-tep-port, output->nn-tep-port
table=21, priority=1,dl_vlan=grn1-L-vlan actions=strip_vlan, set_tunnel:grn-vni, output->cn2-tep-port, output->nn-tep-port
table=21, priority=0 actions=drop

On Compute Node CN2:

Integration Bridge Rules:

Table 0: (Local switching table)

table=0, priority=2, in_port=patch-tun, dl_src=dvr-cn1-mac actions: goto table 1

table=0, priority=1, actions: output->NORMAL

Table 1: (DVR_TO_LOCALMAC table)

table=1, priority=2, dl_vlan=grn2-L-vlan, dl_dst=grn-subnet actions: strip_vlan, mod_dl_src=r1-grn-MAC,output->port-vm2

table=1, priority=2, dl_vlan=grn2-L-vlan, nw_dst=grn-subnet actions: strip_vlan, mod_dl_src=r1-grn-MAC,output->port-vm2 (this flow will contain a list of VM ports if there is more than one VM in grn-net)

table=1, priority=1 actions: drop

Integration Bridge Rules:

Table 0: (Local switching table)

table=0, priority=2, in_port=patch-tun, dl_src=dvr-cn1-mac actions: goto table 1

table=0, priority=1, actions: output->NORMAL

Table 1: (DVR_TO_LOCALMAC table)

table=1, priority=2, dl_vlan=grn2-L-vlan, nw_dst=grn-subnet actions: strip_vlan, mod_dl_src=r1-grn-MAC,output->port-vm2

table=1, priority=1 actions: drop

***Tunnel Bridge Rules*:**

DEMUX TABLE Table 0 (existing):
table=0, priority=1,in_port=patch_int actions=resubmit(,1) - egress
table=0, priority=1,in_port=<tep-port> actions=resubmit(,4) - ingress
table=0,priority=0 actions=drop

DVR PROCESS Table 1 (New table for dvr):
table=1, priority=4, dl_vlan=red2-L-vlan, dl_type=arp, ar_tpa=r1-red-ip actions: drop
table=1, priority=4, dl_vlan=grn2-L-vlan, dl_type=arp, ar_tpa=r1-grn-ip actions: drop
table=1, priority=2, dl_vlan=red2_L_vlan, dl_dst=r1-red-mac, actions: drop
table=1, priority=2 , dl_vlan=grn2_L_vlan, dl_dst=r1-grn-mac, actions: drop
table=1, priority=1, dl_vlan=red2_L_vlan, dl_src=r1-red-mac, actions: mod_dl_src=dvr-cn1-mac, resubmit(,2)
table=1, priority=1, dl_vlan=grn2_L_vlan, dl_src=r1-grn-mac, actions: mod_dl_src=dvr-cn1-mac, resubmit (,2)
table=1, priority=0, actions: goto table 2

PATCH_LV_TO_TUN Table 2 (existing as Table 1):
table=2, priority=0,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,21)
table=2, priority=0,dl_dst=00:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,20)

GRE_TUN_TO_LV Table 3 (existing as Table 2):
table=3, priority=0 actions=drop

VXLAN_TUN_TO_LV Table 4 (existing as Table 3):
table=4, priority=1,tun_id=red-vni actions=mod_vlan_vid:red2-L-vlan,resubmit(,9)
table=4, priority=1,tun_id=grn-vni actions=mod_vlan_vid:grn2-L-vlan,resubmit(,9)
table=4, priority=0 actions=drop

DVR_LEARNING_BLOCKER Table 9 (New table for dvr):
table=9, priority=1,dl_src=dvr-cn1-mac actions=output->patch_int
table=9, priority=0 actions=resubmit(,10)

LEARN_FROM_TUN Table 10 (existing table 10):
table=10, priority=1 actions=learn(table=20,hard_timeout=300,priority=1,NXM_OF_VLAN_TCI[0..11],NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[],load:0->NXM_OF_VLAN_TCI[],load:NXM_NX_TUN_ID[]->NXM_NX_TUN_ID[], output:NXM_OF_IN_PORT[]), output:patch_int

UCAST_TO_TUN Table 20 (existing table 20):
table=20, priority=0 dl_vlan=grn2-L-vlan, dl_dst=vm2-mac, actions=strip_vlan, set_tunnel:grn-vni, output->cn1-tep-port
table=20, priority=0 dl_vlan=red2-L-vlan, dl_dst=vm7-mac actions=actions=strip_vlan, set_tunnel:red-vni, output->cn1-tep-port
table=20, priority=0 actions=resubmit(,21)

FLOOD_TO_TUN Table 21 (existing table 21):
table=21, priority=1,dl_vlan=red2-L-vlan actions=strip_vlan, set_tunnel:red-vni, output->cn1-tep-port,output->nn-tep-port
table=21, priority=1,dl_vlan=grn2-L-vlan actions=strip_vlan, set_tunnel:grn-vni, output->cn1-tep-port,output->nn-tep-port
table=21, priority=0 actions=drop

All those tables and rules shown in brown are the new ones that will be additionally managed by the L2 Agent, while it runs in DVR Mode.

A brief description of the rules in BROWN are given below:

a. ARP broadcast requests generated by tenant VMs are broadcasted to every other CN in the cloud. However, if the ARP request frame's target is the default gateway IP (router subnet ip), then such frames are dropped by the local tunnel bridge from being forwarded to the cloud. Because, such ARPs need and will be serviced only by locally available DVR instances.

Tunnel bridge
DVR PROCESS Table 1 (New table for dvr):
table=1, priority=4, dl_vlan= red1-L-vlan, dl_type=arp, ar_tpa= r1-red-ip actions: drop
table=1, priority=4, dl_vlan= grn1-L-vlan, dl_type=arp, ar_tpa= r1-grn-ip actions: drop

b. All requests generated by the distributed router interface ports, be it ARP request, other broadcast (or) unicast packets, will be sent to the cloud. But, all such frames are considered "dvr routed frames" and hence such frames will carry "local unique dvr macaddress" in the source mac of the frame before being forwarded to the cloud, on the originating node itself.
This translation of local router interface mac-address to "local unique dvr macaddress" is done by the following rules in the DVR PROCESS table.

Tunnel bridge
DVR PROCESS Table 1 (New table for dvr):
table=1, priority=1, dl_vlan=red2_L_vlan, dl_src=r1-red-mac, actions: mod_dl_src=dvr-cn1-mac, resubmit(,2)
table=1, priority=1, dl_vlan=grn2_L_vlan, dl_src=r1-grn-mac, actions: mod_dl_src=dvr-cn1-mac, resubmit (,2)

c. Complementing to what was captured as point b above, in DVR routed frames received by a compute node, the integration bridge on destination node will rip off the unique DVR MAC Addresses from the source MAC field of the frame. In place of the same, the integration bridge will substitute the local dvr instance subnet interface mac address , before the frame is forwarded to the local destination VM. Per VM rules precede network-subnet wide rules. The per VM rules ensure that the packet is put forth just to the right VM directly instead of the packet being sent like a subnet-directed broadcast to all VM ports. We may be removing the network-subnet wide rules as DVR code stabilizes. This is done by new Table 1 on integration bridge. For example on CN2:

Integration Bridge Rules:
Table 0: (Local switching table)
table=0, priority=2, in_port=patch-tun, dl_src=dvr-cn1-mac actions: goto table 1
table=0, priority=1, actions: output->NORMAL

Table 1: (DVR_TO_LOCALMAC table)

table=1, priority=2, dl_vlan=grn2-L-vlan, nw_dst=grn-subnet actions: strip_vlan, mod_dl_src=r1-grn-MAC,output->port-vm2
table=1, priority=2, dl_vlan=grn2-L-vlan, nw_dst=grn-subnet actions: strip_vlan, mod_dl_src=r1-grn-MAC,output->port-vm2
table=1, priority=1 actions: drop

d. Packets destined to the local dvr subnet interface mac address are dropped by the tunnel bridge in the originating compute node, as forwarding them to the cloud, will create mac ambiguity when the packet is decoded by other compute nodes in the cloud. This is accomplished by the
following rules:

Tunnel bridge
DVR PROCESS Table 1 (New table for dvr):
table=1, priority=2, dl_vlan=red2_L_vlan, dl_dst=r1-red-mac, actions: drop
table=1, priority=2 , dl_vlan=grn2_L_vlan, dl_dst=r1-grn-mac, actions: drop

e. In order to prevent multiple unicast of routed packets destined to remote VMs to all compute nodes in the cloud, the l2 pre-population technique is used to pre-populate the FDB table in the compute node to put out the frame only to the correct single destination compute node.

f. For rules like these (network-subnet wide rule) in the integration bridge, where a long list of ports might appear in the ‘output port’ action, this document proposes the use of ‘Group Tables’ facility available from OpenVswitch(OVS) version 2.1.

Integration bridge
Table 1: (DVR_TO_LOCALMAC table)
table=1, priority=2, dl_vlan=grn2-L-vlan, nw_dst=grn-subnet actions: strip_vlan, mod_dl_src=r1-grn-MAC,output->port-vm2

4 DataModel Extension for L2 in DVR

DistributedVirtualRouterMacAddress	
PK	host
	mac_address

The DistributedVirtualRouterMacAddress table is maintained in the controller and this table is used to store/retrieve unique DVR MAC Addresses supplied to L2 agent running on hosts identified by 'host'.

The other new table that is used to hold port bindings for DVR router interfaces is the ml2_dvr_port_bindings table shown below. This is similar to the portbindings table, but this table will hold bindings only for dvr router interfaces.

ml2 dvr port bindings	
PK **PK**	**port_id** **host**
	router_id vif_type vif_details vnic_type profile cap_port_filter driver segment status

In the above table port_id will refer to 'id' field of the port table.

The original portbindings table will also hold one-binding row for a dvr router interface, but that won't hold binding information. That binding row is held there, only to ensure transparency of dvr presence to the tenants themselves. A blueprint has been filed to give an admin only CLI to view (not manage) the bindings available in ml2_dvr_port_bindings here: https://blueprints.launchpad.net/neutron/+spec/dvr-interface-binding

Also the 'status' field value of the single-binding row for the dvr router interface in original portbindings table will be an **ORed** value of 'status' field all bindings for that same dvr router interface in the ml2_dvr_port_bindings table shown above.

5. Configuration specific to L2 Agent

5.1 enable_distributed_routing

The **ovs_neutron_plugin.ini** file being used by the OVS L2 Agent, will have an additional configuration flag:

`enable_distributed_routing=False`

In order to run the OVS L2 Agent in DVR Mode, the above flag must be set to True and the OVS L2 Agent must be restarted. The default value of this flag is False. For example on the Network Node, the L2 Agent will not run in DVR Mode. That is in NN, this flag will need to be left as is by the cloud administrator.

The L2 Agent will operate normally as it is today. However, when the **`enable_distributed_routing=True`** is read by the L2 Agent in its **init(),** it will additionally run in DVR Mode. What is DVR Mode? DVR Mode just refers to the enhanced behavior of the L2 Agent, wherein it will intelligently handle Distributed Router Interface ports that are detected on the integration bridge. As part of processing distributed router interface port (presence/absence), it will be using the OVS Rules documented in this document.

5.2 dvr_base_mac

There is another flag that will need to be made available in **neutron.conf** which represents the base-mac to use for DVR Unique MAC allocation by the ML2 Plugin. That configuration is given here:

DVR Base MAC address. The first 3 octets will remain unchanged. If the

4th octet is not 00, it will also used. The others will be randomly generated.

3 octet

dvr_base_mac = fa:16:3f:00:00:00

4 octet

dvr_base_mac = fa:16:3f:4f:00:00

The above **dvr_base_mac MUST be different from the base-mac** used allocated for virtual ports. This is required in order to ensure isolation of virtual port Mac Addresses from the DVR Unique Mac Addresses themselves.

6. RPC Changes

This section describes the new RPC calls that will be introduced in the L2 area of OpenStack to enable DVR functionality.
The following new RPCs will be introduced:

`get_dvr_mac_address_by_host (hostname)`

Method hosted by: ML2 Plugin

Invoked by: L2 Agent

Purpose for L2 Agent:
L2-Agent when running in DVR mode, invokes this method during its init(), in order to obtain the unique LMAC to be used for its Compute Node.

Controller:
Allocates a single unique MAC Address from the DVR _BASE_MAC configured pool and returns that value to the caller. For example, return value is a dict like {'host': 'T1-cn1', 'mac_address': 'aa:bb:cc:dd:ee:f1'}.

`get_dvr_mac_address()`

Method hosted by: ML2 Plugin

Invoked by: L2 Agent

Purpose for L2 Agent:
L2-Agent when running in DVR mode, invokes this method during its init(), in order to obtain list of dvr unique mac addresses available in the controller. The return mac_addresses will be used to program flows in the integration and the tunnel bridge.

Controller:
Returns a list of dvr mac addresses in use by all the compute nodes (including the caller) in the cloud.
The return value will be a list like [{'host': 'T1-cn1', 'mac_address': 'aa:bb:cc:dd:ee:f1'}, {'host': 'T1-cn2', 'mac_address': 'aa:bb:cc:dd:ee:f2'}]

`dvr_mac_address_update()`

Method hosted by: L2 Agent

Invoked by: Controller

Purpose for L2 Agent:
The L2 Agent will need to update the tunnel and integration bridge flows with the DVR Unique MACs available in the cloud (similar to TEPs for VXLAN/GRE).

Controller:
Invokes this method on all the L2 Agents running in DVR mode, and passes in a list of DVR MAC Addresses currently available in the DistributedVirtualRouterMacAddress table maintained in the controller. This is not a periodic call. This invocation is made by the controller when a change happens to the DistributedVirtualRouterMacAddress table due to new incoming L2 Agents (on new compute nodes) [or] due to deletion of rows in the DistributedVirtualRouterMacAddress table due to decommissioning of existing compute nodes.

So the pushed return value by the controller will be [{'host' : 'T1-cn1', 'mac_address' : 'aa:bb:cc:dd:ee:f2'}, {'host' : 'T1-cn2', 'mac_address' : 'aa:bb:cc:dd:ee:f3'}]

`get_compute_ports_on_host_by_subnet()`

Method hosted by: ML2 Plugin

Invoked by: L2 Agent

Purpose for L2 Agent:
The L2 Agent when recognizes that there is a distributed router interface port available on br-int to be plumbed to the cloud, it will invoke this method to figure out the list of compute ports in the local host that will need to be used in the OVS rules to plumb the distributed router interface to the cloud. The L2 Agent will pass in the hostname and the subnet_id.

Controller:
The controller returns a list of compute ports available on the input subnet and hostname.
So the pushed return value by the controller will be a list of ports [{'port_id' : 'a123578a-b12345ad-123afs87', ..}, {'port_id' : 'a123568-c645783930-s34123', …}]

`get_device_details (device, agent_id)`

Method hosted by: ML2 Plugin

Enhancement: This method return value is a port dict. The return port dict will be enhanced to contain the following additional keys : fixed_ips and device_owner , with their respective values.

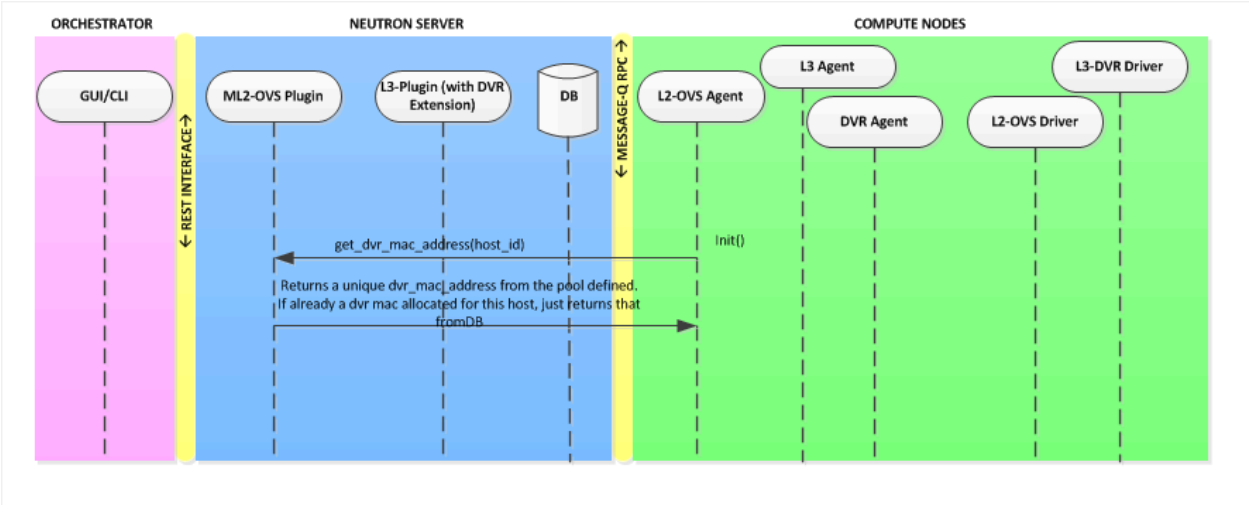
Purpose for L2 Agent:
L2-Agent when recognizes a new vif port to plumb to the cloud, invokes the above method during its treat_devices_aded(). When L2 agent is running in DVR mode the additional fields fixed_ips and device_owner are used by L2 Agent to identify what type of port it is being plumbed. For example, whether the port is a distributed router port (or) a normal compute port etc.

Controller:
In the return value for get_device_details(), the controller will additionally ship fixed_ips and device_owner values.

7. L2 Interaction with Plugin and L3 Agent

7.1 L2 OVS Agent initialization

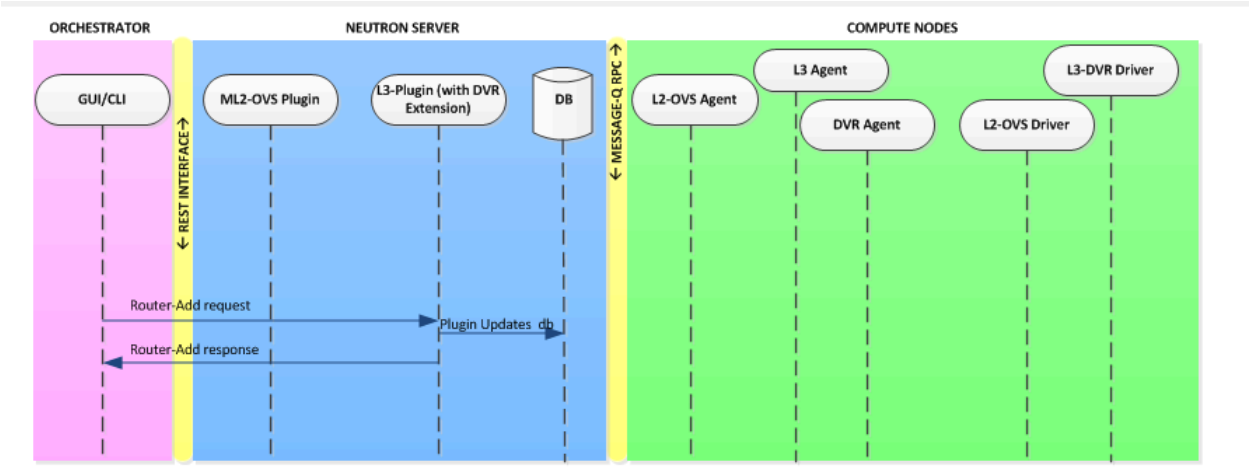
During initialization the L2 OVS Agent needs to know its hosted unique dvr mac address, in order to key in the appropriate OVS rules into the tunnel and integration bridges. For this purpose, the L2 agent invokes RPC get_dvr_mac_address(host_id) served by ML2 plugin.



7.2 Distributed Router creation

A router can be created to explicitly as a distributed router. There is typically no action required on the agents due to creation of router (be in distributed mode or otherwise). Only when an interface is added to the router, there are actions taken in both the L2 and L3 agents.

On the plugin side, as usual the created router information is stored in the DB which is shown in the figure below.



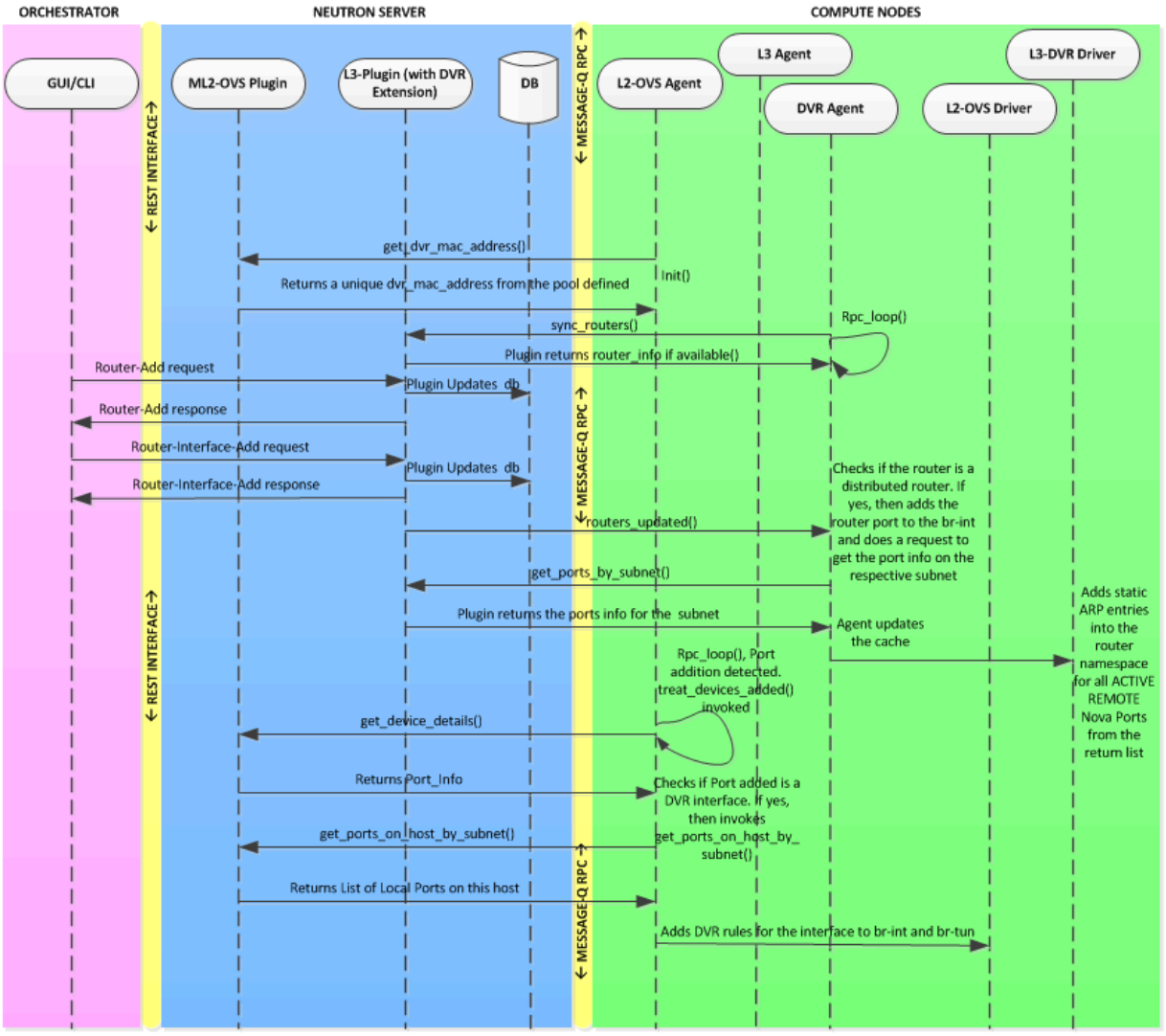
7.3 Interface addition to a distributed router

A router-interface-add command executed on a distributed router results in routers_updated() RPC to be invoked on the L3-Agent-on-CN. As part of servicing such a request, the L3-Agent-on-CN initially validates if the router affected is a distributed router. If so, it then gets the interface port corresponding

to that newly added interface and attaches that port on the integration bridge. This part of operation is similar in nature to that of the L3 Agent except for the difference being that the L3-Agent-on-CN runs in the Compute Node and adds router ports only if such ports are destined on a distributed router. *Ports that are interfaces of a distributed router will have a special* ***device_owner*** *field value as* ***'network:router_interface_distributed'.***

After adding the router interface port on the br-int, the L3-Agent-on-CN requests information about the list of ports on the cloud available on this subnet interface. For this it invokes get_compute_ports_ by_subnet (subnet_id) towards the L3 Plugin. The L3 Plugin then contacts the ML2 Plugin to get all the ports available on the input subnet and returns the list of ports to the L3-Agent-on-CN. The L3-Agent-on-CN caches these ports and then uses the port information to create static ARP entries in the DVR router namespace. This completes the DVR-side handling of the router-interface.

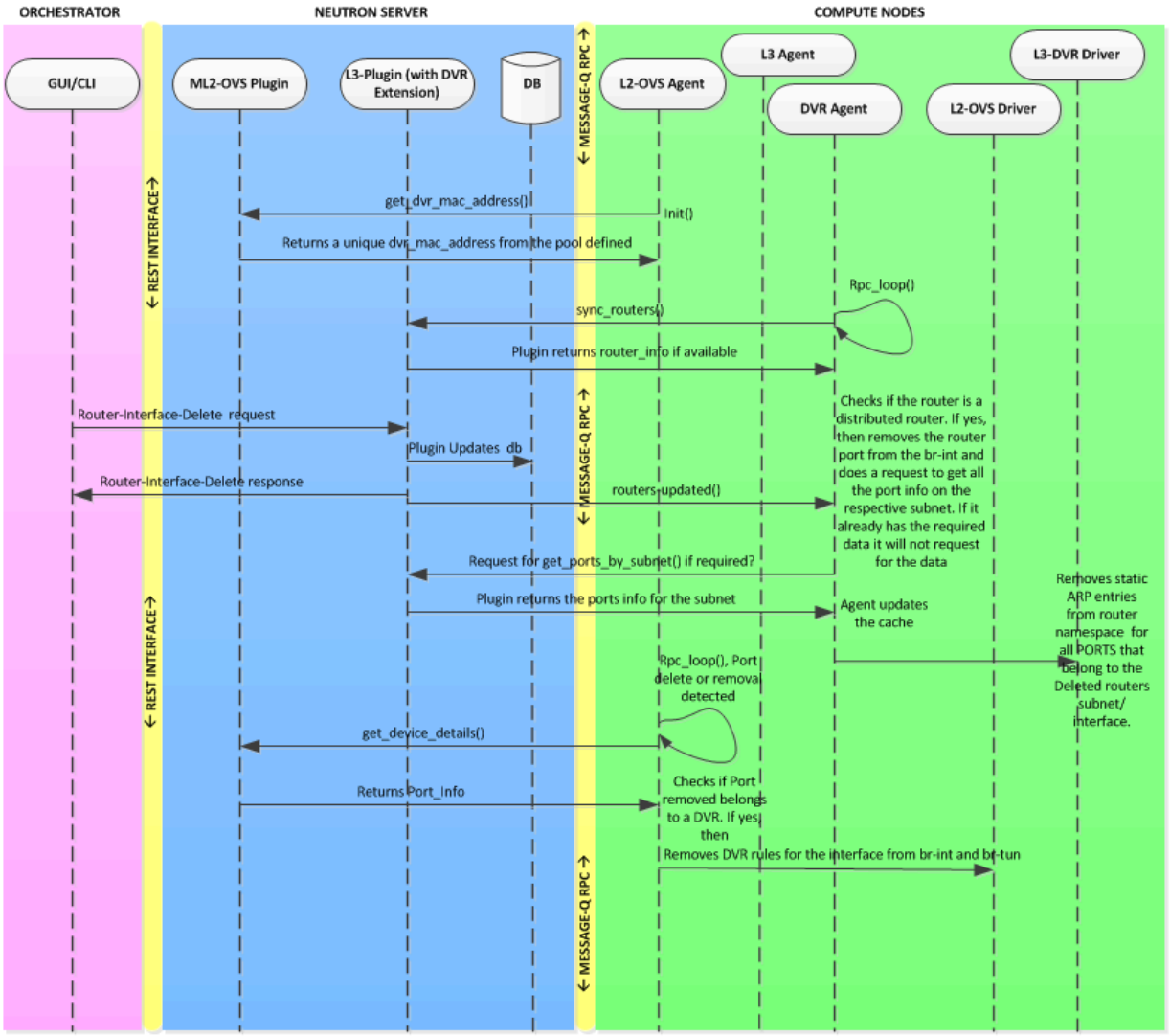
The router interface port added by the L3-Agent-on-CN is detected by the L2 Agent. The L2 Agent identifies if this port is a distributed router interface. If not, it does normal processing. If yes, it does special processing wherein it invokes get_compute_ports_on_host_by_subnet(subnet_id) to get the list of local VMs available on this router interface. It then uses this list of ports and the router interface port information to create OVS rules in the tunnel and the integration bridges.



7.4 Interface removal from a distributed router

A router-interface-delete command executed on a distributed router results in routers_updated() RPC to be invoked on the L3-Agent-on-CN. As part of servicing such a request, the L3-Agent-on-CN initially validates if the router affected is a distributed router. If so, it then gets the interface port corresponding to that being deleted. It removes that router interface port on the integration bridge. This part of operation is similar in nature to that of the L3 Agent except for the difference being that the L3-Agent-on-CN runs in the Compute Node and deletes router ports only if such ports belong to a distributed router.

After deleting the router interface port on the br-int, the L3-Agent-on-CN hits the port cache to ascertain the list of ports available on the deleted router interface. It then removes the static ARP entries from the router namespace for all the ports in that list. This completes the DVR-side handling of the router-interface deletion.



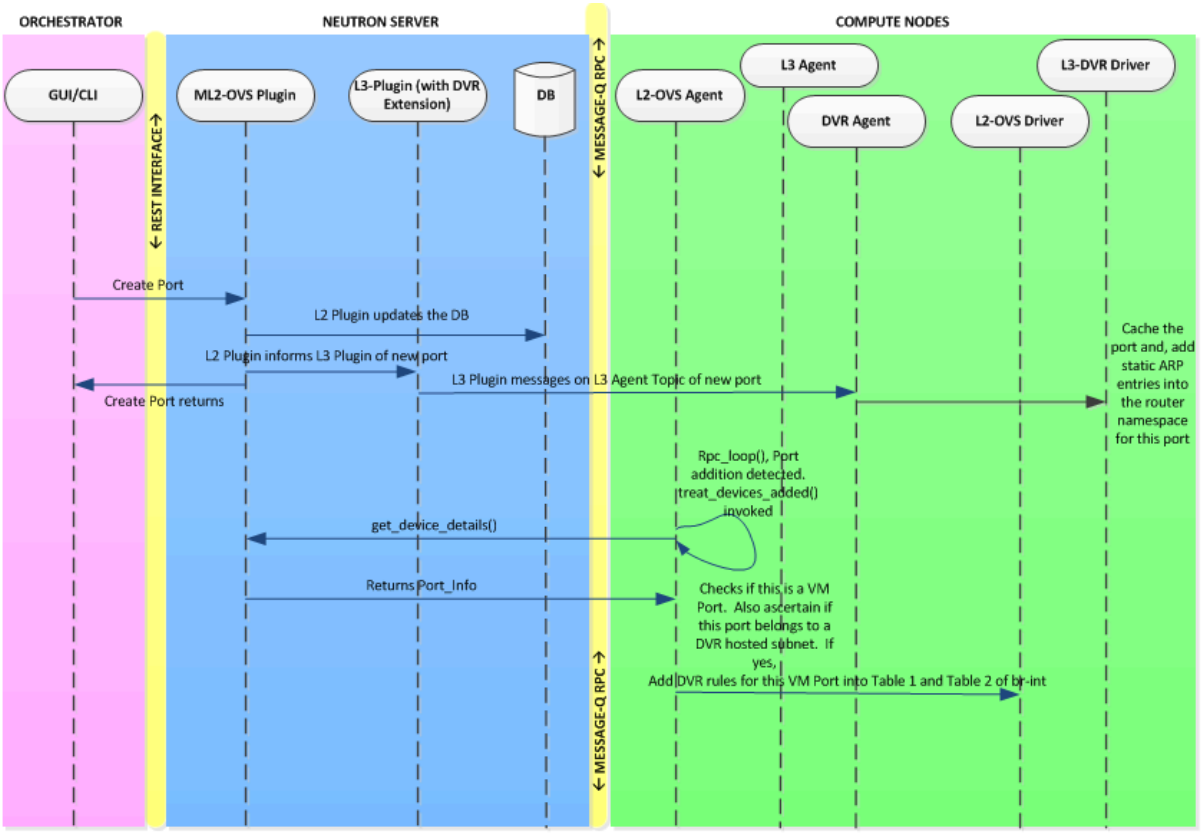
The router interface port deleted by the L3-Agent-on-CN is detected by the L2 Agent. The L2 Agent identifies if this port is a distributed router interface. If not, it does normal processing. If yes, it does special processing wherein it removes all the OVS rules from the integration and the tunnel bridges, matching the removed router interface port.

7.5 New VM added to a subnet interface of a distributed router

When a new tenant VM is added to a subnet interface managed by the distributed router, a CreatePort API invocation is done by Nova to host the new tenant VM. As part of createPort servicing by the ML2 Plugin, the ML2 plugin will inform the L3 Plugin about a new port being

available. The L3 Plugin will check if this new port is in a DVR hosted subnet. If not it will not do any processing. If yes, the L3 Plugin will initiate an RPC call port_add() to the L3-Agent-on-CN. The L3-Agent-on-CN will service this RPC in which it will get this port information and add static ARP entry for that port in the corresponding router namespace. This completes the DVR-side handling of the new tenant VM port addition.

The new tenant VM port is detected by the L2 Agent. The L2 Agent identifies if this port is a member of subnet that is already distributed router interface. If not, it does normal processing. If yes, it does special processing wherein it adds this port (OFPORT) to existing OVS Rules in br-int and br-tun for the matching subnet gateway.



7.6 Existing VM is removed from a distributed router subnet interface

When a tenant VM is removed from a subnet interface managed by the distributed router, a DeletePort API invocation is done by Nova for the deleted tenant VM. As part of deletePort servicing by the ML2 Plugin, the ML2 plugin will inform the L3 Plugin about a port being deleted.

The L3 Plugin will check if this deleted port is in a DVR hosted subnet. If not it will not do any processing. If yes, the L3 Plugin will initiate an RPC call port_delete() to the L3-Agent-on-CN. The L3-Agent-on-CN will service this RPC in which it will get this port information and remove matching static ARP entry for that port in the corresponding router namespace. This completes the DVR-side handling of the tenant VM port removal.

The removed port is detected by the L2 Agent. The L2 Agent identifies if this port is a member of subnet that is already distributed router interface. If not, it does normal processing. If yes, it does special processing wherein it modifies the existing OVS rules in br-int and br-tun so that the rule does not contain this port being deleted (OFPORT).