Proof-of-Concept (POC) for a Networking Agent which handles Overlay Networking and implements Overlay VXLAN to Underlay IP Gateway for OpenStack

by

Piyush Raman
(Roll: 2011257)

External Supervisors:

Mr. Rachappa B Goni
Advisory Engineer, IBM

Mr. Prashanth K Nageshappa
Senior Engineer / Master Inventor, IBM

Internal Supervisor:

Mr. Saket Saurav
Research Engineer, IIITDM Jabalpur

Computer Science and Engineering

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING, JABALPUR

2014

INDEX

CHAPTER 1- INTRODUCTION

This section covers an overview on the problem definition that the internship work solved followed by a literature survey on the internship work. This section is concluded by an internship plan, describing the scope of work done during 6 months Research and Development (R&D) Project Based Internship (PBI) at IBM India Cloud Networking Labs, Bangalore from May 6, 2014 to Nov 5, 2014

SECTION 1- INTRODUCTION

The PBI work at IBM India Cloud Networking Labs, Bangalore revolved around Network Virtualization and Software Defined Networking (SDN) for a virtual data center deployed using OpenStack. The past decade has seen a major shift in the services delivery mechanism from Traditional on-site resources hosting model to off-site cloud-based resource management by a third-party vendor. The internship work predominantly focused on the Infrastructure-as-a-Service delivery model of Cloud Services using OpenStack.

OpenStack is one of the largest open-source project supported by more than 250 organizations including IBM, HP, Intel, CISCO, NetApp, Rackspace, SUSE, AT&T, VMware. OpenStack is a cloud operating system managing resources including compute, network and storage delivered via cloud delivery model. We focused primarily on the Network-as-a-Service (NaaS) component of OpenStack – Neutron. The release cycle of OpenStack project is 6 months. The PBI work focuses on providing a new direction towards solving the problems regarding scalability of Neutron for a data centric deployment by leveraging use of OpenFlow protocol and Open vSwitch. As mentioned in - http://searchsdn.techtarget.com/news/2240220771/openstack-gains-steam-but-Neutron-networking-is-fraught-with-problems

"It's a great concept, but the design doesn't scale," he said. Neutron doesn't have its own router. "It uses a Linux kernel and Linux routing, but that's not good enough."

The PBI aimed particularly on solving this problem and thus providing a new methodology for virtualizing routers by leveraging use of OpenFlow and OVS . The existing OpenStack architecture virtualizes a Neutron routing instance using Host machine's TCP/IP stack (enabling IP forwarding), thus adding the load of handling overlay network traffic on the host machine and making the network node as a single point of failure for overlay and underlay

traffic. This was one of the major problem that we solved as a part of internship work. The internship work thus focused on following main features-

- Virtualize Neutron Routing instance independent of Host TCP / IP Stack
- Virtualize Neutron Routing instance using Software Defined Networking (SDN) using OpenFlow protocol and Open vSwitch (OVS)
- Implement a Distributed Virtual Router (DVR) on each node to avoid network node from becoming a single point of failure
- Implement an overlay to underlay IP gateway (for NATing) using OpenFlow protocol and Open vSwitch (OVS)

SECTION 2 - LITERATURE REVIEW

SUB-SECTION 1 - CLOUD COMPUTNG AND VIRTUALIZATION

Before getting into depth of cloud, its better to understand Virtualization and the relation between Virtualization and Cloud computing. Most of the people new to cloud might call cloud as being a technology but rather its a concept of delivering things “as-a-service”. Virtualization is one of the technologies used to realize the concept of cloud.

Virtualization is one of the fundamental technologies that makes cloud computing feasible. However, virtualization is NOT cloud computing. Cloud providers have large data centers full of servers to power their cloud offerings, but they can’t devote a single server to each customer since this leads to under-utilization of the physical resources.

Virtualization aims to solve this problem by essentially allowing sharing of physical resources amongst multiple clients with each resource being completely isolated from one-another. Thus, they virtually partition the servers, enabling each client to work with a separate “virtual” instance of the same software. Cloud computing, is an umbrella term that encompasses virtualization. It gives clients access to complex applications and massive computing resources via the Internet

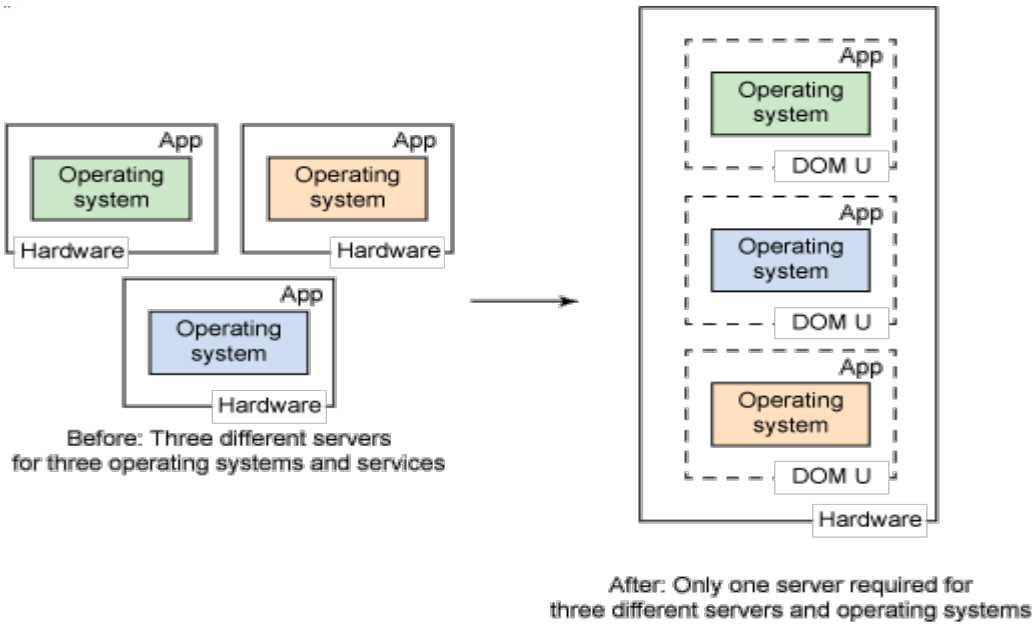


FIG 1.1- VIRTUALIZATION

So virtualization is technology that separates physical infrastructures to create various dedicated resources. Virtualization, in its most common form, makes it possible to run multiple operating systems and multiple applications on the same server at the same time. Essentially, virtualization differs from cloud computing because virtualization is software that manipulates hardware, while cloud computing refers to a service that results from that manipulation. Virtualization is a part of realization of cloud computing. Cloud treats computing as a utility (as-a-Service) rather than a specific product or technology. Cloud computing evolved from the concept of utility computing and can be thought of as many different computers pretending to be one computing environment. Cloud computing is the delivery of compute and storage resources as a service to end-users over a network. Virtualization itself, allows companies to fully maximize the computing resources at its disposal. Another form of virtualization can be Network Virtualization, wherein we virtualize network creating entire networking topology defined in software rather than hardware. This would include logical routers and switches instead of physical one's. Tunneling is also a part of network virtualization, wherein we can extend a network over different geographic locations using the Internet.

Virtualization and cloud computing work together to provide different types of services. Virtualization can be applied very broadly to just about everything you can imagine including

memory, networks, storage, hardware, operating systems, and applications.

SUB-SECTION 2 - OPENSTACK

The PBI focused on a data-centric environment and Infrastructure-as-a-Service (IaaS) service model of cloud services. As mentioned earlier, we focused on IaaS using OpenStack. OpenStack is a cloud operating system.

OpenStack manages the resources of the physical machine along with adding the element of virtualization over it. Virualization in a simple sense can be summarized as a technology that allows us to share a single physical resource as multiple isolated logical resources. OpenStack is not just a single software. Its a collection of softwares each handling a different problem and orchestrating together as a single unit. OpenStack itself is an umbrella organization for many sub-projects, each sub-project handling various resources. OpenStack's main components include-

- Nova (compute)
- Neutron (Networking)
- Keystone (Authorization)
- Glance (Image Management)
- Cinder (Volume / Block storage)
- Swift (Object Storage)
- Horizon (Dashboard)

OpenStack Nova (CPU Virtualiztaion)

Nova handles the CPU / compute virtualization. By CPU, we mean components of CPU such as RAM, Cores, Architecture, OS. So nova handles is the component that creates virtual CPUs for each virtual machine according to the configuration of the user. It also helps us implement security rules / firewall using iptables rules on each virtual instance. Security rules here would be things like- allowing ssh / ping / tcp / udp packets in/out from the virtual machine. Apart, Nova also provisions for vnc-viewer for access to the virtual machine. So each CPU launched as a virtual machine is handled by the Nova

OpenStack Neutron (Network Virtualization)

This is the networking component of OpenStack. It handles the entire networking aspect of OpenStack. Networking here would include-

- Creating networks, subnets, routers dynamically

- Connecting the virtual network of VMs to the external network (Internet)
- Provide routing facility amongst the VM's using virtual routers
- Handle SNAT/DNAT facility
- Managing network connectivity / isolation amongst the virtual machines

OpenStack Keystone (Authorization)

Each action taken by any client is treated as a request to OpenStack via some GUI / CLI. Now this request has the ID of the client. Now this ID is used to differentiate amongst the clients and their access rights. Thus keystone handles all these clients and what actions / operations are allowed to them including resource usage

OpenStack Horizon (GUI)

Horizon is the OpenStack GUI module. It lets you user use OpenStack in a much friendly manner against the usage of CLI on the terminal.

This was broadly the function of each module, however, all these module work together to achieve a common task. No module works in isolation and is self-sufficient. Simple example is launching a VM from dashboard- This involves usage of all the above service-

- Horizon for GUI for launching a VM
- Keystone for validating the request from tenant / client
- Nova for launching a virtual instance
- Neutron for allocating it to a network

-Glance for providing the image for OS / snapshot

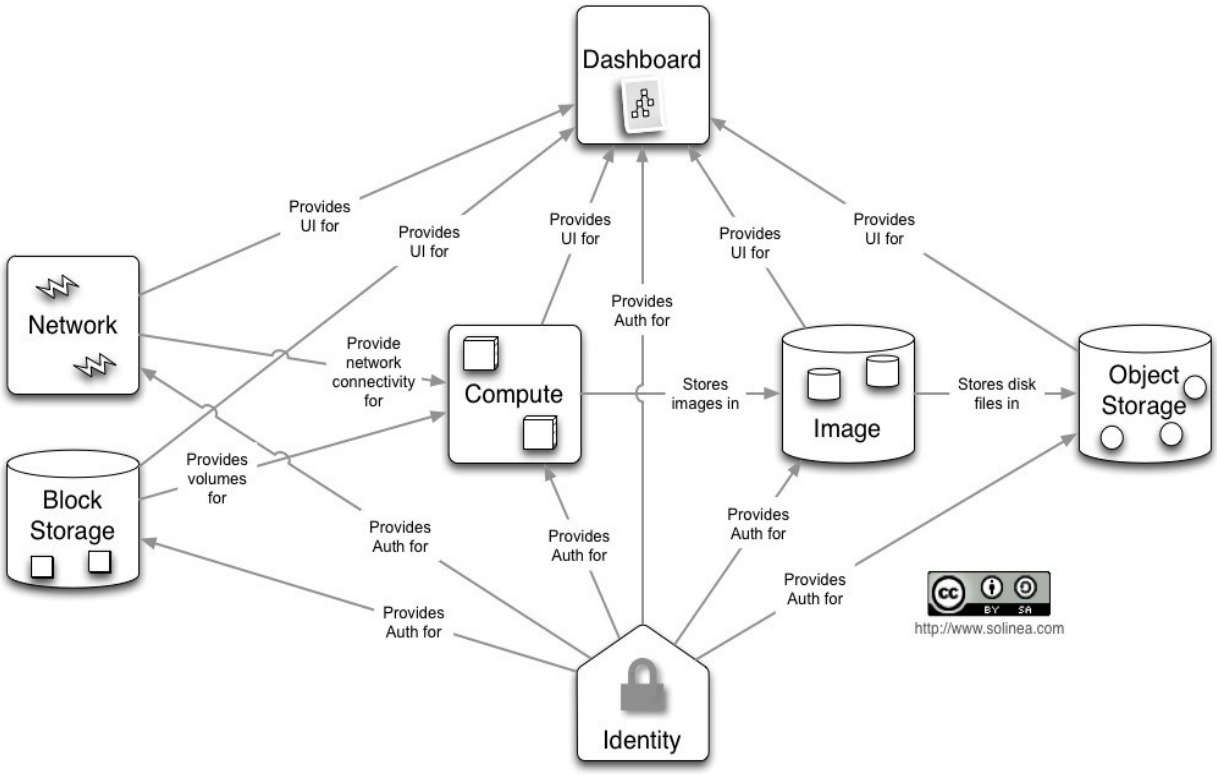


FIG 1.2- VARIOUS COMPONENTS OF OPENSTACK AND THEIR INTERACTION

Our work focused on OpenStack Neutron, the networking component of OpenStack handling Network-as-a-Service for virtual interfaces (vNICs) on each Virtual Machine (VM) managed by

OpenStack Nova. All nodes are connected to 3 networks generally in a data-centric environment-

- **API Network** connecting the end-user
- **Data Network** for transmission of data amongst nodes (e.g. for tunneling)
- **Management network**

Any data center deployed using OpenStack includes pre-dominantly 3 types of Node (Physical machines)-

- **Compute Node**

Nodes on which the VM's are hosted. The VM's use HDD / RAM / VCPUs from the compute nodes on which they are hosted.

- **Network Node**

This node handles the overlay to underlay network traffic. Prior to OpenStack Juno Release in October 2014, this node was handling overlay across subnet and overlay top underlay network traffic including Network Address Translations (NATing) and thus acted as a single point of failure for overlay across subnet and overlay to underlay networking. Note that the external network connectivity is localized on Network Nodes only

- Controller Node

All the servers of various OpenStack Modules (e.g. nova, neutron, glance, etc.) are hosted on this node. Thus the servers on the controller node handles communication with each agent local to the compute node.

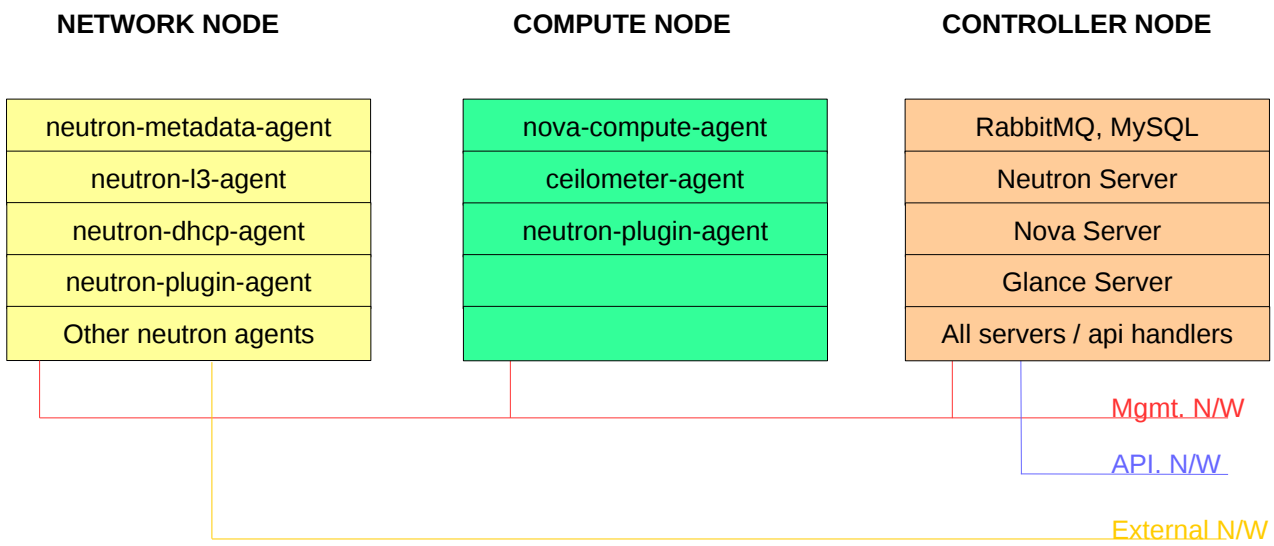


FIG 1.3- OPENSTACK NODES AND DATA-CENTRIC ENVIRONMENT

OpenStack Neutron Architecture

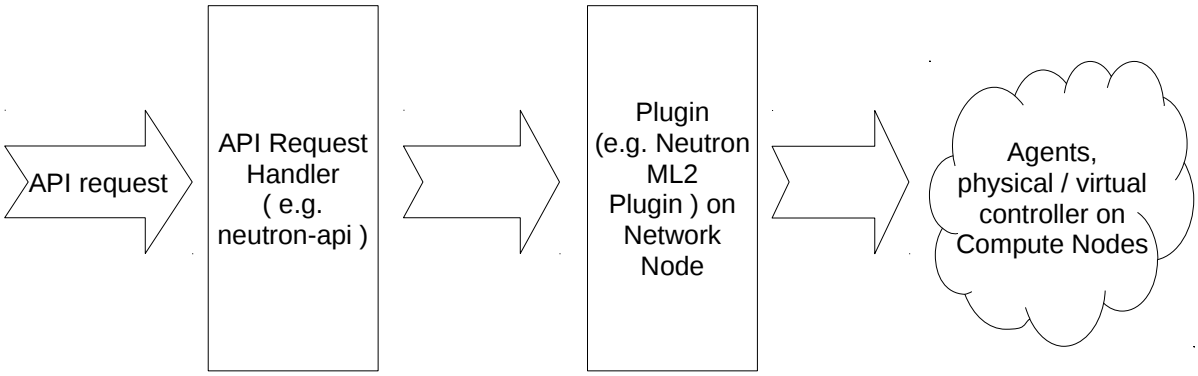


FIG 1.4- OPENSTACK NEUTRON PLUGGABLE ARCHITECTURE

OpenStack Neutron has a pluggable architecture. This means that third-party vendor can develop their own OpenStack Neutron Agents for handling networking in OpenStack. As shown in the diagram, the controller node has a neutron-server running (which includes the plugin which receives requests). All the queries are thrown at the core / service plugins, which further communicate with each instance of agents running on each compute node. Note, that none of the agents local to each compute node interact directly with each other, but via the plugin on controller node. As shown earlier, various modules on OpenStack interact with each other to achieve a task. Various components (nova, neutron, glance, cinder, etc) use REST API Calls to interact with each other. However, within each component (e.g. Neutron server on controller node and neutron agents on compute node for OpenStack Neutron) use RPC mechanism to interact and synchronize with each other and not REST API Calls. Following are some major components of OpenStack Neutron and nodes where each of them is hosted

- **Core / Service Neutron Plugin** (part of Neutron server)

  The neutron plugin forms a part of the OpenStack Server and receives all the requests from users / other modules. The plugin also interacts with the agents on each compute node and synchronize amongst various agents on different nodes. A single instance of core and service runs on each network node

- **Neutron-openvswitch-agent**

  This is neutron's L2 agent localized on each compute node. This agent handles adding flows on OVS tunnel bridge for internal VLAN ID to VXLAN ID translations and vice versa for VXLAN tunneling and transmission across hosts. It also handles VLAN tagging of ports on OVS integration Bridge

- **Neutron-l3-agent**

  This is the neutron's L3 agent handling overlay to external underlay network including NATing for the same. A single instance of neutro-l3-agent handles one external network (until OpenStack HAVANA release). Each instance of neutron-l3-agent resides on network node only.

SUB-SECTION 3 - SDN

Software Defined Networking (SDN) is a new approach to designing, building and managing networks. The basic concept is that SDN separates the network's control (brains) and forwarding (muscle) planes to make it easier to optimize each. Centralized and decentralized control does not matter in SDN.

Data plane is part of the switch / router that actually forwards the packet based on forwarding / routing tables and control plane is that part of switch/router that populates these tables. So data plane deals with actual forwarding of packet and control plane deals with how the packet is to be forwarded.

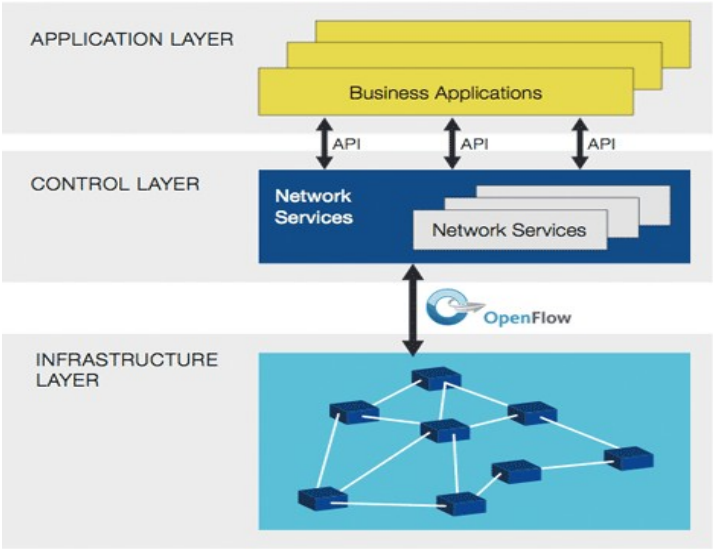


FIG 1.5- SDN ARCHITECTURE

In an SDN environment, a Controller acts as the "brains". A controller is typically a program that run on some remote host and is not local to the switch/router. This is what distinguishes SDN. The control plane is not defined locally to router but rather on a remote machine. Through the Controller, network administrators can quickly and easily make and push out decisions on how the underlying systems (switches, routers) of the forwarding plane will handle the traffic. The most common protocol used in SDN networks to facilitate the communication between the Controller and the switches is currently OpenFlow. However, OpenFlow is not SDN. OpenFlow is just one of the protocols used to realize the entire SDN architecture.

SDN enables a network administrator to shape traffic and deploy services to address

changing business needs, without having to touch each individual switch or router in the forwarding plane. Thus any mechanism that helps to manage traffic according to the varying workloads on-the-go is called SDN. (Note that here on-the-go would represent a time period of few hours instead of traditional period of multiple days)

The Benefits of SDN

Following are some advantages of SDN model-

- Reducing the need to purchase costly networking hardware having specialized and coupled control and data plane
- More fine grained and more centralized control over network traffic without the need to access the physical hardware
- Helping organizations rapidly deploy new applications, services and infrastructure to quickly meet their changing business goals and objectives.

SUB-SECTION 4 – NETWORK VIRTUALIZATION

Network Virtualization creates logical / virtual networks, subnets, routers, switches, etc. that are decoupled from the underlying network hardware. Virtualization creates a logical software-based view of the hardware and software networking resources (switches, routers, etc.). The physical networking devices are simply responsible for the forwarding of packets, while the virtual network (software) provides an intelligent abstraction that makes it easy to deploy and manage network services and underlying network resources. Thus with Network Virtualization, the goal is to take all of the network services, features, and configuration necessary to provision the application's virtual network (VLANs, VRFs, Firewall rules, Load Balancer pools, isolation, multi-tenancy, etc.) – take all of those features, decouple it from the physical network, and move it into a virtualization software layer. With the virtual network fully decoupled, the physical network configuration is simplified to provide packet forwarding service from one hypervisor to the next.

Network Virtualization reproduces the L2-L7 network services necessary to deploy the application's virtual network at the same software virtualization layer hosting the application's virtual machines – the hypervisor kernel and its programmable virtual switch. Similar to how server virtualization reproduces vCPU, vRAM, and vNIC – Network Virtualization software reproduces Logical switches, Logical routers (L2-L3), Logical Load Balancers, Logical

Firewalls (L4-L7), and more, assembled in any arbitrary topology, thereby presenting the virtual compute a complete L2-L7 virtual network topology.

SUB-SECTION 5 – NETWORK VIRTUALIZATION VS. SDN

There has been long arguments that virtualization already exists in network technologies (e.g. VXLAN, VLAN's and network overlays). However, just saying that VXLAN Tunneling is Network Virtualization is wrong. These technologies virtualize transport of the network they don't actually change the operation model of networking. Network virtualization lets you abstract the operations of your network from the physical access layer. Configurations can be RECORDED, COPIED, PAUSED AND REWINDED. They way you provision and manage your network is completely changed by network virtualization.

SDN just deals with the decoupling of control plane from the physical plane. It is not defined with respect to being explicit to virtual network or physical network. So SDN is a networking architecture in which the data and control planes are decoupled be it for physical network or virtual networks (created via Network Virtualization). When we say that the data and control plane are decoupled, we actually mean that the forwarding database / routing decisions are not local to the switch / router itself, but, rather taken by a remote SDN Controller via a communication protocol such as OpenFlow. In essence, for network control and administration, we are not needed to physically access each and every router / switch for manipulation in control plane.

SDN is more about having a programmatic way to define the control plane. So we can have a SDN run inside of a virtual network controlling that control plane of the virtualized network. For e.g. OpenStack – lets say you deploy instances of OpenStack on multiple virtual VM's on one physical host. Now you link all the VM's via VXLAN Tunneling. Now OpenStack Neutron uses SDN for managing the tunnel bridge. So the entire decision for flow on packets to-and-from tunnel end points on the tunnel bridge (i.e. the control plane) and hence the overlay is administered using a remote agent (openvswitch-agent in Neutron) instead of decision being solely taken locally by the tunnel bridge itself and hence the SDN architecture on the tunnel.

Thus Network Virtualization relates to virtualizing L2-L7 services of a network i.e. defining them in software rather than physical underlying hardware with the ability to copy, pause, take snapshot of the entire topology. SDN however relates to particular decoupling of data and control plane in router / switches, be it for physical or virtual networks created using

Network Virtualization. A virtual router created as apart of Network Virtualization may still have a coupled control and data plane i.e. the routing decisions are being taken locally on that router. However, SDN architecture de-localizes this decision making to a remote controller that does the routing decisions.

SECTION 3 - PLAN OF INTERNSHIP

This section describes the details of various stages during R&D internship at IBM India Cloud Networking Labs, Bangalore. As mentioned the PBI was based on Research and Development profile. The entire 6 months duration work can be broken down into following groups-

- May 2014

The first month of internship focused on understanding basics of Cloud Computing, various service models- Saas, PaaS and IaaS and various deployment models- Public, Private and Hybrid Clouds. This duration also included developing a basic understanding of Computer Networking, specifically Layer 2 and Layer 3 of TCP/IP Model. The learning topic then switched to basics of virtualization, network virtualization, software defined networking, OpenFlow protocol and Open vSwitch. This duration also included hands-on experience using libvirt (via virt-manager) for virtualization and understanding various networking use cases (isolated, nated, routed) for VM's launched using libvirt. It also included hands on experience with Linux Bridge and understanding the pot mappings used by libvirt on Linux Bridge.

- June 2014

The second month of internship focused on understanding the existing OpenStack Architecture, specifically the networking component- Neutron in detail. Cloud computing using OpenStack and deployment of OpenStack Havana and Icehouse including RHEL Administration was also covered during this internship. Multiple research assignments were covered to cover various networking use cases. VXLAN tunneling for transmission of packet across hosts was also covered during this period. Following were the use-cases studied for OpenStack Neutron-

For single / multiple hosts (using VXLAN Tunneling):

- Overlay within subnet
- Overlay across subnet
- Source NAT (SNAT)

- Destination NAT (DNAT) / floatingIP

- June 2014-August 2014

After understanding the existing setup and the problem definition with reference to the existing setup, this duration focused on Research around developing a solution using OpenFlow protocol on Open vSwitch Bridge for handling-

- Overlay within subnet
- Overlay across subnet
- Source NAT (SNAT)
- Destination NAT (DNAT) / floatingIP

and also implementing a Distributed Virtual Router local to each compute node. During the given period we conceptualized and developed a working POC model for achieving the above functionalities with using Host TCP/IP Stack and Linux Kernel for the same as in case of existing OpenStack setup. For conceptualizing and designing the POC, we designed multiple flows tables and OpenFlow pipeline of various OVS bridges by hard-coding the flows directly onto the bridge instead of using neutron-agents for the same.

- August – September 2014

After conceptualizing - designing the POC and successfully achieving the needed functionalities by hard-coding the flows directly onto the various bridges in OpenStack, the PBI focused on modifying the existing OpenStack Neutron Agent to add the flows programatically and handle addition of ports on OVS Bridges. Thus, during this duration, we first studied the existing Neutron source code and then made changes to the Neutron source to reflect the changes needed for POC and finally developed a stand-alone neutron agent implement the POC and adding flows programatically onto the OVS Bridges.

- September – October 2014

After implementing a working OpenStack Agent for POC, the work shifted on testing the POC for a multi-tenant environment and for multiple networks. This included testing the new agant for various use cases and rectifying any errors or shortcomings. The work was then concluding by performing an analysis on the bandwidth for various use cases on SNAT / DNAT and benchmarking the POC performance against the OpenStack's Performance.

CHAPTER 2 – OPENSTACK NEUTRON ARCHITECTURE AND MECHANISM

This section describes in detail the existing OpenStack Neutron architecture and how it handles networking in a virtual data center deployed using OpenStack. The section begins with discussion of OpenFlow protocol, Open vSwitch and VXLAN Tunneling and is followed by existing bridge setup on Compute and Network node. It then describes how Neutron handles-

- Overlay within subnet traffic
- Overlay across subnet traffic
- Overlay to underlay (SNAT)
- Overlay to underlay (DNAT)

SECTION 1 - OpenFlow PROTOCOL

OpenFlow defines the open communications protocol in SDNs that enables the Controller to interact with the forwarding plane and make adjustments to the network, so it can better adapt to changing business requirements. With OpenFlow, entries can be added and removed to the internal flow-table of switches and potentially routers to make the network more responsive to real-time traffic demands.

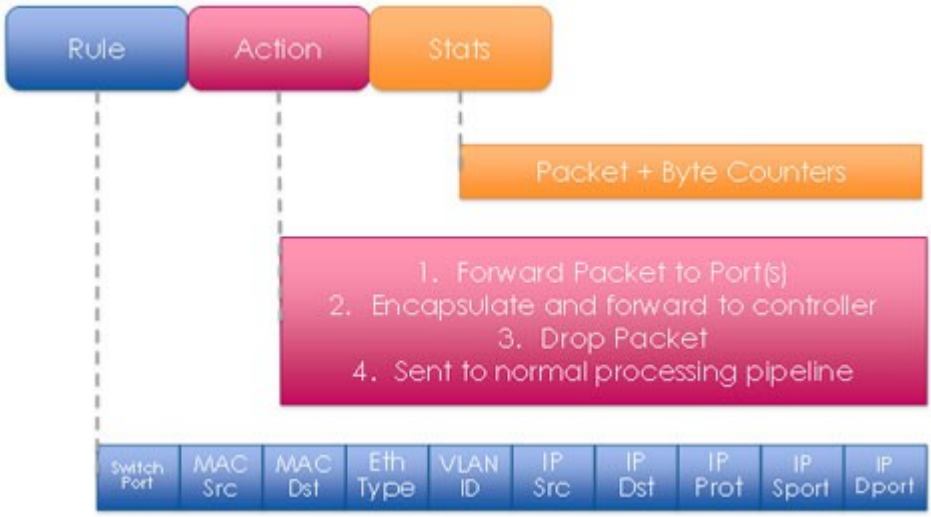


FIG 2.1 – FLOW-TABLES ENTRIES THAT CAN BE MANIPULATED USING OPENFLOW

OpenFlow allows direct access to and manipulation of the forwarding plane of network devices such as switches and routers, both physical and virtual (hypervisor-based). The protocol specifies basic primitives that can be used by an external software application to program the forwarding plane of network devices.

Any device that would like to participate in this environment must support OpenFlow with a standardized interface. This interface exposes the internal workings of the device, which enables the Controller to push down changes to the flow-table. Once the devices are OpenFlow-enabled, network administrators can use them to partition traffic, control flows for optimal performance, and start testing new configurations and applications.

Benefits of OpenFlow-based-SDN-

- Centralized control of multi-vendor environments:

SDN control software can control any OpenFlow-enabled network device from any vendor, including switches, routers, and virtual switches. Rather than having to manage groups of devices from individual vendors, IT can use SDN-based orchestration and management tools to quickly deploy, configure, and update devices across the entire network.

- Reduced complexity through automation:

OpenFlow-based SDN offers a flexible network automation and management framework, which makes it possible to develop tools that automate many management tasks that are done manually today. These automation tools will reduce operational overhead, decrease network instability introduced by operator error, and support emerging IT-as-a-Service and self-service provisioning models. In addition, with SDN, cloud-based applications can be managed through intelligent orchestration and provisioning systems, further reducing operational overhead while increasing business agility.

- More granular network control:

OpenFlow‘s flow-based control model allows IT to apply policies at a very granular level, including the session, user, device, and application levels, in a highly abstracted, automated fashion. This control enables cloud operators to support multi-tenancy while maintaining traffic isolation, security, and elastic resource management when customers share the same infrastructure.

SECTION 2 - OPEN VSWITCH

Open vSwitch (OVS), is an implementation of a virtual multilayer switch. Open vSwitch is a virtual switch for hypervisors providing network connectivity to VM's. Primarily used as a virtual switch for connecting VM's. Open vSwitch provides a switching stack for hardware virtualization environments, while supporting multiple protocols and standards used in

computer networks. It support Open Flow Protocol. It implements a production quality switch platform that supports standard management interfaces and opens the forwarding functions to programmatic extension and control. In addition to exposing standard control and visibility interfaces to the virtual networking layer, it was designed to support distribution across multiple physical servers.

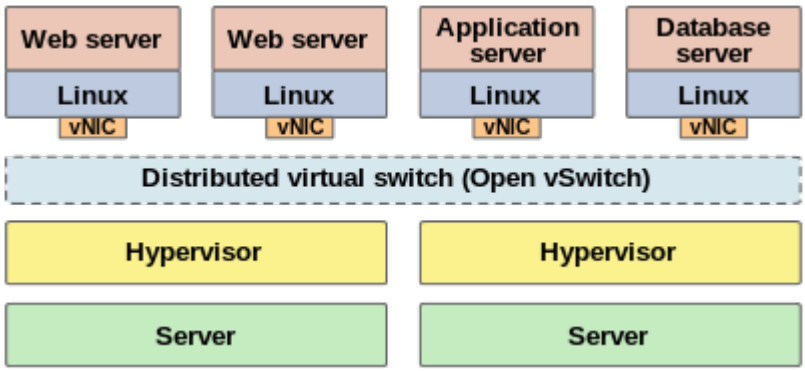


FIG 2.2- OPEN VSWITCH

An Open vSwitch bridge can operate in either “normal” mode or “flow” mode. In normal mode it acts as a regular layer 2 learning switch. For each incoming frame it learns its source MAC address and places it on its incoming port. Thus, each VM mapped on OVS bridge actually has a unique / different MAC address which is not the actual physical MAC address. It then either forwards the frame to the appropriate port if the destination MAC address was previously learned, or floods the frame if it wasn’t. Broadcast and multicast frames are flooded as usual. In flow mode, the bridge’s flow table is used instead. (table filled by Open Flow) In this mode it maintains a contact with the SDN Controller.

Each Open vSwitch flow is composed of a match and action part. Flow tables are composed of many flows which are processed in a well defined order. The match part of a flow defines what fields of a frame/packet/segment must match in order to hit the flow. Once a match is found, the action part of a flow defines what actually happens now that the flow was hit. We can match on most fields in the layer 2 frame, layer 3 packet or layer 4 segment. So, for example, we could match on a specific destination MAC and IP address pair, or a specific destination TCP port. Note that the match must make sense top to bottom, so we cannot specify that in the IP packet the “Next Protocol” field must be ICMP, but then in the same flow match against a TCP destination port, as TCP and ICMP are both encapsulated at layer 4 inside of an IP packet.

Matches may also be wildcarded, so you can match against a range of ports or IP addresses. Any field not explicitly defined is wildcarded against, so if a flow doesn't say anything about the source MAC address then any source MAC address matches. The action part of a flow defines what is actually done on a message that matched against the flow. We can forward the message out a specific port, drop it, change most parts of any header, build new flows on the fly (For example to implement a form of learning), or resubmit the message to another table.

Extensive flow matching capabilities-

- Layer 1 – Tunnel ID, In Port, QoS Priority, skb mark
- Layer 2 – MAC Address, VLAN ID, Ethernet Type
- Layer 3- IPv4 / IPv6 fields, ARP
- Layer 4- TCP/UDP, ICMP, ND

Possible chain of actions-

- Output to port (port range, flood, mirror)
- Discard, Resubmit to table X
- Packet Mangling (Push/Pop VLAN Header, TOS, …)
- Send to Controller, Learn

Open vSwitch supports the following features

- Fully functional Layer 2 switch
- Support for link aggregation through Link Aggregation Control Protocol (LACP, IEEE 802.1AX-2008)
- Standard 802.1Q Virtual LAN (VLAN) model with trunking
- Fine-grained quality of service (QoS) control
- Traffic policing at the level of virtual machine interface
- Network interface controller (NIC) bonding; with load balancing by source MAC addresses, active backups, and layer 4 hashing
- Support for OpenFlow protocol (including many extensions for virtualization)
- Support for IPv6
- Multiple tunneling protocols (GRE, Virtual Extensible LAN (VXLAN), Internet Protocol Security (IPsec), GRE and VXLAN over IPsec)
- Remote configuration protocol, with C and Python bindings

- Kernel space and user space forwarding engine options
- Multi-table forwarding pipeline with a flow-caching engine
- Forwarding layer abstraction, making porting to new software and hardware platforms easier

SECTION 3 - Virtual eXtensible Local Area Network (VXLAN) Tunneling

The most popular virtualization technique in the data center is VXLAN. The basic use case for VXLAN is to connect two or more layer three (L3) networks and make them look like they share the same layer two (L2) domain. It does this by encapsulating frames in VXLAN packets. This would allow for virtual machines to live in two disparate networks yet still operate as if they were attached to the same L2

To operate a VXLAN needs a couple of components in place:

- Multicast support, IGMP and PIM (optional)
- VXLAN Network Identifier (VNI)
- VXLAN Gateway
- VXLAN Tunnel End Point (VTEP)
- VXLAN Segment/VXLAN Overlay Network

VXLAN is an L2 overlay over an L3 network. Each overlay network is known as a VXLAN Segment and identified by a unique 24-bit segment ID called a VXLAN Network Identifier (VNI). Only virtual machine on the same VNI are allowed to communicate with each other. Virtual machines are identified uniquely by the combination of their MAC addresses and VNI. As such it is possible to have duplicate MAC addresses in different VXLAN Segments without issue, but not in the same VXLAN Segments

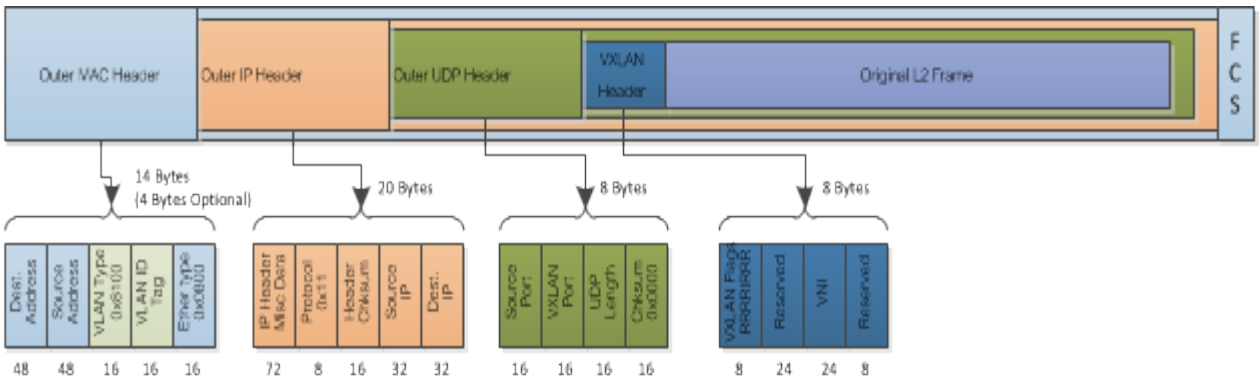


FIG 2.3 – VXLAN PACKET HEADER

The original L2 packet that the virtual machines send out is encapsulated in a VXLAN header that includes the VNI associated with the VXLAN Segments that the virtual machine belongs to. The resulting packet is then wrapped in a UDP (8 bytes) → IP (20 bytes)→ Ethernet packet (14 bytes) for final delivery on the transport network. The VTEPs are responsible for encapsulating the virtual machine traffic in a VXLAN header as well as stripping it off and presenting the destination virtual machine with the original L2 packet.

The encapsulation is comprised of the following modifications from standard UDP, IP and Ethernet frames:

- Ethernet Header:

Destination Address – This is set to the MAC address of the destination VTEP if it is local of to that of the next hop device, usually a router, when the destination VTEP is on a different L3 network.

VLAN – This is optional in a VXLAN implementation and will have an associated VLAN ID tag

Ethertype – This is set to 0x0800 as the payload packet is an IPv4 packet.

- IP Header:

Protocol – Set 0x11 to indicate that the frame contains a UDP packet

Source IP – IP address of originating VTEP

Destination IP – IP address of target VTEP. If this is not known, as in the case of a target virtual machine that the VTEP has not targeted before a discovery process needs to be done by originating VTEP. This is done in a couple of steps:

Destination IP is replaced with the IP multicast group corresponding to the VNI of the originating virtual machine

1. All VTEPs that have subscribed to the IP multicast group receive the frame and decapsulate it learning the mapping of source virtual machine MAC address and host VTEP
2. The host VTEP of the destination virtual machine will then send the virtual machines response to the originating VTEP using its destination IP address as it learned this from the original multicast frame

3. The Source VTEP adds the new mapping of VTEP to virtual machine MAC address to its tables for future packets

- UDP Header:

Source Port – Set by transmitting VTEP

VXLAN Port – IANA assigned VXLAN Port.

UDP Checksum – This should be set to 0x0000. If the checksum is not set to 0x0000 by the source VTEP, then the receiving VTEP should verify the checksum and if not correct, the frame must be dropped and not decapsulated.

- VXLAN Header:

VXLAN Flags – Reserved bits set to zero except bit 3, the I bit, which is set to 1 to for a valid VNI

VNI – 24-bit field that is the VXLAN Network Identifier

Reserved – A set of fields, 24 bits and 8 bits, that are reserved and set to zero

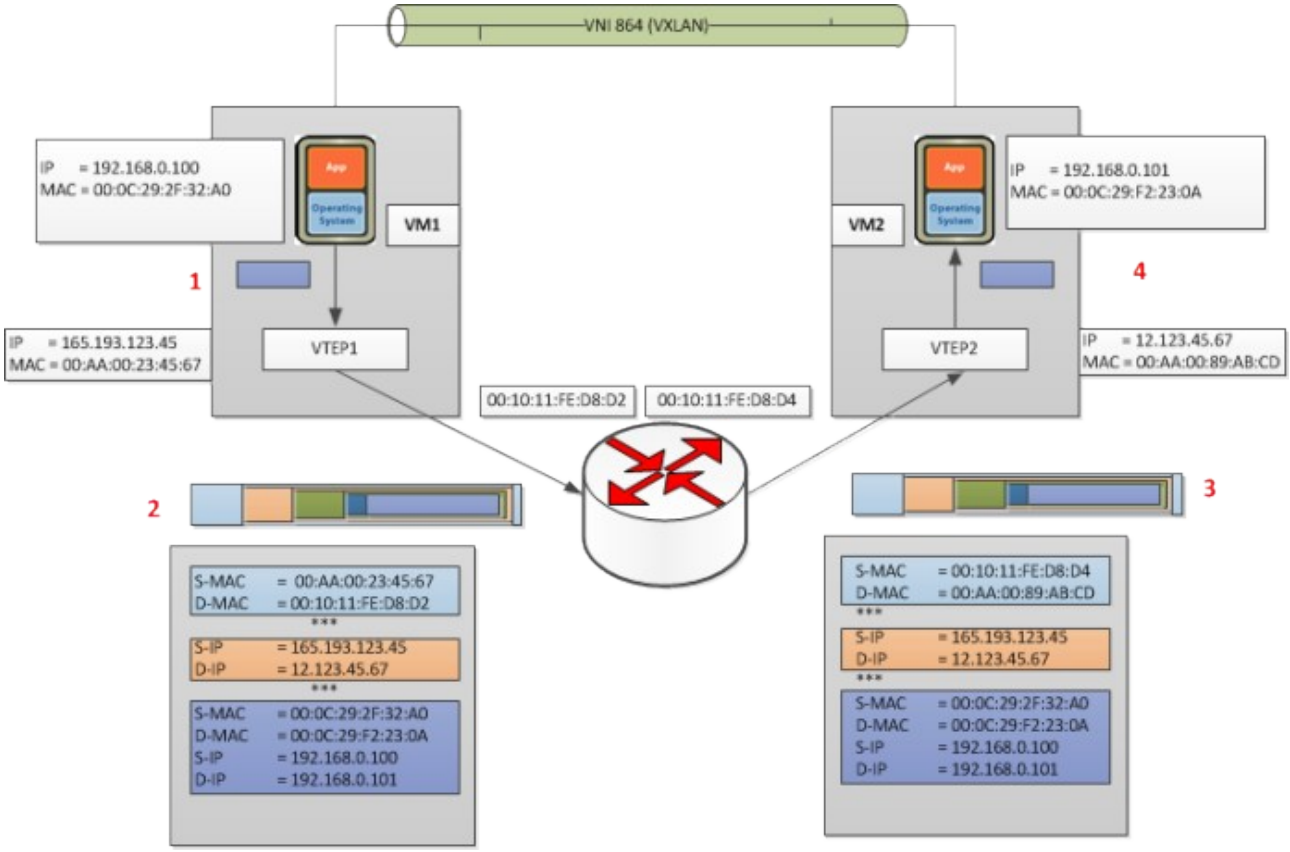


FIG 2.4 – VM TO VM COMMUNICATION

When VM1 wants to send a packet to VM2, it needs the MAC address of VM2 this is the

process that is followed:

- VM1 sends a ARP packet requesting the MAC address associated with 192.168.0.101
- This ARP is encapsulated by VTEP1 into a multicast packet to the multicast group associated with VNI 864
- All VTEPs see the multicast packet and add the association of VTEP1 and VM1 to its VXLAN tables
- VTEP2 receives the multicast packet decapsulates it, and sends the original broadcast on portgroups associated with VNI 864
- VM2 sees the ARP packet and responds with its MAC address
- VTEP2 encapsulates the response as a unicast IP packet and sends it back to VTEP1 using IP routing
- VTEP1 decapsulates the packet and passes it on to VM1

At this point VM1 knows the MAC address of VM2 and can send directed packets to it as shown in in Fig 2.4 – VM to VM Communication

VM1 sends the IP packet to VM2 from IP address 192.168.0.100 to 192.168.0.101

1. VM1 sends the IP packet to VM2 from IP address 192.168.0.100 to 192.168.0.101

2. VTEP1 takes the packet and encapsulates it by adding the following headers:
    - VXLAN header with VNI=864
    - Standard UDP header and sets the UDP checksum to 0x0000, and the destination port being the VXLAN IANA designated port. Cisco N1KV is currently using port ID 8472.
    - Standard IP header with the Destination being VTEP2's IP address and Protocol 0x011 for the UDP packet used for delivery
    - Standard MAC header with the MAC address of the next hop. In this case it is the router Interface with MAC address 00:10:11:FE:D8:D2 which will use IP routing to send it to the destination

3. VTEP2 receives the packet as it has it's MAC address as the destination. The packet is decapsulated and found to be a VXLAN packet due to the UDP destination port. At this point the VTEP will look up the associated portgroups for VNI 864 found in the VXLAN header. It will then verify that the target, VM2 in this case, is allowed to

receive frames for VNI 864 due to it's portgroup membership and pass the packet on if the verification passes.

4. VM2 receives the packet and deals with it like any other IP packet.

The return path for packet from VM2 to VM1 would follow the same IP route through the router on the way back.

SECTION 4 – OPENSTACK NEUTRON BRIDGE SETUP

The following section describes the OVS Bridge setup on Compute Node and Network Node using OpenStack

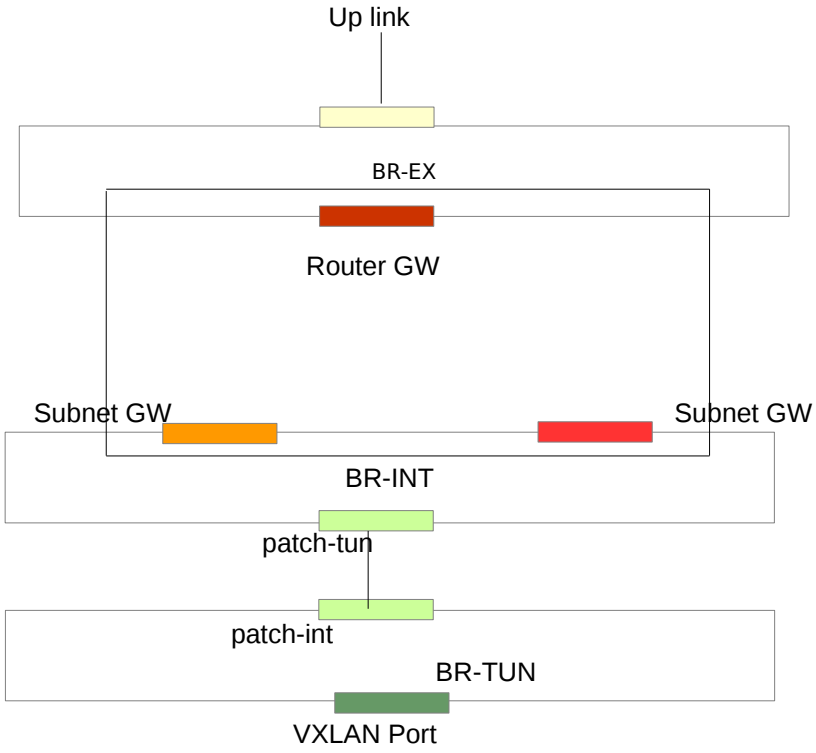


FIG 2.5- NETWORK NODE

Router GW – Router gateway ports

Subnet GW – Subnet gateway ports

As shown in Fig 2.5, the network node consists of 3 OVS Bridges- OVS External bridge (BR-EX), OVS Tunnel Bridge (BR-TUN) and OVS Integration Bridge (BR-INT). All other compute nodes are linked to network node via VXLAN ports on BR-TUN. Note that for each compute node, there is one separate dedicated VXLAN port on BR-TUN. The router ports – router gateway and subnet gateway ports are mapped onto the BR-EX and BR-INT on

network node respectively. The uplink for underlay external network is mapped onto the BR-EX.

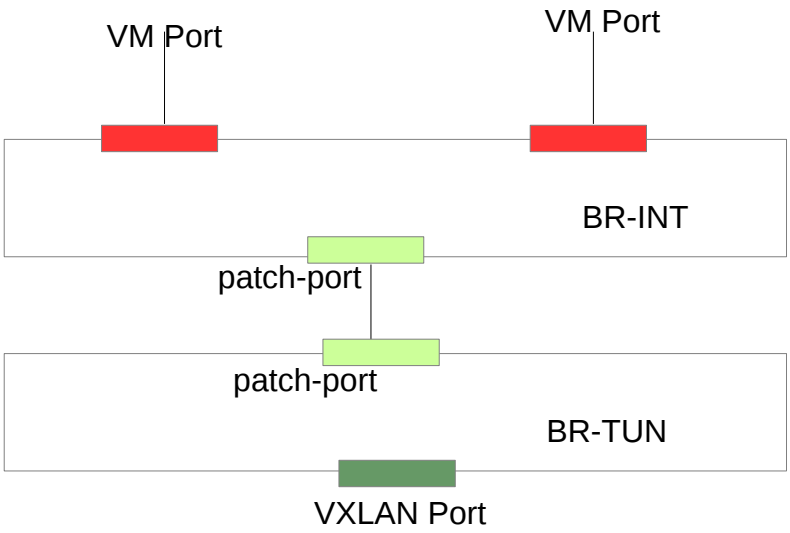
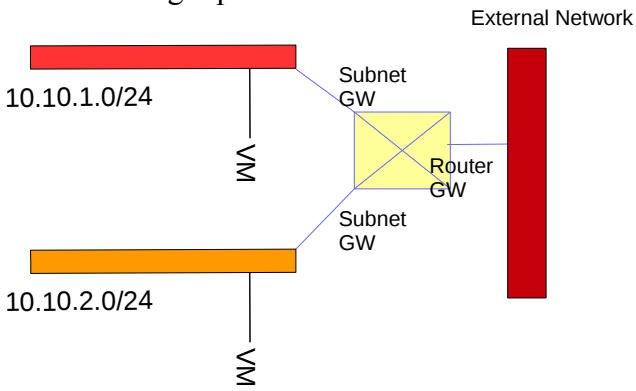


FIG 2.6- COMPUTE NODE

As shown in Fig 2.6, the compute node consists of 2 OVS Bridges- OVS Tunnel Bridge (BR-TUN) and OVS Integration Bridge (BR-INT). All other compute nodes and network nodes are linked to compute node via VXLAN ports on BR-TUN. Note that for each other node (compute and network node), there is one separate dedicated VXLAN port on BR-TUN. All the VM's are hosted on the compute nodes and are mapped onto the BR-INT. For all type of nodes, OVS Integration (BR-INT) and OVS Tunnel (BR-TUN) are linked via patch ports. Also all the ports on BR-INT are VLAN Tagged to separate networks. On the network node, the OVS External Bridge (BR-EX) and BR-INT are linked via linux namespace.

SECTION 5 – OPENSTACK NEUTRON ROUTING INSTANCE

This section describes th existing representation of a Neutron Routing instance.



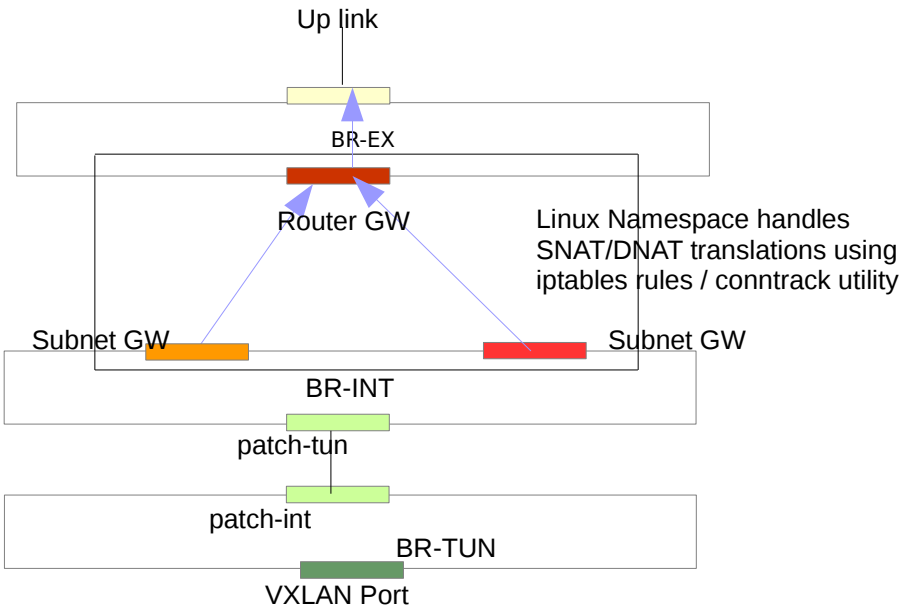


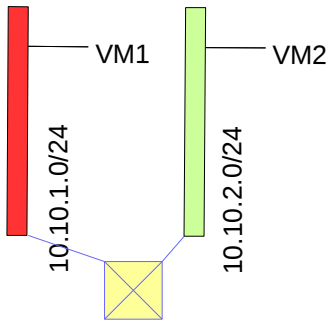
FIG 2.7- NEUTRON ROUTING INSTANCE ON NETWORK NODE

The above figure depicts the representation of a Neutron Routing instance. A neutron routing instance is realized only on the Network node. Each neutron routing instance is represented by a linux namespace including all the router ports – router gateway and subnet gateway ports

All the routing amonst the various ports and NATing (SNAT / DNAT) is handled by the Linux Namespace / Host TCP/IP Stack.

SECTION 6 – OPENSTACK NEUTRON NETWORKING USE-CASES

Use Case 1- Overlay across subnet routing



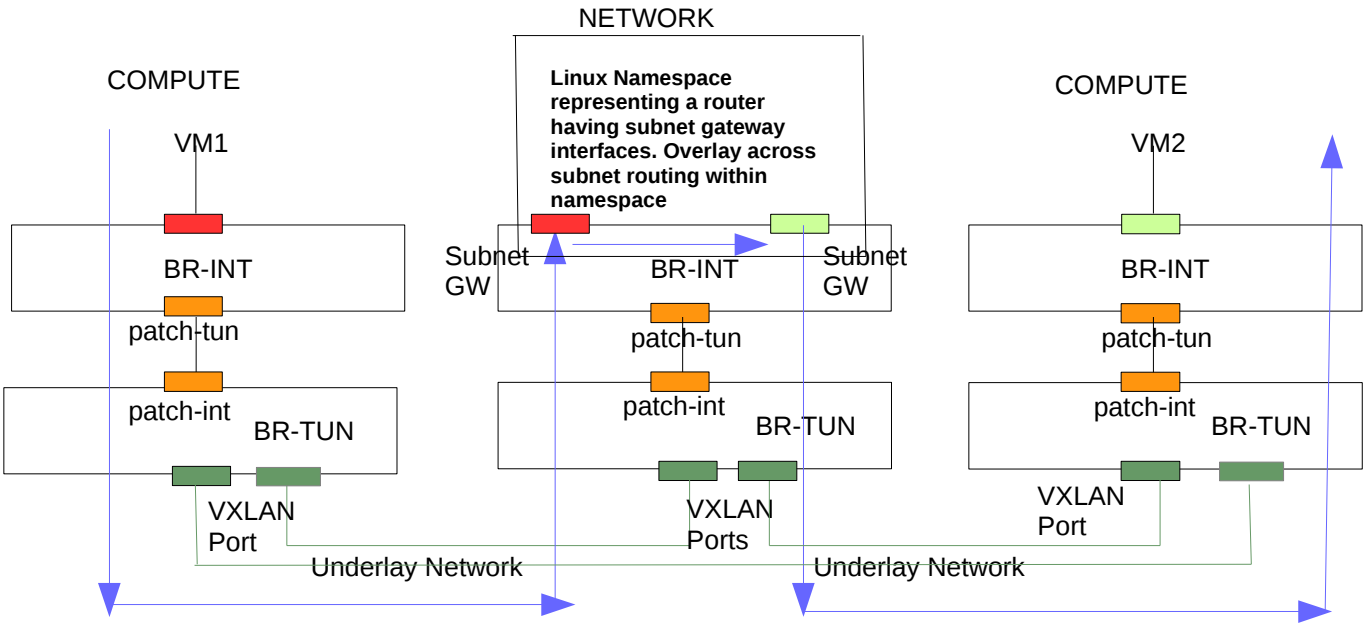
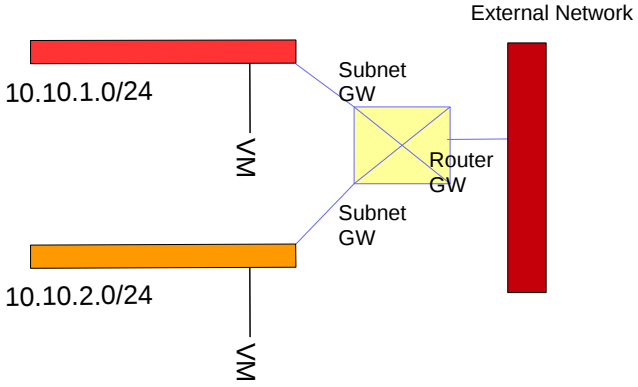


FIG 2.8 - OVERLAY ACROSS SUBNET ROUTING

Fig 2.8 represents across subnet routing path followed in OpenStack. Lets say we have 2 VM's – VM1 and VM2 on 2 different subnet and different compute nodes. The packets from VM1 destined for VM2 always goes through the network node hosting the subnet gateway ports for source and destination subnet. Thus any packet fro overlay across subnet goes through the network node hosting the subnet gateway interfaces. Movement of packets across subnet gateway ports in the routing instance is handled by the Network Node's Host TCP/IP stack. The reply from VM2 to VM1 retrace the same path. Now consider the scenario when both source and destination VM lie on the same compute node. Still the packet would go the network node, despite both source and destination being local to the compute node. This is one problem that we solved in the POC by implementing a Distributed Virtual Router, de-localizing overlay across subnet decision on each compute node instead of going to the network node for the same.

Use Case 2- SNAT / DNAT for overlay to underlay network



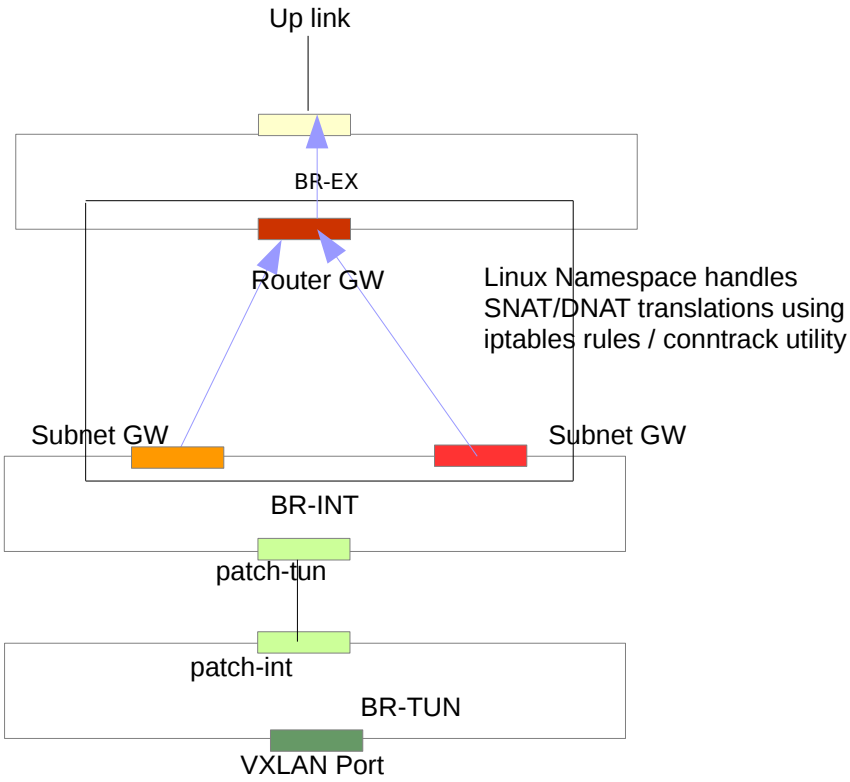


FIG 2.9 - NAT (SNAT/DNAT) ON NETWRK NODE

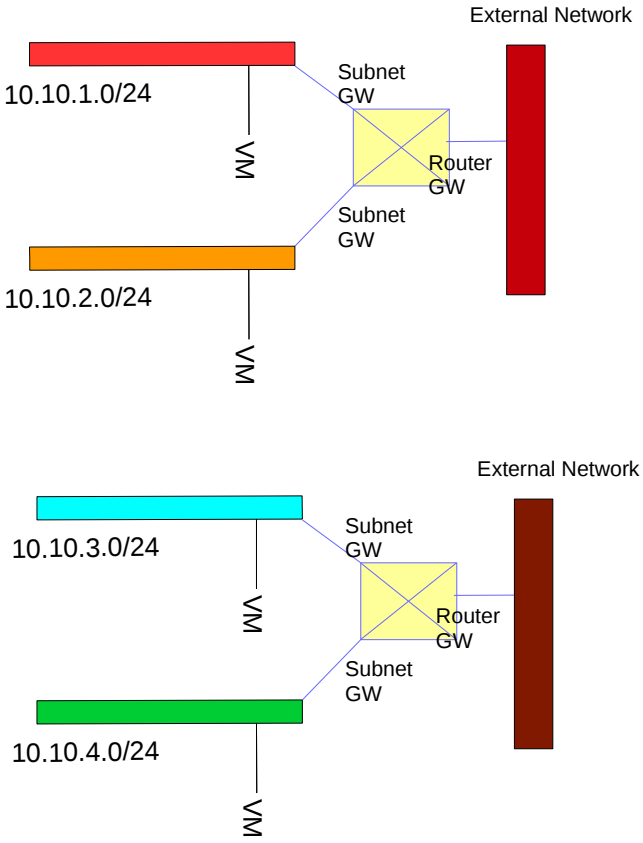
Figure 2.9 depicts the SNAT/DNAT handling by Neutron router. The VM hosted on a compute nodes sends a packet destined for external network to the network node via VXLAN tunnels (fig 2.8). On receiving the packet destined for external network are sent to the source subnet gateway port on OVS integration bridge on network node. From there, the packet is router to the router gateway port on OVS External bridge after NATing (SNAT / DNAT). NATig is handled by the Linux Kernel / Namespace using iptables rules and linux conntrack utility to maintain session for replies. NATing includes translations for source MAC / IP for egress traffic from BR-EX and destination MAC / IP for ingress on BR-EX.

SECTION 6 – OPENSTACK NEUTRON L3 AGENT AND EXTERNAL NETWORKS

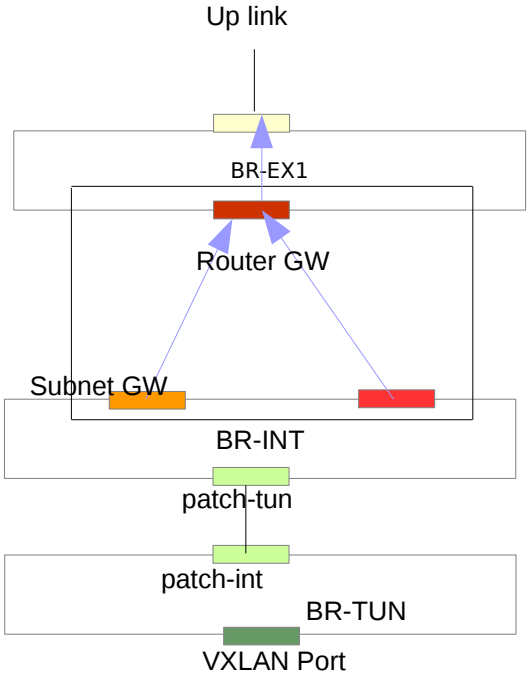
Prior to OpenStack ICEHOUSE release, each neutron l3 agent instance on network node handled 1 external network only. Thus for each external network we needed an instance of neutron-l3-agent on network node. Also each external network uplink is mapped to only 1 OVS External bridge. Thus for 'N' external networks we need 'N' OVS External bridges and 'N' instances of neutron-l3-agent running on Network Node

OpenStack ICEHOUSE release introduced provision for multiple external networks as

provider networks for a common neutron-l3-agent, however, still one OVS bridge corresponds to one external network uplink only



neutron-l3-agent (1st instance)



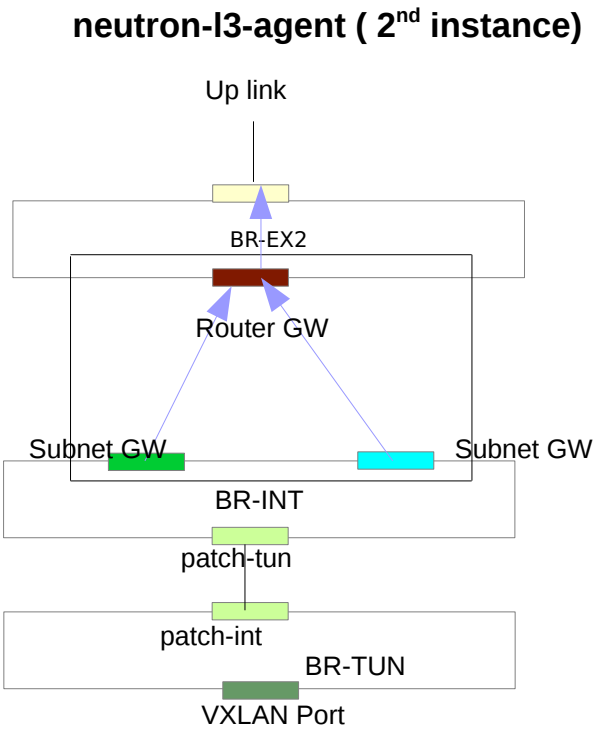


FIG 2.10 - NEUTRON-L3-AGENT FOR EACH EXTERNAL NETWORK ON NETWORK NODE

This was also one of the problems that the POC solved by allowing multiple external networks to be handled by common agent and also providing scalability in terms on mapping of external network uplinks on the OVS external bridges.

CHAPTER 3 – PROBLEM DEFINITION AND SOLUTION OVERVIEW

This chapter deals on defining the problem that the internship work focused on. This is followed by a brief overview on the solution developed as a part of PBI.

SECTION 1 – PROBLEM DEFINITION

The OVS Integration bridge acts as a normal L2 Switch (i.e. single flow having NORMAL action for all packets) and relies on the Open vSwitch Database (OVSDB) for VLAN Tagging / Untagging and packet forwarding. OVS Tunnel contains flows defined via OpenFlow protocol by the neutron-openvswitch-agent for handling packet transmission across hosts and the local VLAN – VXLAN translations (for VXLAN Tunneling). A Neutron Router is represented by a Linux Namespace consisting of router interfaces- subnet gateway, router gateway ports, ARP Cache and routing table associated with router. Neutron-l3-agent is responsible for creation of router ports and namespaces. It is also responsible for adding iptables rules for SNAT/DNAT. Additionally, Network node needs to enable ip forwarding to allow movement of packets across router ports in the linux namespace.

Overlay across subnet and L3 routing / NATing for the traffic generated by the VM's in OpenStack is currently handled completely by the Host using Linux Namespaces, iptables and Host TCP/IP Stack, creating an additional load on the Host machine. In the existing setup (OpenStack HAVANA), each neutron-l3-agent manages one external network and each external network corresponds to only one OVS external bridge. Thus, for each external network, administrators need to launch a different instance of neutron-l3-agent and new OVS external bridge. (OpenStack ICEHOUSE introduced provider network mechanism wherein one single neutron-l3-agent can handle multiple external networks but still one external network corresponds to one OVS external bridge)

SECTION 2 - SOLUTION OVERVIEW

As a part of the new setup, overlay within / across subnet and L3 Routing / NATing is completely handled by the flows defined via OpenFlow protocol on the OVS Integration, Tunnel and External Bridge. The POC implements one unified L2-L3 OpenStack Agent which implements the mentioned functionalities instead of 2 different agents- neutron-openvswitch-agent and neutron-l3-agent. Also, one instance of the new agent handles multiple

external networks instead of existing setup of using one instance of neutron-l3-agent per external network (as of OpenStack HAVANA release). The POC provides scalability in mapping multiple external network up-links on a common OVS bridge.

OpenStack Neutron router is virtualized as a Distributed Virtual Router having instance on each compute node, completely defined using flows which implements routing table and ARP cache (This functionality has been introduced in OpenStack JUNO but our implementation is different). NATing is handled by the flows defined on the OVS external bridge which maintain external network sessions on the bridge. L2 unicast / broadcast is also handled completely via flows which maintain record of VLAN Tags on each port on OVS integration Port and handle packet forwarding, thus, removing the dependency on the OVSDB for the same

The solution provides a more Software Defined Networking (SDN) approach for virtualizing Neutron Router and L3 Routing for overlay to underlay network independent of Host Stack.

CHAPTER 4 – PROOF-OF-CONCEPT (POC) DESCRIPTION

SECTION 1 – POC BRIDGE SETUP

This section presents the OVS bridge setup used for POC. For the POC, we used VxLAN tunneling for connecting multiple compute nodes. The POC was developed for OpenStack HAVANA using ML2 Plugin for OpenStack Neutron (L2 population and ARP Responder enabled) and OVS 2.1.

The OVS Bridge setup on compute remains same but the setup on network node is modified.

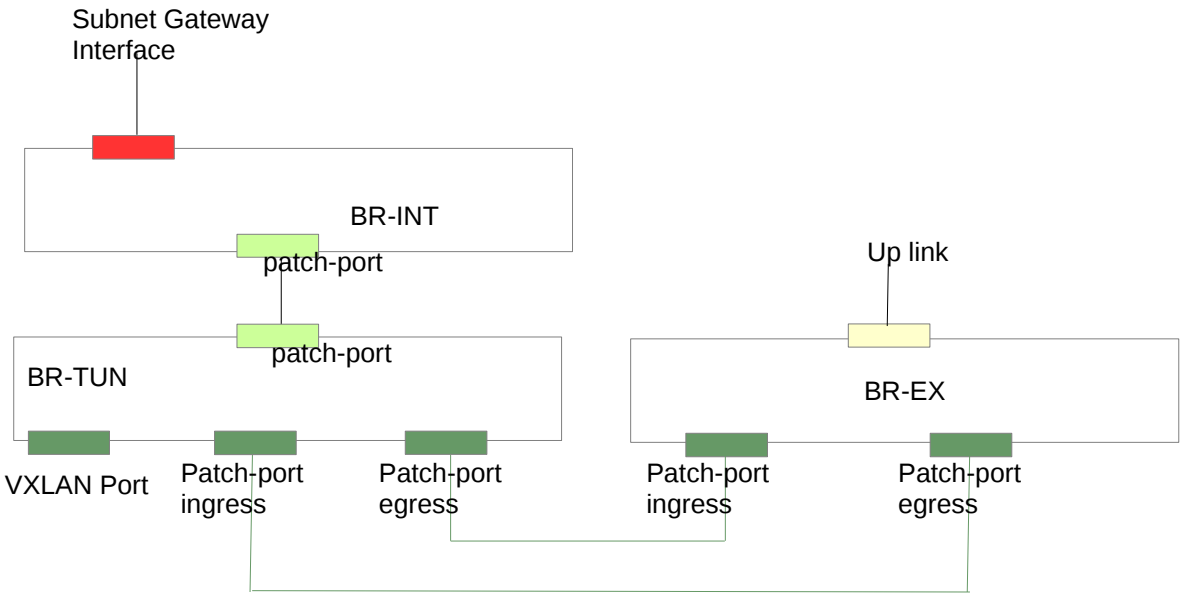


FIG 4.1 – NETWORK NODE

For the new setup, OVS external (BR-EX) and OVS Tunnel (BR-TUN) are linked via 2 patch-ports instead of linking OVS Integration (BR-INT) and OVS external (BR-EX) using linux namespace. One patch-port handles the ingress traffic and other the egress traffic on each bridge.

The compute node remains the same

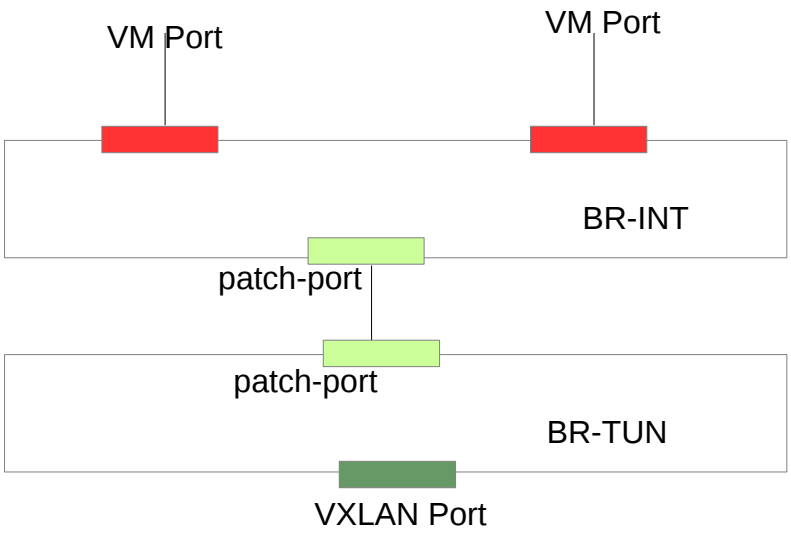


FIG 4.2 – COMPUTE NODE

SECTION 2 – POC DESCRIPTION

POC DESCRIPTION

This section describes the flows defined on OVS integration (BR-INT) , tunnel (BR-TUN) and external (BR-EX) to achieve overlay within / across subnet routing and L3 routing for overlay to underlay (NATing).

L2 Unicast / Broadcast

Flows are defined on BR-INT to handle VLAN tagging / untagging of packets and packet forwarding (L2 unicast / broadcast) amongst ports on BR-INT based on VLAN Tags. For transmission of packet across hosts, flows are defined on BR-TUN, which handle local VLAN – global VXLAN translation for packets to-and-from VXLAN Ports. VXLAN ID for each network is identical throughout compute nodes but the VLAN tags corresponding to each network is local to each compute node and may be different on various nodes. POC uses ARP Responder mechanism, hence, flows are defined on BR-TUN to generate ARP Reply for ARP Request from ports on BR-INT on each host. L2 population mechanism is used which further limits the L2 broadcasting domain amongst VXLAN Tunnels for unknown unicasts.

Overlay across subnet

OpenStack Neutron Router is virtualized as a Distributed Virtual Router having instance on each compute node, using flows defined on BR-INT which implement Routing table / ARP Cache. Each router is identified by a Router-ID managed by the new Agent. The POC currently does not support IP packets destined to (having destination IP address) subnet

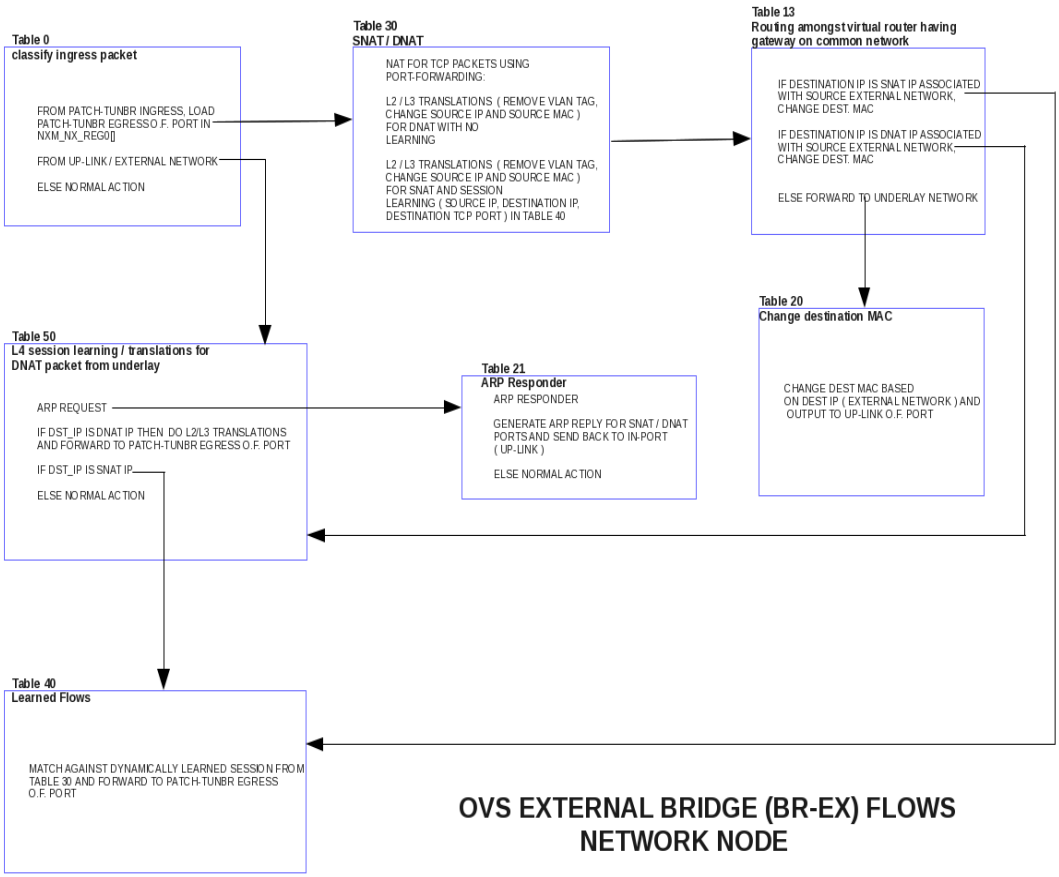
gateway interfaces. Thus, any packet having destination MAC address of subnet gateway is either destined for external network or for another subnet in the overlay network. The intermediate MAC translations are handled by flows on BR-INT

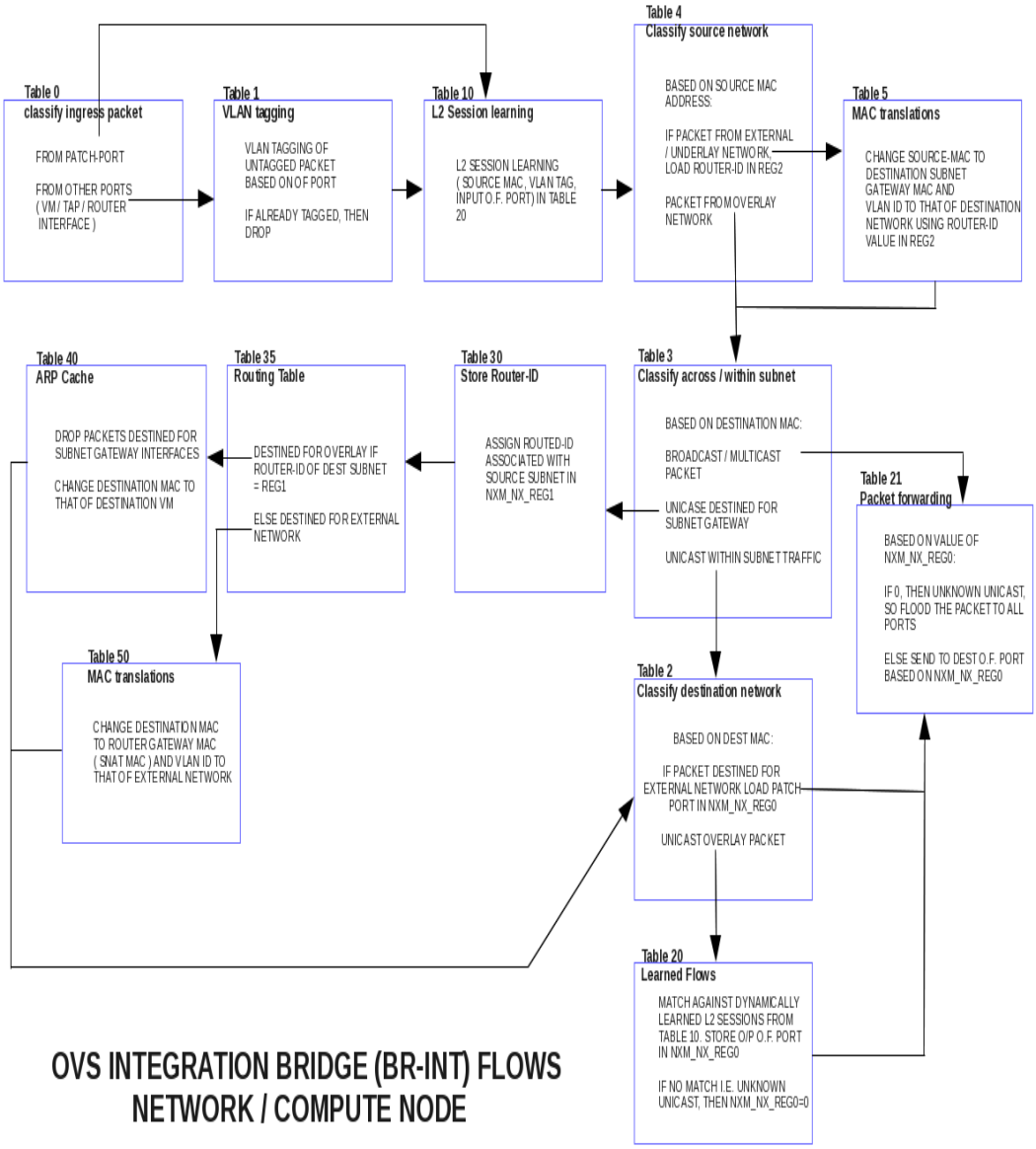
L3 Routing / Network Address Translation (SNAT / DNAT) for overlay to underlay network

On each compute node, flows are defined on BR-TUN for forwarding the packets destined for external network directly to network node. On network node, flows are defined on BR-TUN for deciding if the packet belongs to overlay network or external network. It thus handles the packet forwarding to-and-from BR-EX, VXLAN ports and patch port for BR-INT unambiguously on network node. BR-TUN maintains L3 sessions for both overlay network traffic and external network which ensures that any SNAT sessions are initiated from the overlay network only. BR-EX defines flows for SNAT / DNAT and maintains L4 sessions for TCP / UDP Packets. ARP responder mechanism is also used on the BR-EX to generate ARP Replies for SNAT / DNAT Ports from the underlay network.

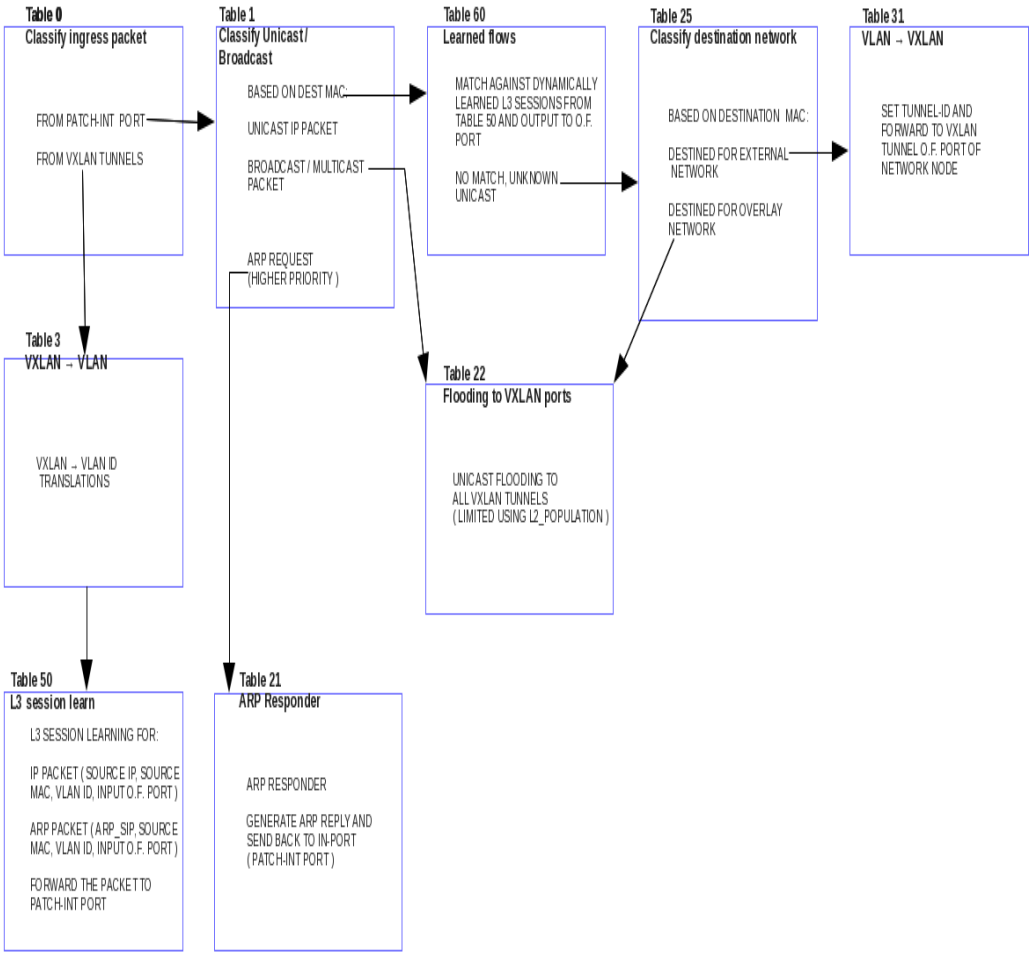
SECTION 3 – FLOW TABLES DESIGN

The following section depicts the flow tables design developed for POC on various OVS Bridges on Compute / Network Node.

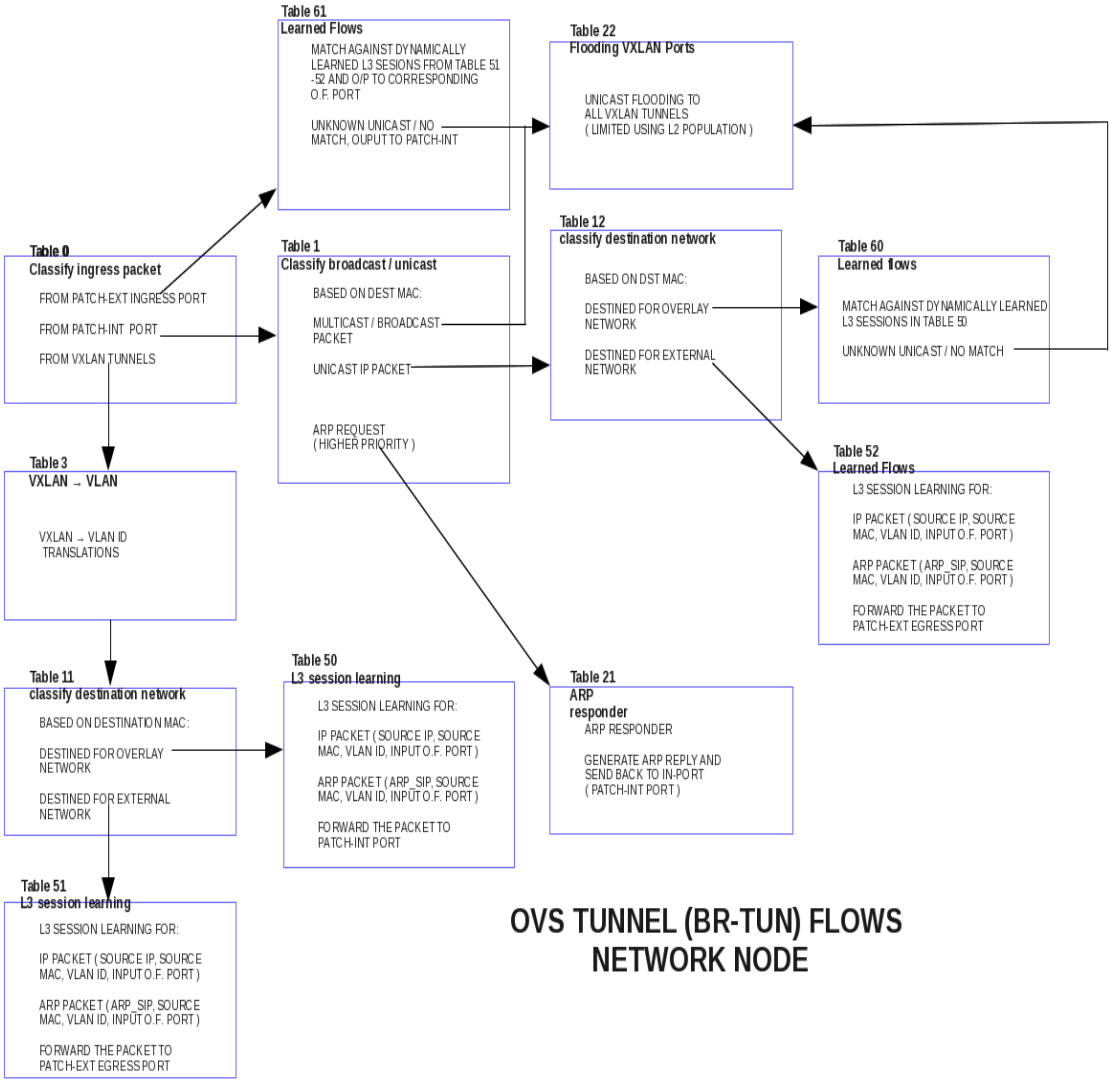




OVS INTEGRATION BRIDGE (BR-INT) FLOWS
NETWORK / COMPUTE NODE



OVS TUNNEL (BR-TUN) FLOWS
COMPUTE NODE



OVS TUNNEL (BR-TUN) FLOWS
NETWORK NODE

SECTION 4 – FLOW TABLES DESCRIPTION

This section provides an overview on the functionality implemented by each table.

OVS Integration Bridge (BR-INT)

The flows on integration bridge are same for compute / network node

Table 0

It handles packet classification on the basis of ingress OpenFlow ports (VM port or patch-port) and thus decides the OpenFlow pipeline to be followed accordingly.

NOTE:

The POC currently does not handle DHCP, hence no consideration for 'tap' ports for DHCP

Table 1

This table handles the VLAN Tagging on untagged packets from VM ports. If the packets are already tagged, it drops them since the packets sent / received by VM's are untagged

Table 10

This is the learning tables, which contains a single flow for maintaining L2 session via flows on Table 20

Sample flow:

cookie=0x0, duration=2545.425s, table=10, n_packets=15, n_bytes=1158, idle_age=850, 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:NXM_OF_IN_PORT[]->NXM_NX_REG0[0..15]),resubmit(,4)

Matching part-

-NXM_OF_VLAN_TCI[0..11]- If the VLAN tag of packet being matched in Table 20 is same as the VLAN Tag of this ingress packet

-NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[]- If the destination MAC address of the packet being matched in Table 20 is same as the source MAC address of this ingress packet

Action part-

-load:NXM_OF_IN_PORT[]->NXM_NX_REG0[0..15]- Load the input OpenFlow port number of the ingress packet in NXM_NX_REG0[] register

Table 4

This tables uses source MAC address and VLAN Tag to decide whether the ingress packet is from overlay network or from the external network. If the packet is from external network, it loads the router-id in NXM_NX_REG2[]

Table 5

This tables is encountered if the ingress packet is from the external network. It-

- Changes source MAC to destination subnet gateway MAC
- Changes VLAN tag to that of the destination network using the router-id in NXM_NX_REG2[] and destination IP address

Table 3

This table uses destination MAC address and VLAN tag to classify ingress packet into- (in the order of priority)

- Broadcast / Multicast Packet (such as ARP / L3 Broadcast)
- Unicast packet having destination MAC of the subnet gateway (overlay across subnet or external network)
- Unicast packet for overlay (within subnet traffic)

NOTE:

Since we do not support IP packets destined for subnet gateway interface, thus, any packet having destination MAC as subnet gateway is destined for either- overlay across subnet or external network.

Table 30

This tables stores the router-id assigned by the agent in NXM_NX_REG1[] on the basis of source subnet CIDR and VLAN tag.

Table 35

This table implements the routing table for a router, where each router is separated using router_id value in NX_NX_REG1[]. This table decides if the packet is destined for any other

subnet having gateway interface on router associated with source subnet using value of NXM_NX_REG1[] and destination subnet CIDR, else forwards it to the router gateway (external network).

For overlay across subnet, this table-

- Changes source MAC to that of the destination subnet gateway MAC
- Changes VLAN tag to that of the destination network

Table 40

This tables implements the ARP Cache for overlay across subnet traffic, thus changing the destination MAC to that of the destination machine using VLAN tag and destination IP. It also implements the rule for dropping IP packets destined for subnet gateway interfaces using destination IP address

Table 50

This tables is encountered if the packet is destined for external network. It-

- Changes destination MAC to that of the router gateway interface (SNAT port)
- Changes VLAN tag to that of the external network

Table 2

This table uses destination MAC address and VLAN Tag to decide if the packet is destined for

- External network- If so, then store the OpenFlow port value of patch port in NXM_NX_REG0[]
- Unicast overlay network

Table 20

This tables stores the learned flows from Table 10 and loads the output OpenFlow port value in NXM_NX_REG0[]

Table 21

This tables forwards the packet to destination OpenFlow port based on value in NXM_NX_REG0[]. In case the value is 0 i.e. for unknown unicast, it does unicast flooding to all ports on OVS integration bridge

OVS Tunnel Bridge (BR-TUN) - Network Node

Table 0

It handles packet classification on the basis of ingress OpenFlow ports (OVS integration patch port, OVS external ingress patch-port, VXLAN Tunnel port) and thus decided the OpenFlow pipeline to be followed accordingly

Table 3

This tables handles VXLAN → VLAN translations for packets ingress from VXLAN Tunnel ports

Table 11

This tables uses destination MAC address and VLAN Tag to classify the destination network into following categories for packets ingress from VXLAN tunnels-

- Overlay network
- External Network (if destination MAC is router gateway MAC)

Table 51

This is the learning table for packets destined for external network ingress from the VXLAN Ports. It defined flows to maintain L3 session for external network on OVS Tunnel on Table 61

Sample flow:

cookie=0x0, duration=2558.717s, table=51, n_packets=0, n_bytes=0, idle_age=2558, priority=1,ip actions=learn(table=61,priority=1,NXM_OF_VLAN_TCI[0..11],NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[],eth_type=0x800,NXM_OF_IP_DST[]=NXM_OF_IP_SRC[],load:0->NXM_OF_VLAN_TCI[],load:NXM_NX_TUN_ID[]->NXM_NX_TUN_ID[],output:NXM_OF_IN_PORT[]),output:<patch_port_for_external_bridge_egress>

Matching part-

-NXM_OF_VLAN_TCI[0..11]- If the VLAN tag of packet being matched in Table 61 is same as the VLAN Tag of this ingress packet

-NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[]- If the destination MAC address of the packet being matched in Table 61 is same as the source MAC address of this ingress packet

-eth_type=0x0800 (IP packet)

-NXM_OF_IP_DST[]=NXM_OF_IP_SRC[]- If the destination IP Address of the packet being matched in Table 61 is same as the source IP address of this ingress packet

Action part-

-load:0->NXM_OF_VLAN_TCI[]- Remove the VLAN Tag from the packet being matched in Table 61

-load:NXM_NX_TUN_ID[]->NXM_NX_TUN_ID[]: Load the Tunnel ID of the packet being matched in Table 61 to be same as that of this ingress packet

-output:NXM_OF_IN_PORT[]: Output the matched packet in Table 61 to the input port of this ingress packet.

Table 1

This table uses destination MAC address to classify ingress packet into (in the order of priority)-

- ARP request (highest priority)
- Broadcast / Multicast Packet (L2 / L3 broadcasts)
- Unicast packet

Table 61

This table maintains the learned flows / L3 sessions for external network from table 51 and 52 and forwards the packet to destined OpenFlow port. In case of no match (for e.g. for DNAT case, where the session can be initiated from outside), the packet is unicast flooded to all VXLAN Tunnels and patch port of OVS Integration bridge

Table 22

This tables handles the unicast flooding of packet including VLAN to VXLAN translations, in case the destination VXLAN Tunnel port is unknown (e.g. DNAT session initiated from external network)

Table 50

This is the learning table for packets destined for overlay network from the VXLAN Ports. It defined flows to maintain L3 session for external network on OVS Tunnel on Table 60

Sample flow:

cookie=0x0, duration=2560.551s, table=50, n_packets=1, n_bytes=98, idle_age=861, priority=1,ip actions=learn(table=60,hard_timeout=300,priority=1,eth_type=0x800,NXM_OF_VLAN_TCI[0..11],NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[],NXM_OF_IP_DST[]=NXM_OF_IP_SRC[],load:0->NXM_OF_VLAN_TCI[],load:NXM_NX_TUN_ID[]->NXM_NX_TUN_ID[],output:NXM_OF_IN_PORT[]),output:<patch_port_for_integration_bridge>

Matching part-

-NXM_OF_VLAN_TCI[0..11]- If the VLAN tag of packet being matched in Table 60 is same as the VLAN Tag of this ingress packet

-NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[]- If the destination MAC address of the packet being matched in Table 60 is same as the source MAC address of this ingress packet

-eth_type=0x0800 (IP packet)

-NXM_OF_IP_DST[]=NXM_OF_IP_SRC[]- If the destination IP Address of the packet being matched in Table 60 is same as the source IP address of this ingress packet

Action part-

-load:0->NXM_OF_VLAN_TCI[]- Remove the VLAN Tag from the packet being matched in Table 60

-load:NXM_NX_TUN_ID[]->NXM_NX_TUN_ID[]: Load the Tunnel ID of the matched packet in Table 60 to be same as that of this ingress packet

-output:NXM_OF_IN_PORT[]: Output the matched packet in Table 60 to the incoming port of this ingress packet.

Table 60

This table maintains the learned flows / L3 sessions for overlay network from table 50 and forwards the packet to destined OpenFlow port

Table 21

This table implements the flows for ARP Responder. It generates ARP Replies for ARP

Requests generated by ports on the OVS integration Bridge local to each host

Sample flow:

cookie=0x0, duration=1935.722s, table=21, n_packets=0, n_bytes=0, idle_age=1935, priority=1,arp,dl_vlan=1,arp_tpa=10.10.1.2 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],mod_dl_src:fa:16:3e:b3:6c:31,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:0xfa163eb36c31->NXM_NX_ARP_SHA[],load:0xa0a0102->NXM_OF_ARP_SPA[],IN_PORT

The flow generates an ARP reply for an ARP request for a node having IP address 10.10.1.2 and belonging to network having VLAN 1

The ARP reply contains MAC address- fa:16:3e:b3:6c:31 and ARP_SHA - 0xfa163eb36c31 (fa:16:3e:b3:6c:31) and op code 2

Table 12

This table uses destination MAC address and VLAN tag to classify destination network into following for packets ingress from OVS integration bridge patch port-

- Overlay network
- External Network

Table 52

This is the learning table for packets destined for external network from the ports on OVS integration bridge. It defined flows to maintain L3 session for external network on OVS Tunnel on Table 61

Sample flow:

cookie=0x0, duration=2558.475s, table=52, n_packets=0, n_bytes=0, idle_age=2558, priority=1,ip actions=learn(table=61,priority=1,NXM_OF_VLAN_TCI[0..11],NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[],eth_type=0x800,NXM_OF_IP_DST[]=NXM_OF_IP_SRC[],output:NXM_OF_IN_PORT[]),output:<patch_port_for_external_bridge_egress>

Matching part-

-NXM_OF_VLAN_TCI[0..11]- If the VLAN tag of packet being matched in Table 61 is same as the VLAN Tag of this ingress packet

-NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[]- If the destination MAC address of the packet being matched in Table 61 is same as the source MAC address of this ingress packet

-eth_type=0x0800 (IP packet)

-NXM_OF_IP_DST[]=NXM_OF_IP_SRC[]- If the destination IP Address of the packet being matched in Table 61 is same as the source IP address of this ingress packet

Action part-

-output:NXM_OF_IN_PORT[]: Output the matched packet in Table 61 to the input port of this ingress packet.

OVS Tunnel Bridge (BR-TUN) - Compute Node

Table 0

It handles packet classification on the basis of ingress OpenFlow ports (OVS integration patch port, VXLAN Tunnel port) and thus decided the OpenFlow pipeline to be followed accordingly

Table 3

This tables handles VXLAN → VLAN translations for packets ingress from VXLAN Tunnel ports

Table 1

This table uses destination MAC address to classify ingress packet into (in the order of priority)-

- ARP request (highest priority)
- Broadcast / Multicast Packet (L2 / L3 Broadcasts)
- Unicast packet

Table 50

This is the learning table for packets destined for overlay / external network from the VXLAN

Ports. It defined flows to maintain L3 session for external network on OVS Tunnel on Table 60

Sample flow:

cookie=0x0, duration=2485.194s, table=50, n_packets=31, n_bytes=3038, idle_age=808, priority=1,ip actions=learn(table=60,hard_timeout=300,priority=1,eth_type=0x800,NXM_OF_VLAN_TCI[0..11],NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[],NXM_OF_IP_DST[]=NXM_OF_IP_SRC[],load:0->NXM_OF_VLAN_TCI[],load:NXM_NX_TUN_ID[]->NXM_NX_TUN_ID[],output:NXM_OF_IN_PORT[]),output:<patch_port_for_integration_bridge>

Matching part-

-NXM_OF_VLAN_TCI[0..11]- If the VLAN tag of packet being matched in table 60 is same as the VLAN Tag of this ingress packet

-NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[]- If the destination MAC address of the packet being matched in table 60 is same as the source MAC address of this ingress packet

-eth_type=0x0800 (IP packet)

-NXM_OF_IP_DST[]=NXM_OF_IP_SRC[]- If the destination IP Address of the packet being matched in table 60 is same as the source IP address of this ingress packet

Action part-

-load:0->NXM_OF_VLAN_TCI[]- Remove the VLAN Tag from the packet being matched in table 60

-load:NXM_NX_TUN_ID[]->NXM_NX_TUN_ID[]: Load the Tunnel ID of the matched packet in table 60 to be same as that of this ingress packet

-output:NXM_OF_IN_PORT[]: Output the matched packet in table 60 to the incoming port of this ingress packet.

Table 22

This tables handles the unicast flooding of packet in case the destination VXLAN Tunnel port is unknown

Table 21

This table implements the flows for ARP Responder. It generates ARP Replies for ARP Requests
generated by ports on the OVS integration Bridge on the host
(Same as Table 21 for OVS Tunnel Flows on Network Node)

Table 60

This table maintains the learned flows / L3 sessions from table 50 and forwards the packet to destined OpenFlow port

Table 25

This table uses destination MAC address and VLAN Tag to decide if the destination network is-

- Overlay network
- External Network (if destination MAC is router gateway MAC)

Table 31

This tables forwards the packet destined for external network, directly to the network node

OVS External Bridge (BR-EX) - Network Node

Table 0

It handles packet classification on the basis of ingress OpenFlow ports (OVS tunnel ingress patch port or Up-link interface) and thus decided the OpenFlow pipeline to be followed accordingly

Table 50

This table classifies the ingress packet into following categories (in the order of priority)-

- ARP Request for SNAT (router gateway) / DNAT (floating-ip) interfaces
- IP packet having destination IP of DNAT interface (floating-ip)
- IP packet having destination IP of SNAT interface (router gateway)

For packet having destination IP of DNAT interface, the flows perform following translations-

- Add VLAN Tag of the external network

- Change source MAC to that of the destination subnet gateway MAC
- Change destination MAC to that of the destination VM
- Change destination IP to that of the destination VM

Table 21

This tables implements flows for ARP Responder mechanism for the SNAT/DNAT ports. It generates ARP Replies for ARP request for SNAT/DNAT ports from the external (underlay) network

NOTE

In OpenStack setup, all SNAT / DNAT ports associated with a Neutron router are represented by one single port on the OVS external bridge i.e. a single port shares multiple IP addresses. Thus, ARP Responder flows reply same MAC address for all the DNAT ports and SNAT port associated with a single router.

Table 30

This tables handles SNAT / DNATing using Port forwarding and creates learning flows to maintain L4 sessions for TCP packets in Table 40.

Sample flow for SNAT:

cookie=0x0, duration=423.525s, table=30, n_packets=0, n_bytes=0, idle_age=423, priority=5,tcp,vlan_tci=0x0006/0x0fff,dl_dst=fa:16:3e:b7:62:bc actions=learn(table=40,priority=1,eth_type=0x800,nw_proto=6,NXM_OF_TCP_DST[]=NXM_OF_TCP_SRC[],NXM_OF_IP_SRC[]=NXM_OF_IP_DST[],NXM_OF_TCP_SRC[]=NXM_OF_TCP_DST[],NXM_OF_ETH_DST[],load:NXM_OF_VLAN_TCI[0..11]->NXM_OF_VLAN_TCI[0..11],load:NXM_OF_IP_SRC[]->NXM_OF_IP_DST[],load:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],load:NXM_OF_ETH_DST[]->NXM_OF_ETH_SRC[],output:NXM_OF_IN_PORT[]),strip_vlan,mod_nw_src:9.121.62.77,mod_dl_src:fa:16:3e:b7:62:bc,resubmit(,20)

The above flow represents a SNAT translation flow and the learning tables flow created corresponding for any packet having destination MAC address of fa:16:3e:b7:62:bc and VLAN Tag 6 are SNATed to source IP address of 9.121.62.77 and source MAC address of fa:16:3e:b7:62:bc

Matching Part-

- eth_type=0x800 (IP Packet)
- nw_proto=6 (TCP)
- NXM_OF_TCP_DST[]=NXM_OF_TCP_SRC[] - If the destination TCP port of the packet being matched in Table 40 is same as the source TCP port of the ingress packet
- NXM_OF_TCP_SRC[]=NXM_OF_TCP_DST[] - If the source TCP port of the packet being matched in Table 40 is same as the destination TCP port of the ingress packet
- NXM_OF_IP_SRC[]=NXM_OF_IP_DST[] - If the source IP address of the packet being matched in Table 40 is same as the destination IP address of the ingress packet
- NXM_OF_ETH_DST[] - If the destination MAC address of the packet being matched in Table 40 is same as the destination MAC address of the ingress packet

Action part-

- load:NXM_OF_VLAN_TCI[0..11]->NXM_OF_VLAN_TCI[0..11] – Remove VLAN tag of the packet being matched
- load:NXM_OF_IP_SRC[]->NXM_OF_IP_DST[] - Change destination IP address of the packet being matched in table 40 to the source IP of the ingress packet
- load:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[] - Change destination MAC address of the packet being matched in table 40 to the source MAC address of the ingress packet
- load:NXM_OF_ETH_DST[]->NXM_OF_ETH_SRC[] - Change source MAC address of the packet being matched in table 40 to the destination MAC address of the ingress packet
- output:NXM_OF_IN_PORT[] - Output the packet being matched in table 40 to the ingress packet OpenFlow port

DNAT translations include:

- Changing source MAC to that of the SNAT (router gateway) MAC
- Change source IP to DNAT (router gateway) IP

There is no learning needed for the DNAT session since this can be initiated form the external network as well

Sample flow:

cookie=0x0, duration=309.672s, table=30, n_packets=0, n_bytes=0, idle_age=309, priority=10,ip,vlan_tci=0x0005/0x0fff,dl_dst=fa:16:3e:bc:0a:f3,nw_src=10.10.2.3 actions=mod_nw_src:9.121.62.74,mod_dl_src:fa:16:3e:bc:0a:f3,resubmit(,13)

The above flows shows DNAT for any packet having source IP of 10.10.2.3 and VLAN tag 5. The packet is DNATed to destination IP 9.121.62.74 and source MAC of fa:16:3e:bc:0a:f3 and resubmitted to Table 13 (As mentioned earlier, the source MAC for both SNAT/DNAT is same (router gateway MAC))

Table 13

This table handles routing amongst VM's in overlay network connected via different virtual routers (using SNAT / DNAT IP). For e.g. a VM in the overlay initiates an IP session for floatingIP- so in this case the source and destination both lie in the overlay.

This tables has 2 functionalities (in the order of priority)-

- It drops any packet initiated from non floatingIP-associated VM destined for its own SNAT IP (this is the case when source and destination IP address are same after SNAT for non-floatingIP-associated VM's)
- Check for routing amongst VM's on networks connected by different virtual routers. (if the source and destination both lie in the overlay itself, for e.g.- IP session for floatingIP's). If so, this table handles destination MAC address translations for valid traffic.

If none of the flows match, then the VLAN tag is remove and packet is forwarded to the underlay external network

Table 40

This tables maintains the learned flows from table 30

Table 20

This tables implements the ARP Cache for external network (underlay network nodes). The flows for this table needs to be added manually by the user.

SECTION 5 - FUNCTIONALITIES SUPPORTED BY POC

The POC in its current form cannot maintain unique ICMP SNAT session, since we cannot currently access ICMP identifier field. ICMP SNAT session for a given destination IP can be initiated by only one VM at a time, amongst a group of non floatingIP-associated VM's sharing common SNAT IP. TCP SNAT sessions can still be maintained uniquely because the learning is based on MAC address, IP address, IP protocol and TCP ports. Note that we currenlty use Port Forwarding and not Port Mapping (The ideal way is to implement port mapping and not port forwarding)

Sample learned flow in table 40 on OVS external bridge for ICMP

table=40, hard_timeout=300,
priority=1,**icmp,dl_dst=fa:16:3e:12:58:cb,nw_src=9.121.62.67** actions=load:0x5->NXM_OF_VLAN_TCI[0..11],load:0xa0a0203->NXM_OF_IP_DST[],load:0xfa163ecdbc43->NXM_OF_ETH_DST[],load:0xfa163e1258cb->NXM_OF_ETH_SRC[],output:3

Multiple VM's sharing common SNAT IP share common router gateway MAC so we cannot uniquely identify the ICMP SNAT sessions. ICMP SNAT session learning is based on destination MAC, source IP and protocol only

Sample learned flow in table 40 on OVS external bridge for TCP

table=40, hard_timeout=300,
prority=1,**tcp,dl_dst=fa:16:3e:12:58:cb,nw_src=9.121.62.66,tp_src=22,tp_dst=37965** actions=load:0x5->NXM_OF_VLAN_TCI[0..11],load:0xa0a0103->NXM_OF_IP_DST[],load:0xfa163ebbb36a->NXM_OF_ETH_DST[],load:0xfa163e1258cb->NXM_OF_ETH_SRC[],output:3

TCP SNAT session involves TCP source and destination ports additionally
For example, if we have 4 VM's sharing subnet gateway on one common router and another 4 VM's sharing subnet gateway on another router, with no floatingIP-associations, as shown below then we have 2 groups-

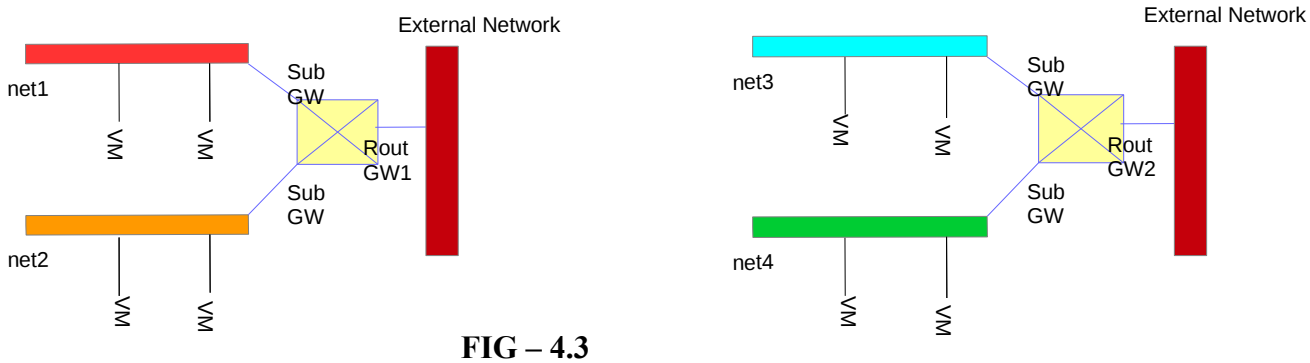


FIG – 4.3

Group 1- 4 VM's → inst_net1, inst_net1_2, inst_net2, inst_net2_2 (SNAT_IP1, router_gateway_MAC1)

Group 2- 4 VM's → inst_net3, inst_net3_2, inst_net4, inst_net4_2 (SNAT_IP2, router_gateway_MAC2)

Hence for each group mentioned above only one VM within a group can be used for ICMP SNATing for a given external network destination IP at a time.

Lets consider following specification for Group 1-

VM	IP	MAC
inst_net1	IP_net1	MAC_net1
inst_net1_2	IP_net1_2	MAC_net1_2
inst_net2	IP_net2	MAC_net2
inst_net2_2	IP_net2_2	MAC_net2_2

Lets consider following specification for Group 2-

VM	IP	MAC
inst_net3	IP_net3	MAC_net3
inst_net3_2	IP_net3_2	MAC_net3_2
inst_net4	IP_net4	MAC_net4
inst_net4_2	IP_net4_2	MAC_net4_2

Consider, inst_net1 starts an ICMP session for external node having IP (9.121.62.66). So on OVS external bridge, following rule is encountered in Table 30 on OVS external bridge (BR-EX) -

table=30,dl_type=0x0800,nw_proto=1,vlan_tci=0x0005/0x0fff**,dl_dst=fa:16:3e:b7:62:bc**,actions=learn(table=40,priority=1,eth_type=0x800,nw_proto=6,**NXM_OF_TCP_DST[]=NXM_OF_TCP_SRC[],NXM_OF_IP_SRC[]=NXM_OF_IP_DST[],NXM_OF_TCP_SRC[]=NXM_OF_TCP_DST[],NXM_OF_ETH_DST[],load:NXM_OF_VLAN_TCI[0..11]->NXM_OF_VLAN_TCI[0..11],load:NXM_OF_IP_SRC[]->NXM_OF_IP_DST[],load:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],load:NXM_OF_ETH_DST[]->NXM_OF_ETH_SRC[]**,output:NXM_OF_IN_PORT[]),strip_vlan,mod_nw_src:**9.121.62.77**,mod_dl_src:**fa:16:3e:b7:62:bc**,resubmit(,20)
where
fa:16:3e:b7:62:bc – router_gateway_MAC1
9.121.62.77- SNAT_IP1

So the packet from inst_net1 (VM) is given a SNAT IP of 9.121.62.77 with router gateway MAC of fa:16:3e:b7:62:bc and forwarded to underlay network. Also a learning flow is created for this session in Table 40 on OVS external bridge (BR-EX)

table=40, hard_timeout=300,
priority=1,**icmp,dl_dst=fa:16:3e:b7:62:bc,nw_src=9.121.62.66** actions=load:0x5->NXM_OF_VLAN_TCI[0..11],load:**0xa0a0203**->NXM_OF_IP_DST[],load:**0xfa163ecdbc43**->NXM_OF_ETH_DST[],load:**0xfa163eb762bc**->NXM_OF_ETH_SRC[],output:3

where -
0xa0a0203 → IP_net1
0xfa163ecdbc43 → MAC_net1
0xfa163eb762bc → router_gateway_MAC1

Now note that in Table 40, for maintaining the session of VM inst_net1, we only use destination MAC address and source IP address of the ICMP reply coming in.

Also now lets assume if another VM- inst_net1_2 also initiates an ICMP session for external network (9.121.62.66). So table 30 on OVS external bridge (BR-EX) is encountered again-

table=30,dl_type=0x0800,nw_proto=1,vlan_tci=0x0005/0x0fff,**dl_dst=fa:16:3e:b7:62:bc**,actions=learn(table=40,priority=1,eth_type=0x800,nw_proto=6,**NXM_OF_TCP_DST[]=NXM_OF_TCP_SRC[],NXM_OF_IP_SRC[]=NXM_OF_IP_DST[],NXM_OF_TCP_SRC[]=NXM_OF_TCP_DST[],NXM_OF_ETH_DST[],load:NXM_OF_VLAN_TCI[0..11]->NXM_OF_VLAN_TCI[0..11],load:NXM_OF_IP_SRC[]->NXM_OF_IP_DST[],load:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],load:NXM_OF_ETH_DST[]->NXM_OF_ETH_SRC[]**,output:NXM_OF_IN_PORT[]),strip_vlan,mod_nw_src:**9.121.62.77**,mod_dl_src:**fa:16:3e:b7:62:bc**,resubmit(,20)
where
fa:16:3e:b7:62:bc – router_gateway_MAC1
9.121.62.77- SNAT_IP1

So the packet from inst_net1_2 (VM) is given a SNAT IP of 9.121.62.77 with router gateway MAC of fa:16:3e:b7:62:bc and forwarded to underlay network. Also a learning flow is created for this session in Table 40 on OVS external bridge (BR-EX)

table=40, hard_timeout=300,
priority=1,**icmp,dl_dst=fa:16:3e:b7:62:bc,nw_src=9.121.62.66** actions=load:0x5->NXM_OF_VLAN_TCI[0..11],load:**0xa0a0203**->NXM_OF_IP_DST[],load:**0xfa163ecdbc43**->NXM_OF_ETH_DST[],load:**0xfa163eb762bc**->NXM_OF_ETH_SRC[],output:3

where -

0xa0a0203	→	IP_net1
0xfa163ecdbc43	→	MAC_net1
0xfa163eb762bc	→	router_gateway_MAC1

Now note that in Table 40 on OVS external bridge (BR-EX), for maintaining the session of VM inst_net1_2, we only use destination MAC address and source IP address of the ICMP reply coming in.

So we observe that for a group of VM's sharing common SNAT IP / router gateway MAC (all the VM's in Group 1 - inst_net1, inst_net1_1, inst_net2, inst_net2_2), if they are all generating ICMP sessions for same destination IP (here 9.121.62.66) at the same time, they share common Router Gateway MAC and source IP for the learned flow session in Table 40 on OVS external bridge (BR-EX). Hence, we cannot differentiate amongst different ICMP sessions for different VM's within a group sharing common SNAT IP and router gateway MAC.

Same applies for group 2 as well

However, all such groups can be checked for the common destination IP simultaneously because they have different SNAT IP and router gateway MAC address but only one within a group.

That is if lets say 2 VM's of different grupus- inst_net1 (Group 1) and inst_net3 (Group 2) start ICM session for common destination IP (9.121.62.66), both will hit tables 30 in OVS external bridge but will hit different flows on table 30 because they have different router gateway MAC, so "dl_dst" parameter would be different for both.

Example-
inst_net1 will hit-

table=30,dl_type=0x0800,nw_proto=1,vlan_tci=0x0005/0x0fff**,dl_dst=fa:16:3e:b7:62:bc**,actions=learn(table=40,priority=1,eth_type=0x800,nw_proto=6,**NXM_OF_TCP_DST[]=NXM_OF_TCP_SRC[],NXM_OF_IP_SRC[]=NXM_OF_IP_DST[],NXM_OF_TCP_SRC[]=NXM_OF_TCP_D**
ST[],NXM_OF_ETH_DST[],load:NXM_OF_VLAN_TCI[0..11]->NXM_OF_VLAN_TCI[0..11],load:NXM_OF_IP_SRC[]->NXM_OF_IP_DST[],load:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],load:NXM_OF_ETH_DST[]->NXM_OF_ETH_SRC[],output:NXM_OF_IN_PORT[]),strip_vlan,mod_nw_src:**9.121.62.77**,mod_dl_src:**fa:16:3e:b7:62:bc**,resubmit(,20)
where
fa:16:3e:b7:62:bc – router_gateway_MAC1

9.121.62.77- SNAT_IP1
and create a learned flow in table 40 on OVS external bridge (BR-EX)

table=40, hard_timeout=300,
priority=1,**icmp,dl_dst=fa:16:3e:b7:62:bc,nw_src=9.121.62.66** actions=load:0x5-
>NXM_OF_VLAN_TCI[0..11],load:**0xa0a0203**-
>NXM_OF_IP_DST[],load:**0xfa163ecdbc43**-
>NXM_OF_ETH_DST[],load:**0xfa163eb762bc**->NXM_OF_ETH_SRC[],output:3
where -
0xa0a0203 → IP_net1
0xfa163ecdbc43 → MAC_net1
0xfa163eb762bc → router_gateway_MAC1

and inst_net3's packet will hit following table 30 flow on OVS external bridge (BR-EX)-

table=30,dl_type=0x0800,nw_proto=1,vlan_tci=0x0005/0x0fff,**dl_dst=fa:16:3e:ef:fa:e8**,acti
ons=learn(table=40,hard_timeout=300,priority=1,eth_type=0x800,**NXM_OF_IP_PROTO[],
NXM_OF_IP_SRC[]=NXM_OF_IP_DST[],NXM_OF_ETH_DST[],load:NXM_OF_VL
AN_TCI[0..11]->NXM_OF_VLAN_TCI[0..11],load:NXM_OF_IP_SRC[]-
>NXM_OF_IP_DST[],load:NXM_OF_ETH_SRC[]-
>NXM_OF_ETH_DST[],load:NXM_OF_ETH_DST[]-
>NXM_OF_ETH_SRC[],**output:NXM_NX_REG0[]),mod_nw_src:**9.121.62.70**,mod_dl_src:
fa:16:3e:ef:fa:e8,resubmit(,20)

where
fa:16:3e:ef:fa:e8 – router_gateway_MAC2
9.121.62.70- SNAT_IP2
and create a learned flow in table 40 on OVS external bridge (BR-EX)
table=40, hard_timeout=300, priority=1,**icmp,dl_dst=fa:16:3e:ef:fa:e8,nw_src=9.121.62.66**
actions=load:0x5->NXM_OF_VLAN_TCI[0..11],load:**0xa0a0103**-
>NXM_OF_IP_DST[],load:**0xfa163ecdab43**->NXM_OF_ETH_DST[],load:**0xfa163eeffae8**-
>NXM_OF_ETH_SRC[],output:3
where -
0xa0a0103 → IP_net3

0xfa163ecdab43 → MAC_net3
0xfa163eeffae8 → router_gateway_MAC2

Thus two different sessions are maintained for 2 VM's belonging to two different groups with each group having common SNAT IP and router gateway MAC.

So we can summarize, for a node in external network having IP- 9.121.62.66 or 9.121.62.67

Only one of the VM's in Group 1 can Ping 9.121.62.66 at a time because they share common router gateway MAC, and,

Only one of the VM's in Group 2 can Ping 9.121.62.66 at a time because they share common router gateway MAC, but

2 different VM's in Group 1 can ping together considering one pings 9.121.2.66 and other pings 9.121.62.67 at the same time
Same holds for Group 2, also,

A VM from Group 1 and another VM from Group 2 can both ping 9.121.62.66 at the same time since they have different router gateway MAC

There is no such hard constraint for TCP SNAT sessions. For TCP packets, Port Forwarding is used so ports collision is less likely amongst the VM's in each group since TCP session learning involves source and destination TCP ports. However, collision is still possible if the source and destination port for any 2 VM's SNAT session in the same group happens to be same. User can initiate TCP sessions simultaneously (e.g. ssh sessions to external nodes) from VM's within a group. To ensure that no port collision is happening for TCP packets, user can dump flows in Table 40 on OVS external bridge, which maintains SNAT TCP sessions. Thus total number of unique TCP SNAT sessions being maintained can be cross checked from these dumps in Table 40. Solution for this port collision is described in "FUTURE WORK" section

Another drawback for SNAT is that since we cannot uniquely identify the SNAT ICMP session, thus any ICMP packet destined for SNAT IP from external network may be accepted

instead of being dropped and forwarded to the VM in the overlay. (SNAT sessions cannot be initiated from the outside / underlay). This may happen if lets say VM (10.10.1.2) is pinging external node (9.121.62.66) with SNATed IP 9.121.62.70. Now a learning session is created on BR-EX / OVS external bridge based on destination MAC address, source IP address and IP protocol in Table 40 for the session. Now, if the connection is terminated by the VM the learning flows still remain on the bridge and are removed after a certain time-out value (100 seconds for POC). Now if the external node (9.121.62.66) initiates an ICMP session for 9.121.62.70 (SNATed IP) before time-out, then the flows cannot determine if the session was initiated from overlay or the underlay and would forward the packet to the same overlay VM as if the session was initiated by the VM in the overlay.

The POC currently does not support IP packets destined for SNAT interfaces. Thus, users need to ensure that no sessions are initiated from outside to SNAT IP. (This problem arises since we cannot identify SNAT sessions uniquely).

NOTE:
Since floatingIP's are part of external network, all the constraint defined for external network IP also apply to floatingIP's.

For ICMP SNAT, only single VM within a group sharing common SNAT IP must ping a common floatingIP at a time. Multiple VM's within a group sharing common SNAT IP are not allowed to send ICMP packets destined for a common floating-IP (Since we cannot uniquely identify each SNAT session, as mentioned in detail earlier in detail for external networks)

For example, if we have 4 VM's sharing subnet gateway on one common router and another 4 VM's sharing subnet gateway on another router, with no floatingIP-associations, (e.g. topology in Fig 5.3.1) then we have 2 groups-

Group 1- 4 VM's → inst_net1, inst_net1_2, inst_net2, inst_net2_2 (SNAT_IP1, router_gateway_MAC1)

Group 2- 4 VM's → inst_net3, inst_net3_2, inst_net4, inst_net4_2 (SNAT_IP2, router_gateway_MAC2)

Lets consider following specification for Group 1-

VM	IP	MAC	FloatingIP-associated
inst_net1	IP_net1	MAC_net1	Yes
inst_net1_2	IP_net1_2	MAC_net1_2	No
inst_net2	IP_net2	MAC_net2	No
inst_net2_2	IP_net2_2	MAC_net2_2	Yes

Lets consider following specification for Group 2-

VM	IP	MAC	FloatingIP-associated
inst_net3	IP_net3	MAC_net3	No
inst_net3_2	IP_net3_2	MAC_net3_2	Yes
inst_net4	IP_net4	MAC_net4	Yes
inst_net4_2	IP_net4_2	MAC_net4_2	No

So for destination IP address as 9.121.62.80 (lets say floatingIP for inst_net3_2)

Only one amongst the non floatingIP-associated VM's in Group 2- inst_net3 or inst_net4_2 can Ping 9.121.62.80 at a time because they share common router gateway MAC, and,

Only one of the non floatingIP-associated VM's in Group 1- inst_net1_2 and inst_net2 can Ping 9.121.62.80 at a time because they share common router gateway MAC, but

2 different non floatingIP-associated VM's within Group 2 – inst_net3 can ping 9.121.62.80 and int_net4_2 can ping 9.121.62.77 (floatingIP associated with int67_net1) at the same time

Same holds for other Group 1, also,

2 different non floatingIP-associated VM's of different groups – inst_net2 and int_net4_2 can both ping 9.121.62.80 at the same time

Like in case of external network, POC cannot distinguish whether the SNAT IP session

session was initiated by SNATed VM or not. Thus, the user needs to take care to not initiate any session to the SNAT IP's. (as mentioned in detail earlier)

The following tables summarizes all the Use-Cases that a user can try out with the POC along with the constraints, if any

USE-CASE	**CONSTRAINTS**
Overlay across / within subnet traffic (IP / ARP)	- No constraints
IP sessions from non floatingIP-associated VM to external network / another floating-IP	- ICMP sessions for a common destination IP can be initiated by one VM at a time within a group sharing common SNAT IP - No constraints for TCP sessions until no port collision
IP sessions initiated from external network / floatingIP-associated VM to SNAT IP	- User must explicitly take care of that no IP sessions are INITIATED TO SNAT IP
IP sessions from one non floatingIP-associated VM to another using destination IP address as SNAT IP	- User must explicitly take care of that no IP sessions are INITIATED TO SNAT IP
IP sessions from floatingIP-associated VM to-and-from external netwoks (SNAT / DNAT – floatingIP)	- No constraints
IP sessions from one floatingIP-associated VM to another	- No constraints

NOTE:

The POC currently does not programatically handle multiple uplinks / external networks on a common bridge. The same can be achieved with minimal efforts by updating the ARP cache table in OVS external bridge manually (adding flows in table 20 and in table 0 for the new up-link). However, the administrator must ensure that there is no redundancy involved amongst the multiple external network's IP pool.

CHAPTER 5 – ADVANTAGES OF POC

This chapter describes the various advantages of using POC in detail

- Overlay across subnet routing using flows on OVS integration and tunnel bridge

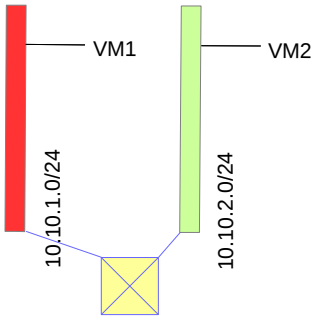
Overlay across subnet routing including MAC / IP / VLAN Tag translations are completely handled using flows defined on OVS integration and tunnel against the existing mechanism of using Linux Namespaces and Host TCP/IP Stack for the same. Thus, Neutron Router (including routing table and ARP cache) is virtualized using flows on OVS Integration Bridge, completely eliminating the dependency of Host system. (Refer Fig. 5.1) Thus there is no more need of enabling IP_forwarding on any node

- Decentralize overlay across subnet routing decision on each node

Overlay across subnet routing decisions is de-centralized on each compute node against the existing mechanism of centralized routing decisions on the network node. This reduces the load and dependency on the Network Node which acts as a single point of failure for overlay across subnet routing in the existing architecture and reduces underlay traffic. (Refer Fig. 5.1)

NOTE-

This feature has been introduced in OpenStack Community as Distributed Virtual Routing (DVR). However, the entire mechanism for the same is different from what is being used in the POC



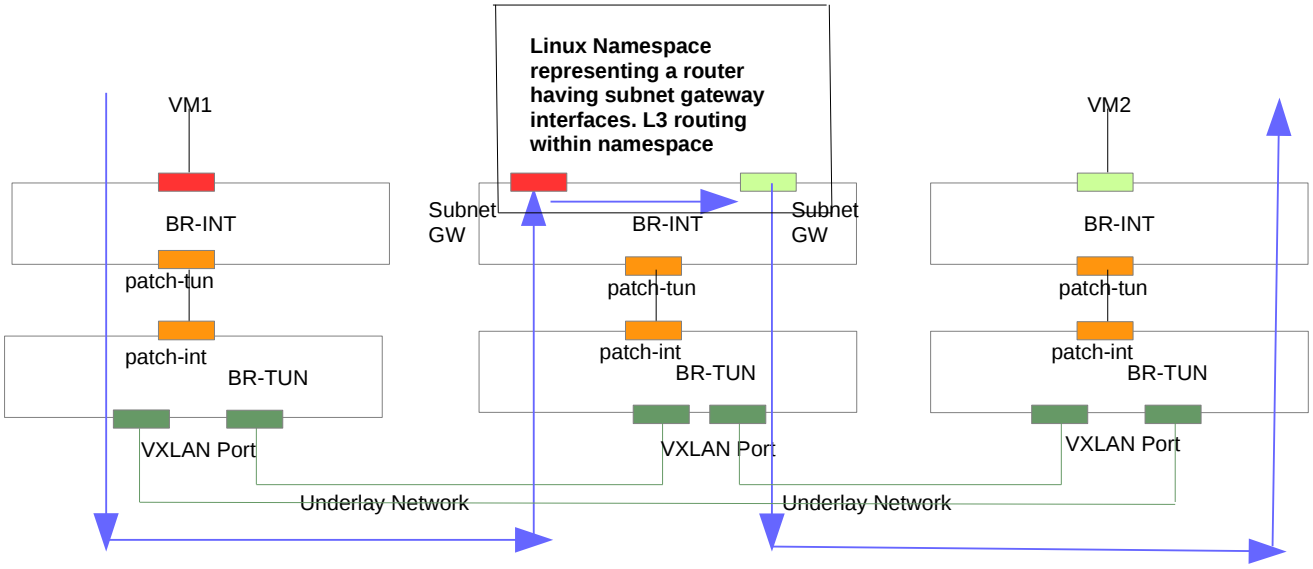


FIG 5.1.1 - ORIGINAL MECHANISM

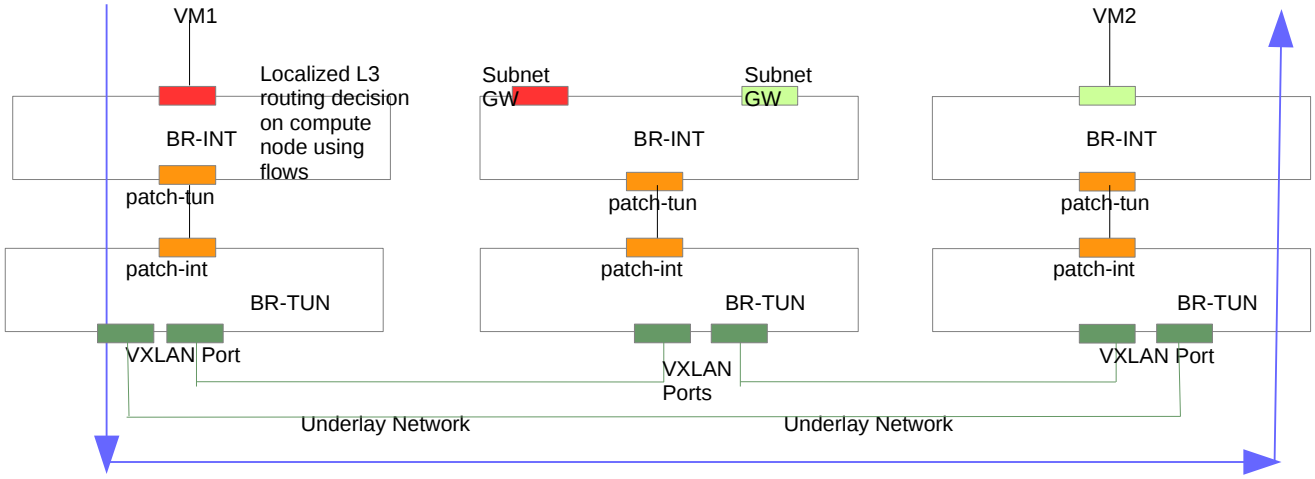


FIG 5.1.2- NEW MECHANISM

- L3 Routing / Network Address Translations (NAT) using flows on OVS external bridge

SNAT / DNAT completely using flows defined on OVS external bridge against the existing mechanism of using Linux Namespaces, iptables rule, Host TCP/IP Stack and conntrack utility. Thus, there is no dependency on Host TCP/IP stack for NATing and maintaining sessions for external network. For the POC, we use port-forwarding for SNAT/DNAT purpose. It thus maintains, L4 sessions on OVS external for TCP/UDP packets. ICMP packets can be also SNATed using flows defined on external bridge, however, flows currently provide access to only ICMP type and ICMP code which are not enough to uniquely identify each

session. Thus, unambiguous support for SNAT of ICMP packets can be supported in future. (Refer Fig. 5.2)

However, there are restrictions imposed due to non-availability of access to ICMP identifier field by the OpenFlow Protocol mentioned in “FUNCTIONALITIES SUPPORTED” section

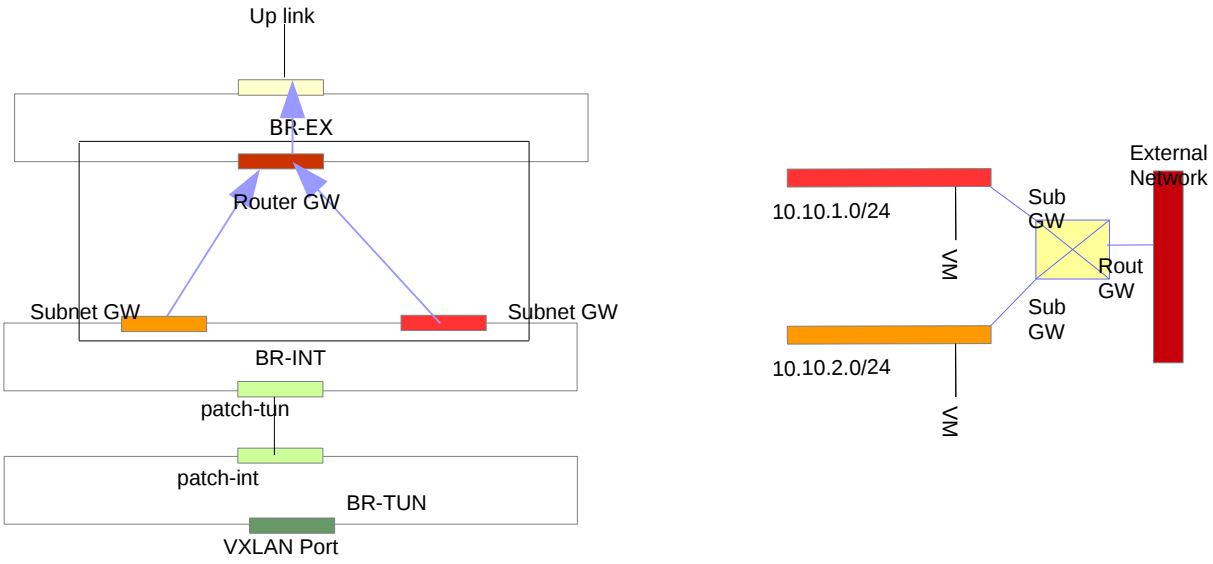


FIG 5.2.1- NETWORK NODE (OLD MECHANISM)

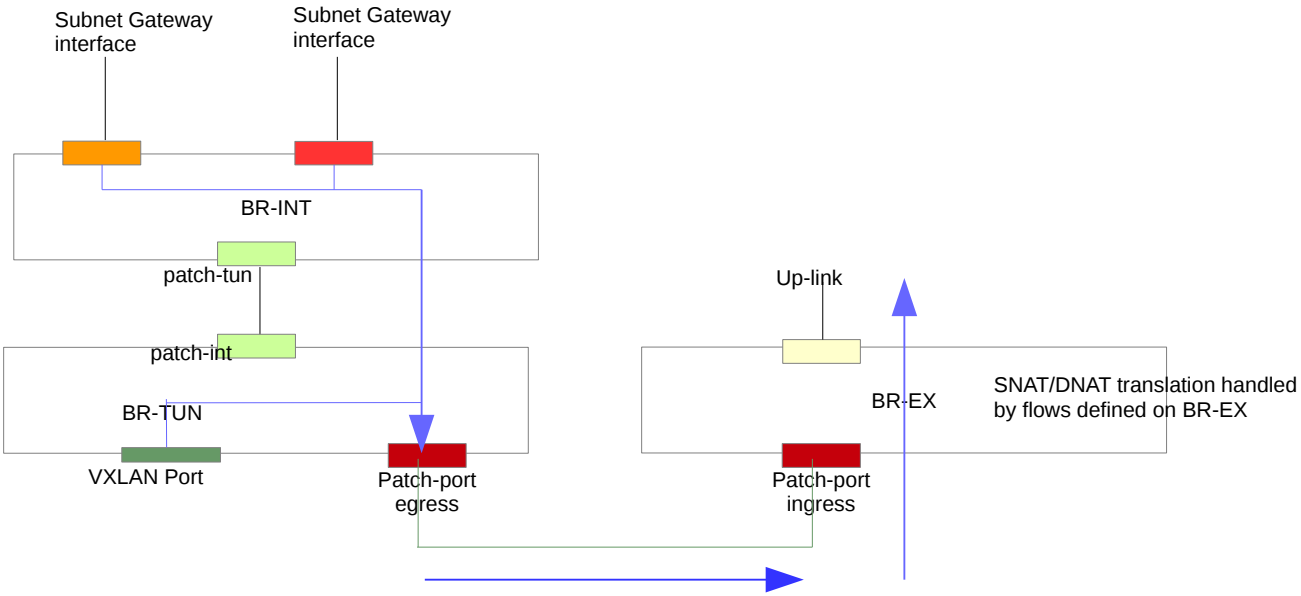


FIG 5.2.2- NETWORK NODE (NEW MECHANISM)

- Single instance of the new agent for multiple external networks

A single instance of the new agent developed as a part of POC handles SNAT/DNAT for multiple external networks against the existing
mechanism (OpenStack HAVANA) of using one neutron-l3-agent per external network on the network node. The same agent can be
scaled to use multiple external networks on same OVS external bridge or one external network per OVS external bridge. The existing
setup uses one OVS external bridge per external network. (Refer Fig. 5.3)

The POC currently can uses one OVS external bridge only and can have multiple uplinks mapped to it. POC can be scaled / modified to distribute up-links on multiple external bridges and handle them separately using single instance of the common agent.

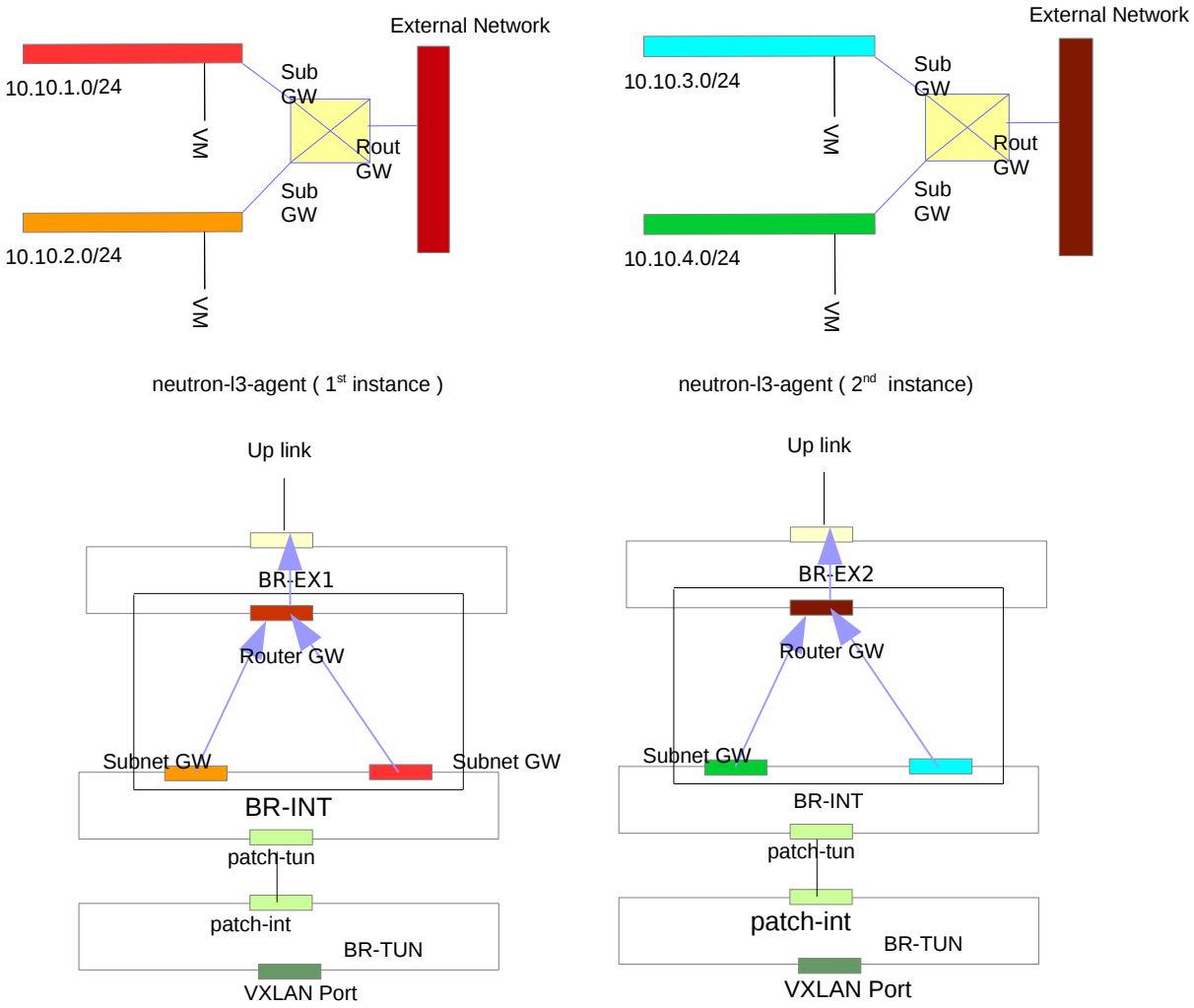


FIG 5.3.1- NETWORK NODE (OLD MECHANISM)

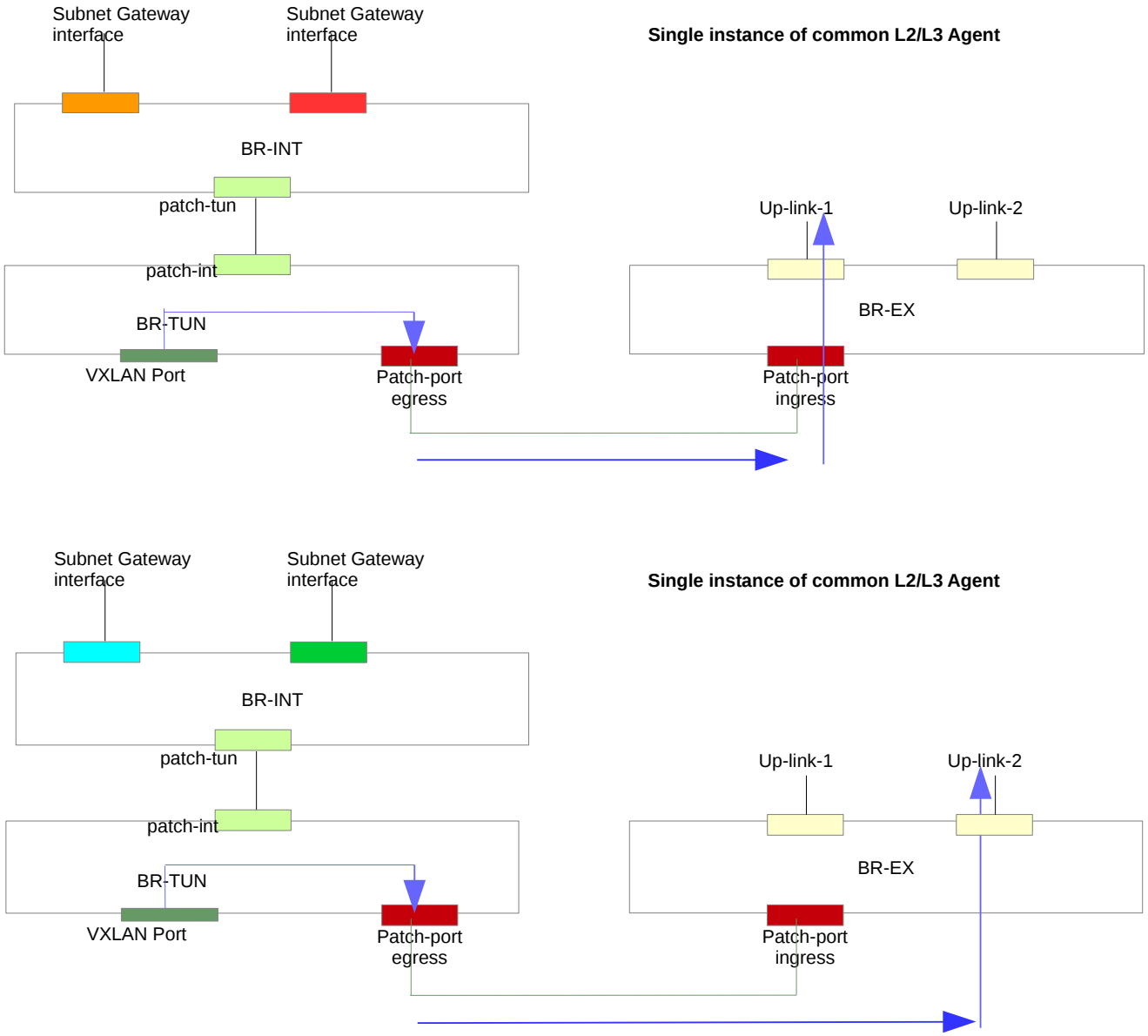


FIG 5.3.2- NETWORK NODE (NEW MECHANISM)

- L2 Population and ARP Responder Mechanism

POC makes use of L2 population and ARP responder mechanism introduced in OpenStack Havana. L2 population reduces unnecessary broadcasts in the underlay network by reducing the L2 broadcasting domains amongst the VXLAN Tunnel Ports.

ARP Responder mechanism is used on each compute node to reduce the flooding of ARP Packets in the underlay network. Thus, ARP replies are generated on each host via flows defined on the OVS Tunnel. (Refer Fig. 5.4)

ARP Responder mechanism is also introduced on the OVS external bridge for handling

ARP Request from underlay network for SNAT / DNAT ports on Neutron Router. Thus OVS external bridge generates ARP Replies for the underlay network completely via flows.

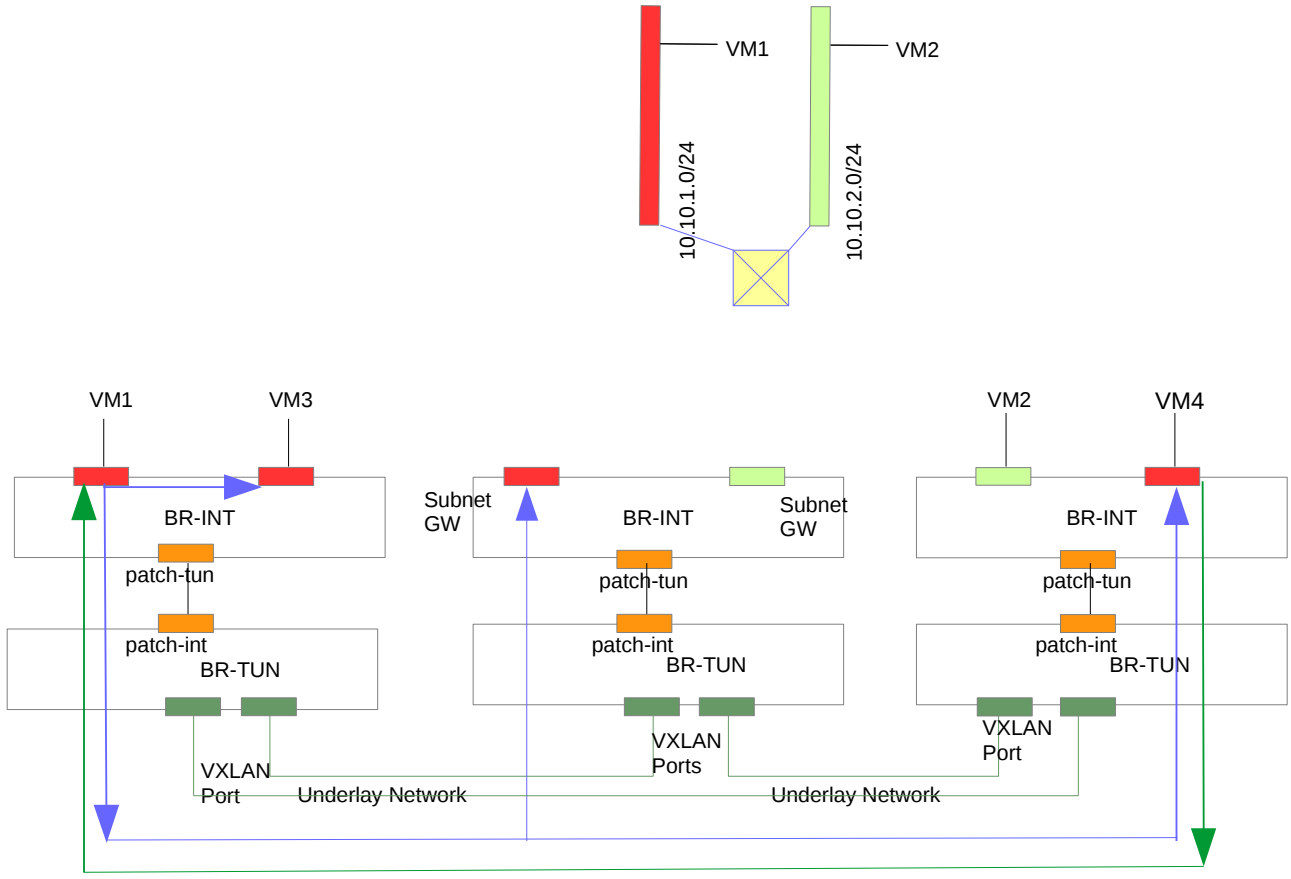


FIG 5.4.1- OLD MECHANISM

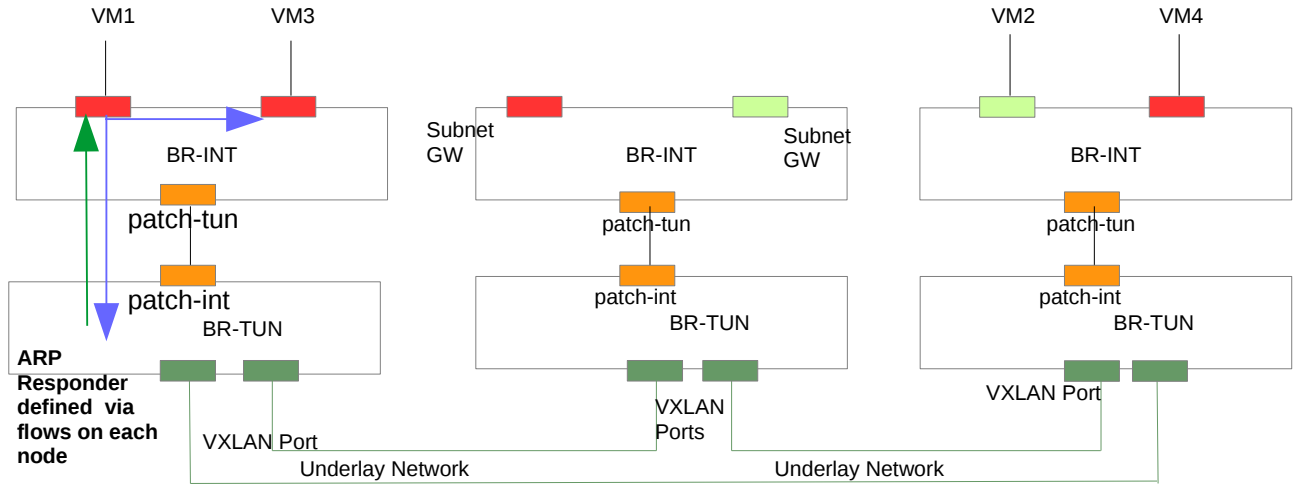


FIG 5.4.2- NEW MECHANISM

- Routing amongst VM's on networks connected by different virtual routers

The POC supports routing amongst different routers in the overlay, with certain limitations constrained by OpenFlow protocol for ICMP sessions mentioned in "FUNCTIONALITIES SUPPORTED" section

- L2 unicast / broadcast using flows defined on OVS Integration and Tunnel Bridge

Flows are defined on OVS integration Bridge for completely handling L2 unicast / broadcast and VLAN Tagging / Untagging against the existing mechanism of using OVS integration as a normal L2 Switch (NORMAL action) and relying on OVSDB. Transmission of packets across host is handled by flows defined on OVS Tunnel.

- Port Optimization / MAC Optimization

The POC does port optimization with respect to router ports (subnet gateway, router gateway ports). Since all the routing decisions are now handled via flows on OVS bridges, the POC does not actually need to create any port on OVS integration for subnet gateways and on OVS external for router gateways. Thus, achieving port optimization in the process.

The Distributed Virtual Router (DVR) functionality introduced in OpenStack Juno requires one MAC address per compute node. The POC requires no such additional MAC for the purpose. Also, the OpenStack Setup currently uses one MAC per subnet gateway interface. The POC can be easily modified to use one global MAC throughout for multiple subnet gateways and still unambiguously handle overlay across subnet and overlay to external routing. Thus, providing MAC optimization in the process

- Support for Multi-Tenant Environment

The POC is fully compatible with the multi-tenant environment

- MAC repetition across networks

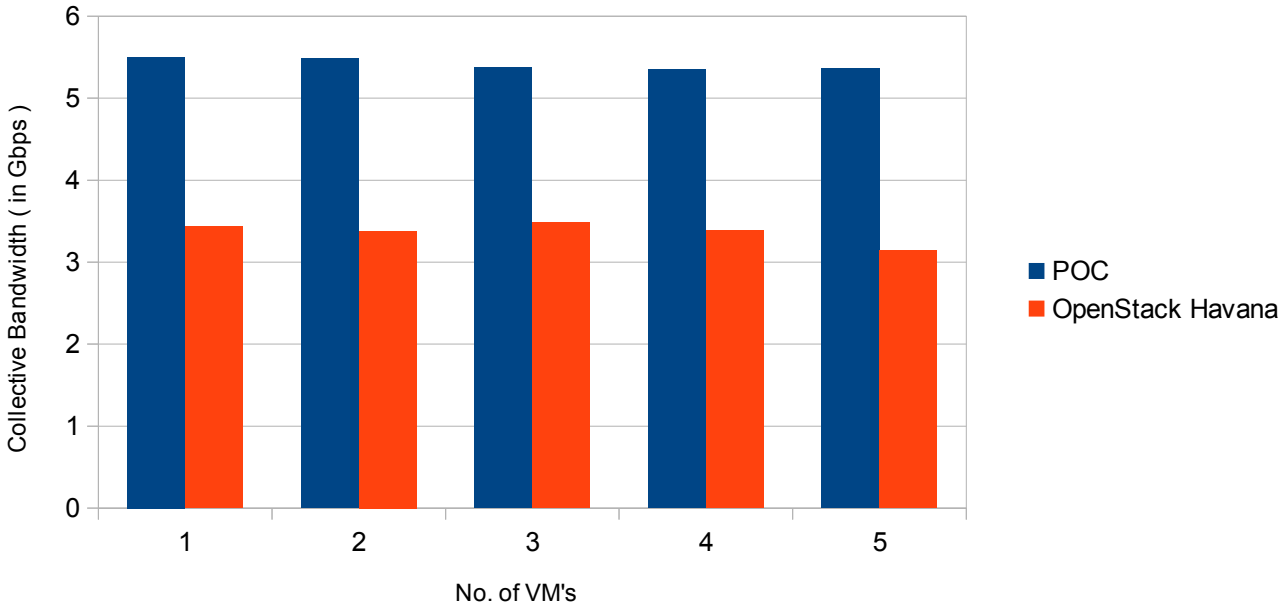
The POC supports MAC Address repetition across networks but not within a network

CHAPTER 6 - RESULTS AND DISCUSSION

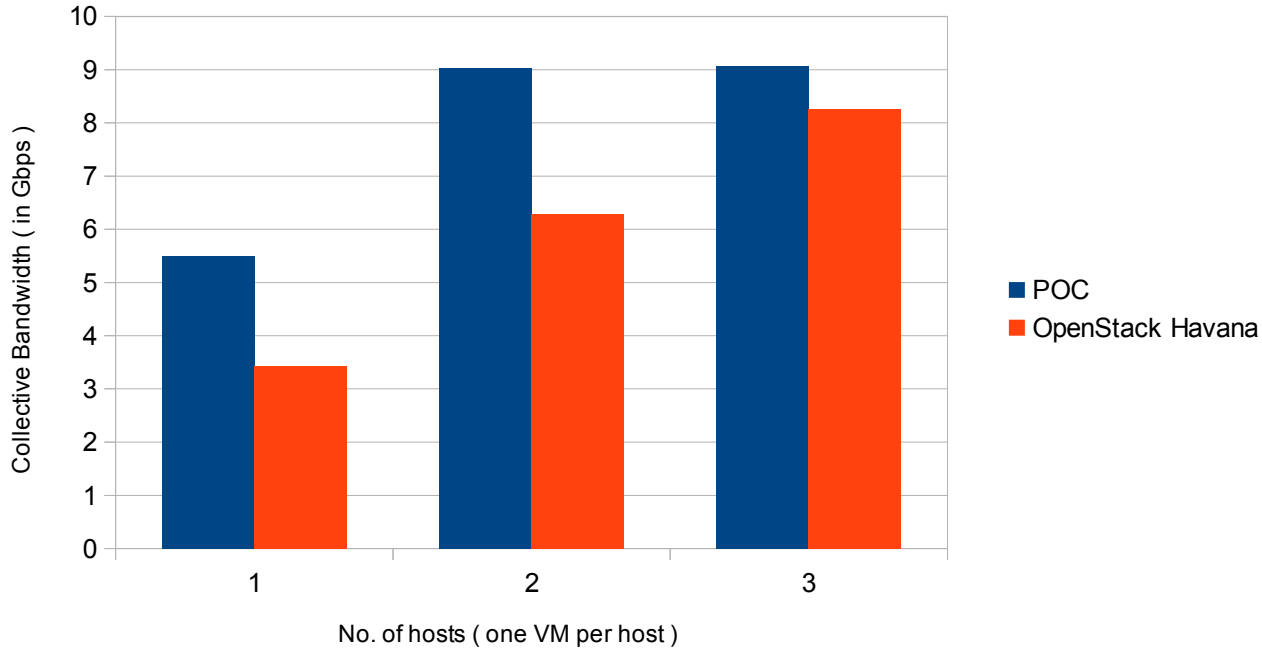
SECTION 1 – PERFORMANCE ANALYSIS

This section presents the performance analysis of the POC against the existing OpenStack Havana Setup for various use cases. For the performance analysis we used a 4 node setup (1 Network-Controller and 3 Compute Nodes). All the up-links were 10 Gbps uplinks (VxLAN tunnel port on each node, external network up-link on Network Node and the up-link on the node in the external network). MTU was 1500 for all the test cases.
Following was the VM configuration used- 2 VCPUs, 2048 MB RAM, 20 GB HDD, CentOS

Use Case 1 – SNAT for single host
For this use case, we deployed multiple VM's on same compute node and checked the collective bandwidth for various VM's on a single node for SNAT

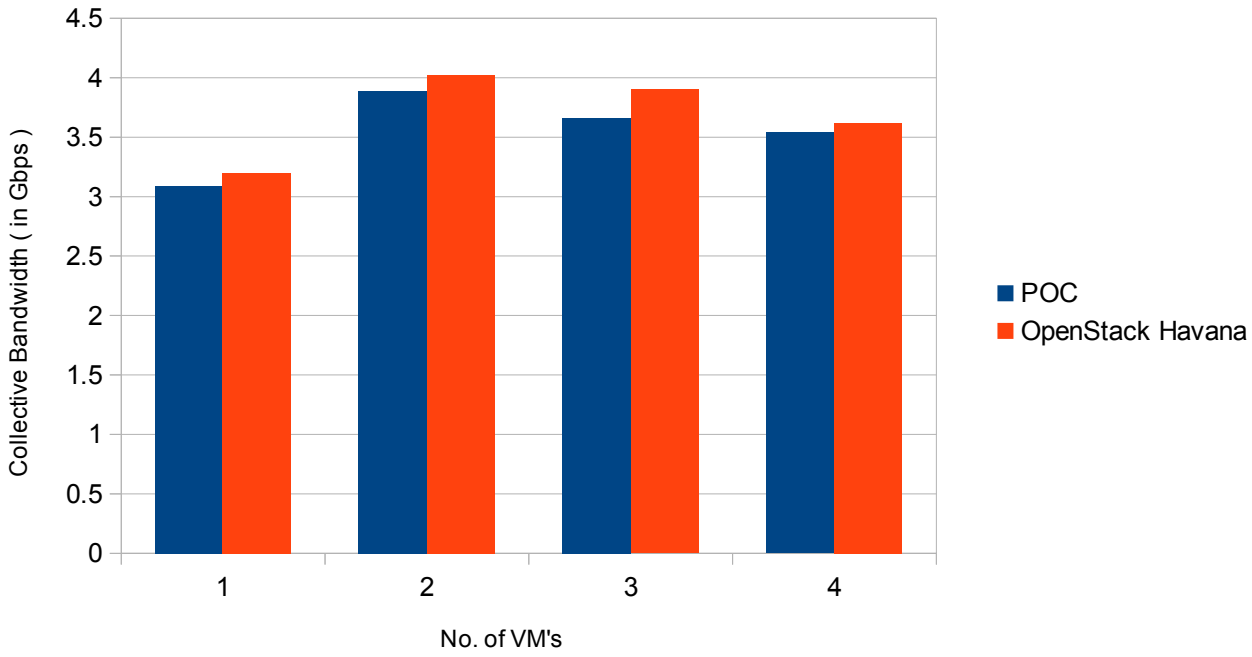


Use case 2- SNAT for Multiple Hosts
For this use case, we deployed multiple VM's on multiple compute nodes and checked the collective bandwidth for various VM's on
multiple hosts for SNAT



Use Case 3 – DNAT for single host

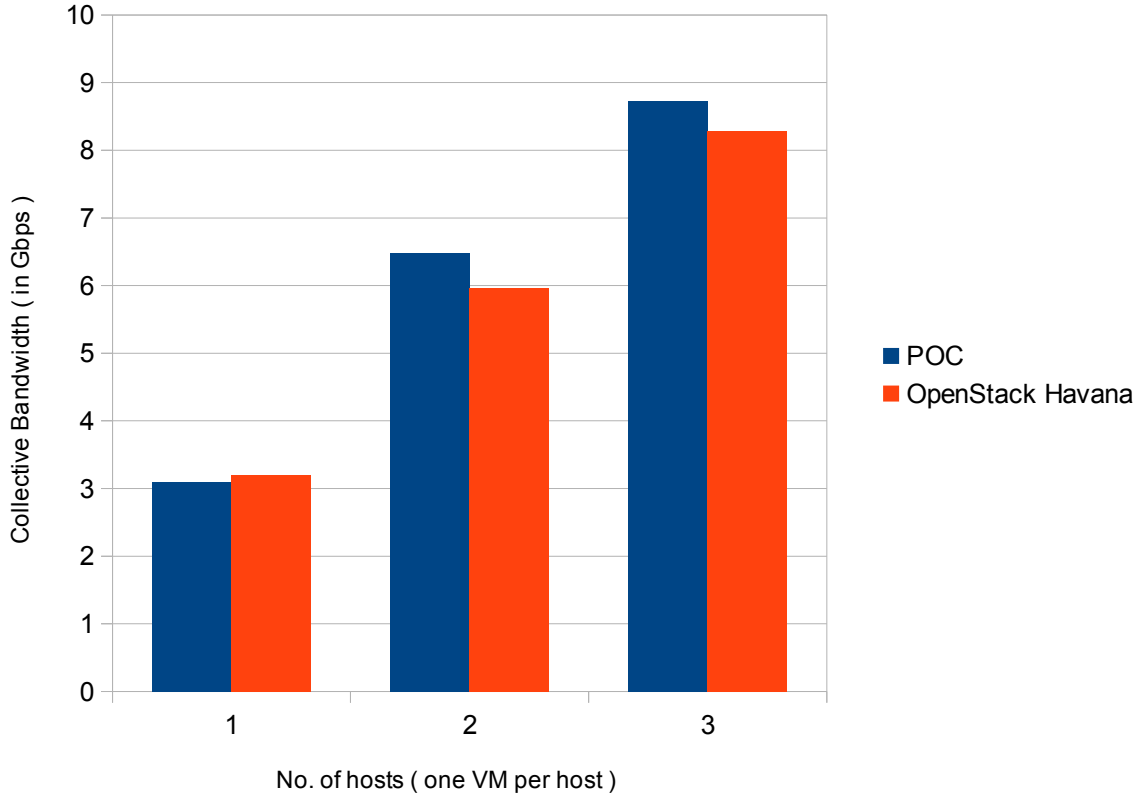
For this use case, we deployed multiple VM's on same compute node and checked the collective bandwidth for various VM's on a single host for DNAT



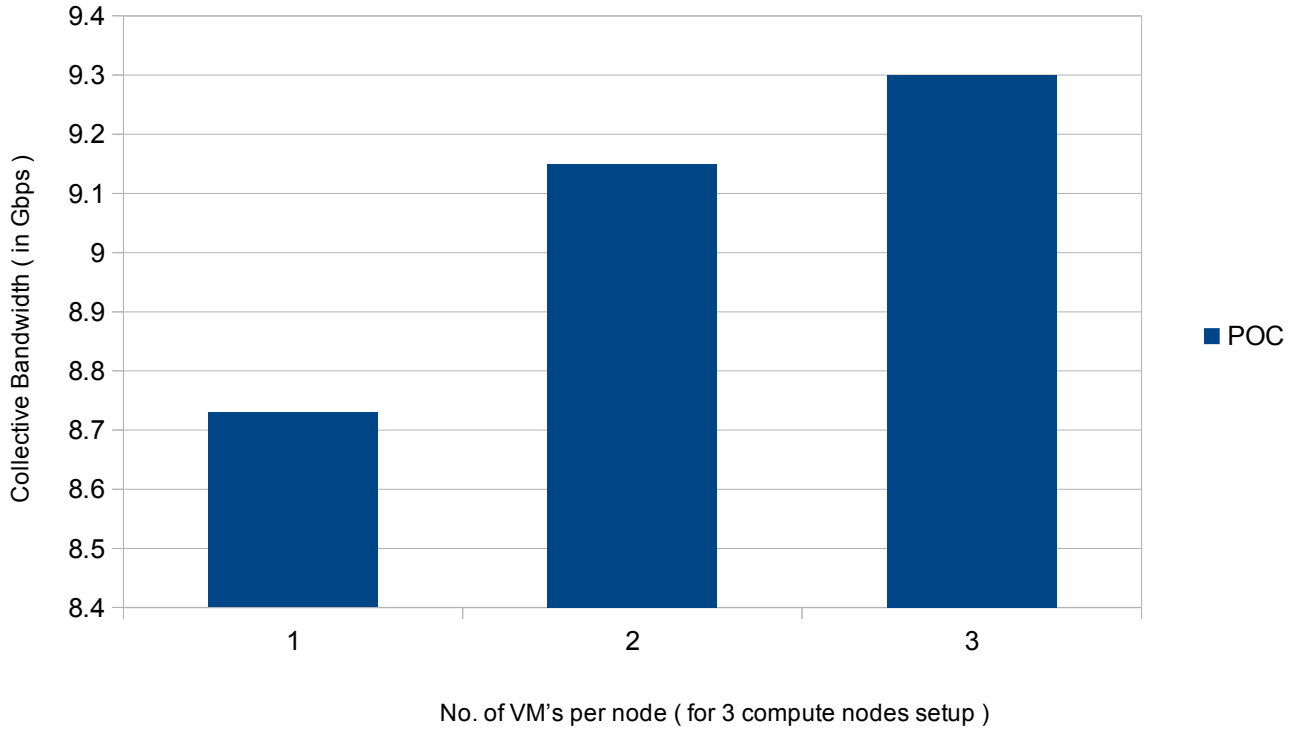
Use case 4- DNAT for Multiple Hosts

For this use case, we deployed multiple VM's on multiple compute nodes and checked the

collective bandwidth for various VM's on multiple hosts for DNAT



Use Case 5- DNAT for Multiple Hosts (for POC only)



SECTION 2 – FUTURE WORKS

- SNAT support for ICMP packets (Ping)

Flows defined by OpenFlow Protocol currently support access to only ICMP type and code and not ICMP Identifier field, and thus, cannot be used for unambiguous SNATing (SNAT requires identifier field). Thus introducing access to ICMP identifier by the flows can help in achieving SNAT and maintain each unique SNAT session and overcome the existing shortcomings relating to ICMP SNATing.

- Port Mapping instead of Port Forwarding for ICMP / IP sessions from overlay to underlay

The POC currently does a direct port forwarding leading to chances of port collisions. This shortcoming can be solved by sending the first packet to the agent and doing Port Mapping and adding the flows on OVS external bridge accordingly for subsequent packets thus avoiding any port collision.

- Support for ICMP Error MessagesFIG 22.2- NEW MECHANISM

The POC currently does not provision for any ICMP Error Messages.

- ICMP packets (Ping) support for subnet gateway and router gateway (SNAT) interfaces on router

The POC currently does not support ICMP Packets (Ping) for router interfaces. These can be supported by forwarding the ICMP requests to the new agent and provisioning the agent to generate the ICMP replies. Alternative way could be to develop ICMP Echo reply mechanism in flows on Open vSwitch just like ARP Responder mechanism

- Support multiple external networks / uplinks on single OVS external bridge programatically

Currently the user needs to add flows manually to support multiple up-links on a single OVS external bridge. The agent in POC currently programatically adds flows for one uplink on the bridge. This process can be automated by the agent for multiple uplinks

- Implement OpenStack Nova Security Groups using flows in OVS integration bridge

OpenStack Nova Security groups are currently implemented using iptables which are not

compatible with OVS bridge. Because of this the VM's are not directly mapped on to the OVS Integration Bridge and have an intermediate Linux Bridge. Security Groups can be implemented using flows on OVS integration Bridge allowing the VM's to be directly mapped on to the OVS Integration Bridge. (refer Fig 6.2)

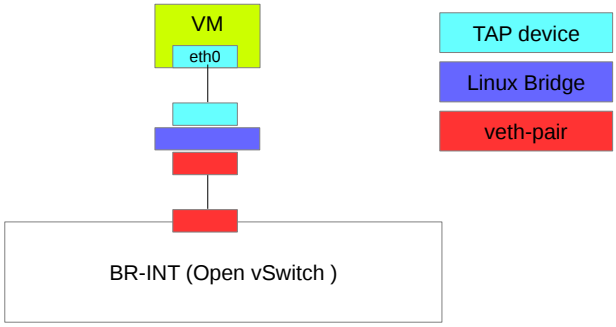


FIG 6.1.1- CURRENT SETUP

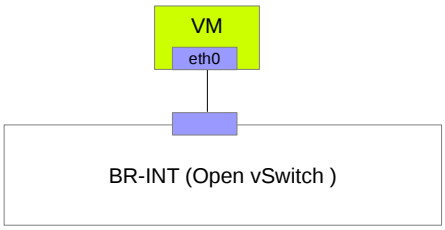


FIG 6.1.2- NEW SETUP

- Updation of ARP Cache for underlay network nodes on OVS External Bridge programmatically

Flows relating to ARP entries for nodes in underlay network needs to be added manually cuurently in the POC. Thus, mechanism can be developed to populate the ARP cache dynamically by the agent

- Updation of ARP Cache for overlay network ports on OVS Integration / Tunnel Bridge programmatically

ARP Cache entries for ports are currently static in nature in POC i.e. they do not timeout. Thus, mechanism can be developed to populate
the ARP cache dynamically by the agent.

- Deletion of flows in a programmatic manner

The POC deals with only creation and addition of flows on the OVS bridges. POC can be extended to support deletion of flows for a stand alone agent.

CHAPTER 7 – SAMPLE USE CASE

This sections presents a a sample use case for SNAT for the following topology

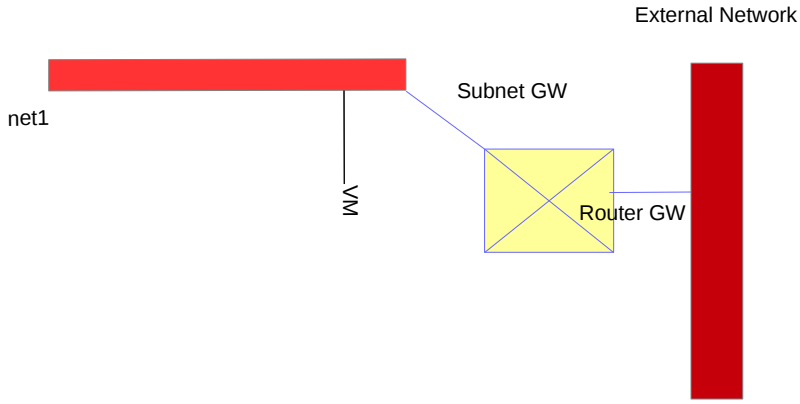


FIG 7.1.1- USE CASE DEPLOYMENT

A 2 node setup, having 1 compute and 1 network-controller node.

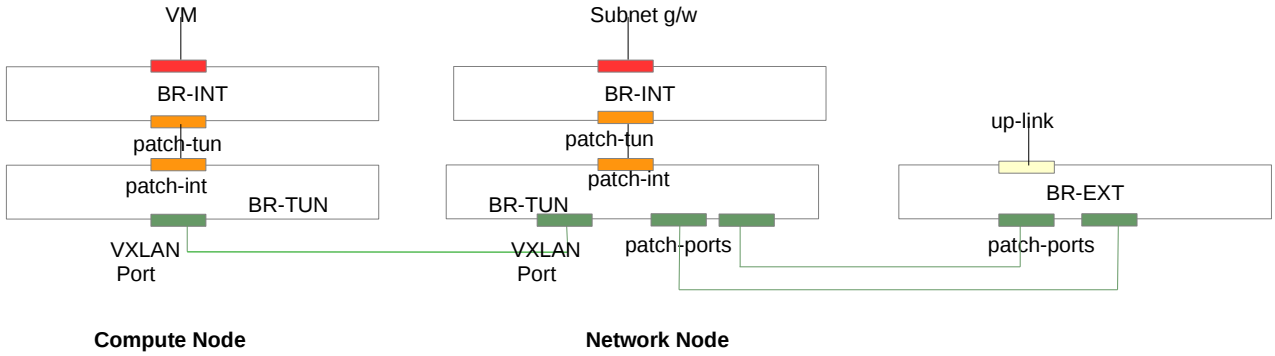


FIG 7.1.2- USE CASE DEPLOYMENT

1 network having 1 subnet

Overlay IP for VM is VM_IP and MAC address of VM is VM_MAC

Subnet gateway IP is SUBNET_GW_IP with MAC address of SUBNET_GW_MAC

Router gteway IP is ROUTER_GW_IP with MAC address ROUTER_GW_MAC

Overlay internal network VLAN tag- VLAN_INTERNAL

Underlay external network VLAN tag- VLAN_EXTERNAL

Underlay external network VXLAN tag- VXLAN_EXTERNAL

Lets say that the VM ping an external node having IP- 9.121.62.66

Following are the major sub-steps-

- ARP request for subnet gateway by VM
- ARP Reply for subnet gateway to VM by ARP Responder Mechanism
- ICMP request for 9.121.62.66 from VM on compute node sent to network node
- SNATing at Network Node for ICMP Request
- NATing to overlay IP and MAC on OVS External Bridge for ICMP reply from 9.121.62.66 received on OVS external bridge
- Forwarding to the VM on compute node

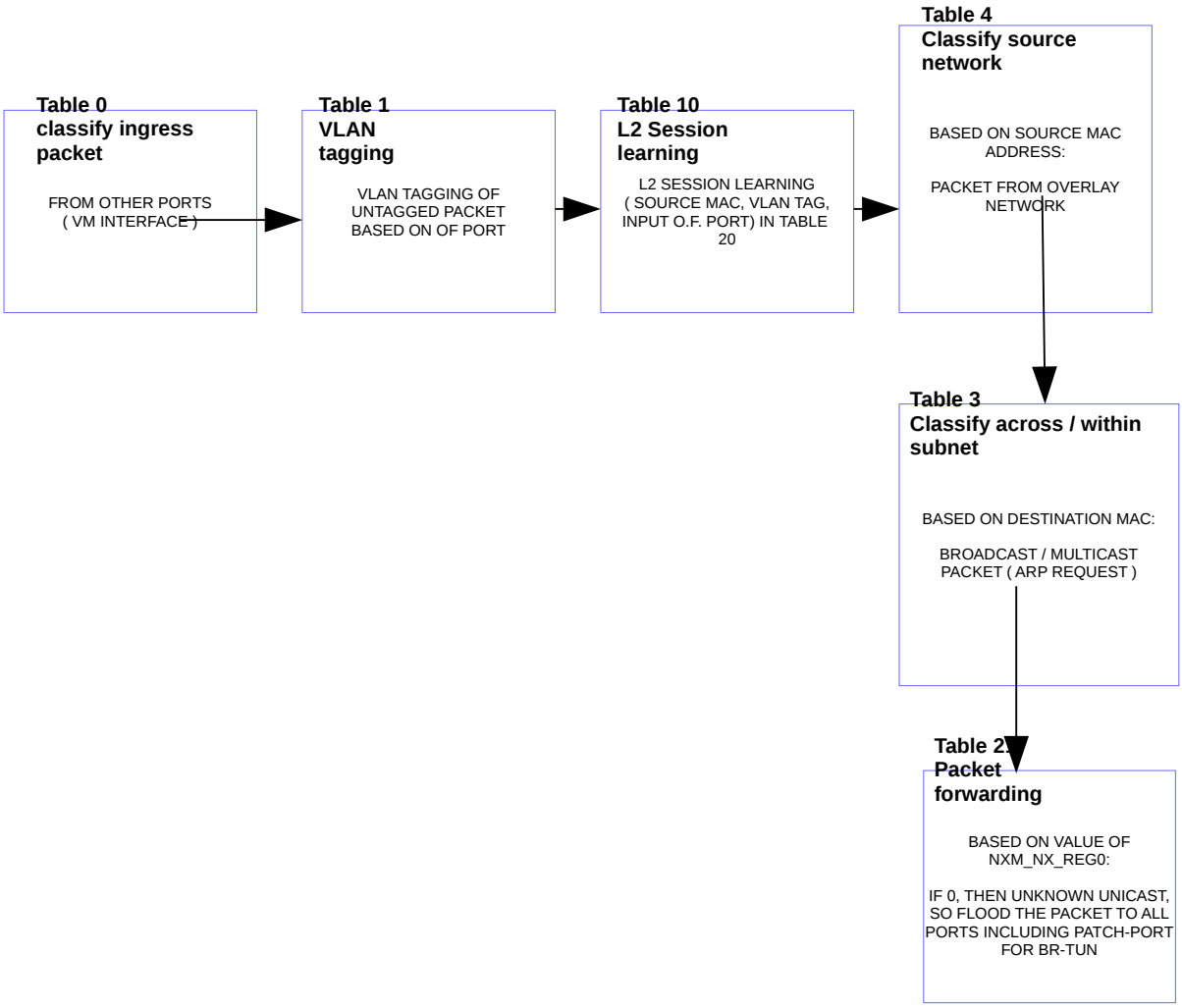
We will now discuss each step in details, shows the OpenFlow pipeline followed on each bridge and the MAC / IP / VLAN / VXLAN translations

Step 1 - ARP request for subnet gateway by VM

The process of sending any ICMP packet to 9.121.62.66 is first preceded by generating an ARP Request for subnet gateway by the VM. The VM thus generates an ARP reply for the subnet gateway which is transferred from VM patch to the OVS Tunnel bridge via patch-port after VLAN Tagging in table2

Source MAC- VM_MAC
Dest MAC- ff:ff:ff:ff:ff:ff
ARP_SPA- VM_IP,
ARP_TP A- SUBNET_GW_IP
ARP_SHA- VM_MAC
ARP_THA- 00:00:00:00:00:00
VLAN_TAG- INTERNAL_VLAN

In the process, a learned flow is created in Table 20 so that the ARP reply can be sent to the VM directly



STEP 2- ARP Reply for subnet gateway to VM by ARP Responder Mechanism

On receiving an ARP request from the OVS integration patch-port, OVS Tunnel Bridge, generates and ARP Reply for the subnet gateway in table 21

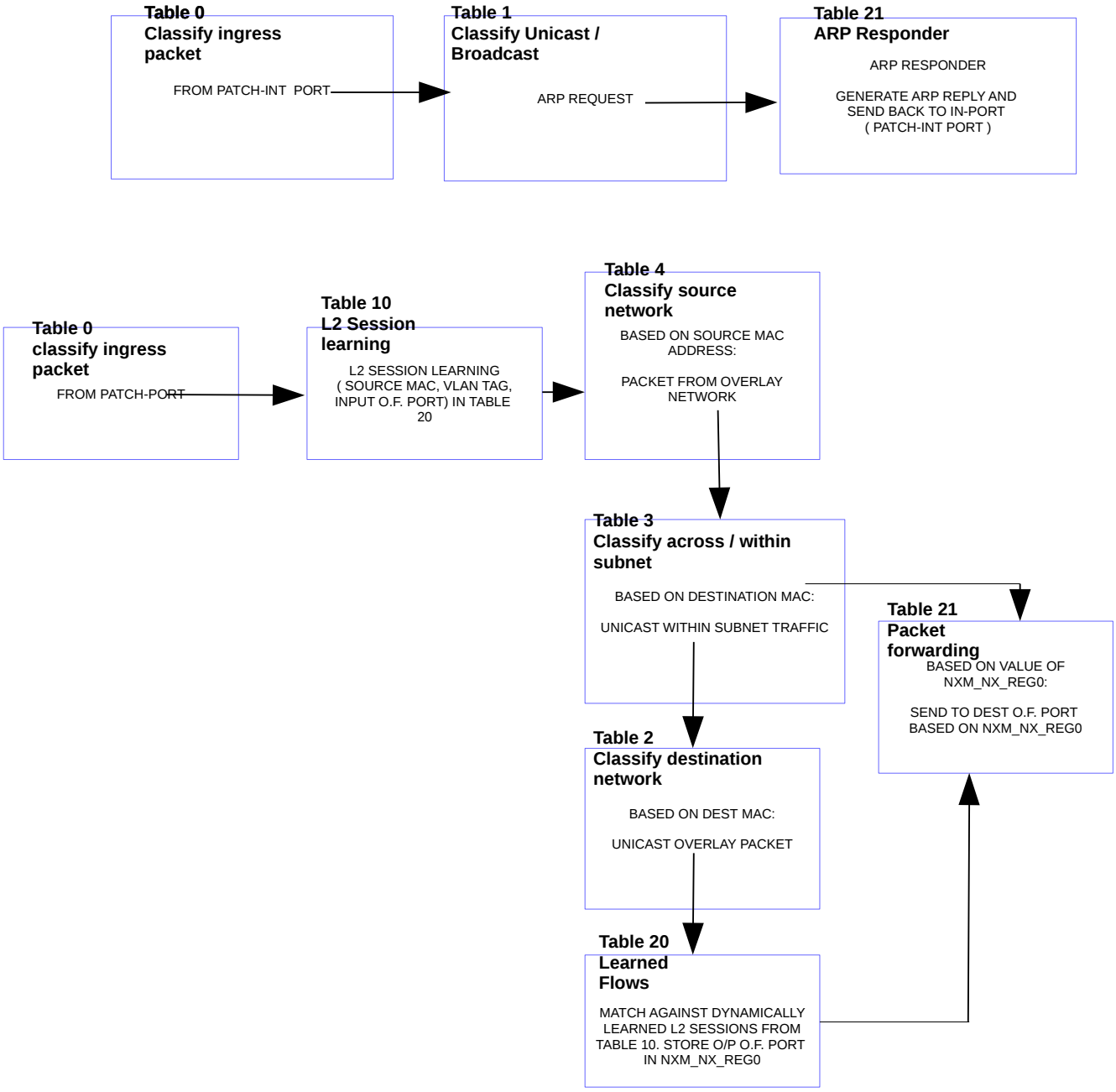
Source MAC- SUBNET_GW_MAC **Dest MAC**- VM_MAC

ARP_SPA- SUBNET_GW_IP **ARP_TP A-** VM_IP

ARP_SHA- SUBNET_GW_MAC **ARP_THA-** VM_MAC

VLAN_TAG- INTERNAL_VLAN

and sends it back to the input port (OVS integration patch-port). On receiving the ARP reply (unicast) on the patch-port in OVS integration bridge, the flows refer to the learned flow in Table 20 and forward the packet to the VM



STEP 3- ICMP request for 9.121.62.66 from VM on compute node sent to network node

After receiving the ARP reply for subnet gateway, the VM sends an ICMP echo request for 9.121.62.66.

Source MAC- VM_MAC **Dest MAC**- SUBNET_GW_MAC

Source IP- VM_IP, **Dest IP** - 9.121.62.66

The flows on OVS integration bridge handle the VLAN tagging of packet for the source network of which VM is a part of in table=1

Source MAC- VM_MAC **Dest MAC-** SUBNET_GW_MAC

Source IP- VM_IP, **Dest IP** - 9.121.62.66

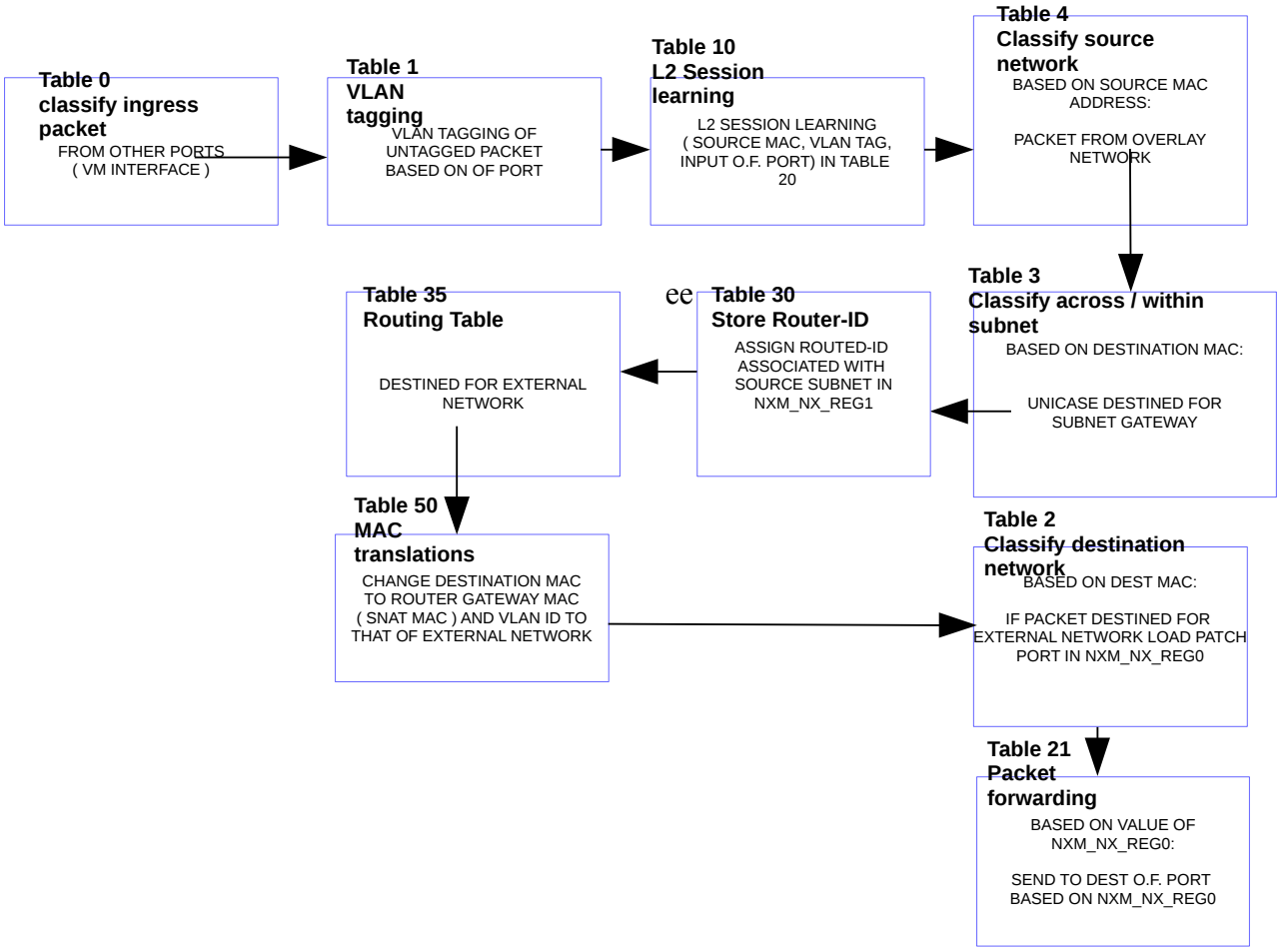
VLAN- INTERNAL_VLAN

A learning flow is created in Table 20 for the ICMP reply from 9.121.62.66. The flows then transform the destination MAC from subnet gateway MAC to router gateway MAC and change the VLAN tag to that of the external network in Table 50 and forward the packet to patch-port

Source MAC- SUBNET_GW_MAC **Dest MAC-** ROUTER_GW_MAC

Source IP- VM_IP **Dest IP-** 9.121.62.66

VLAN- EXTERNAL_VLAN



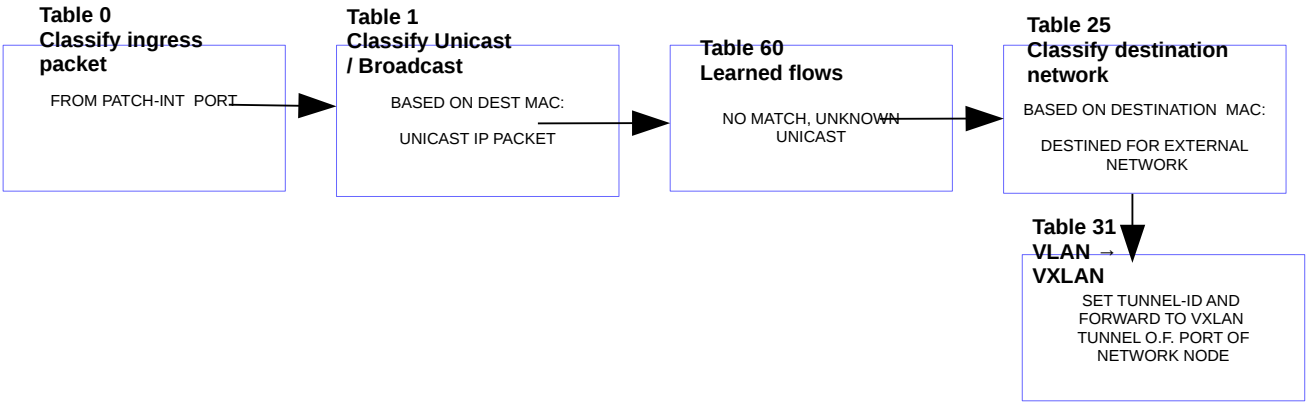
On receiving the packet from patch port in OVS tunnel bridge, the packet is forwarded to the network node directly by removing VLAN tag and loading the VXLAN ID for external network in Table 31

Source MAC- SUBNET_GW_MAC **Dest MAC-** ROUTER_GW_MAC

Source IP- VM_IP **Dest IP-** 9.121.62.66

VXLAN- EXTERNAL_VXLAN

The packet is then forwarded to the VXLAN tunnel port corresponding to the network node



STEP 4- SNATing at Network Node for ICMP Request

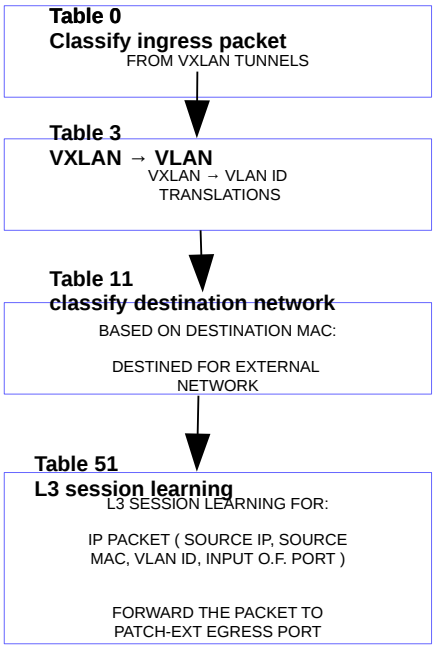
The packet is received by OVS tunnel bridge on the Network Node which handles VLAN tagging of the packet to that of the external network using VXLAN ID in table 3. A flow is created in Table 61 for the L3 session which handles ICM reply from 9.121.62.66 and the packet is sent to the OVS external bridge

(Packet after encountering table 3)

Source MAC- SUBNET_GW_MAC **Dest MAC-** ROUTER_GW_MAC

Source IP- VM_IP **Dest IP-** 9.121.62.66

VLAN- EXTERNAL_VLAN



The packet is received on the OVS external bridge. Since there no floatingIP associated with VM, pipeline for SNATIng is followed. First a flow is created table 40 for ICMP reply then VLAN Tag is stripped from the packet and NATing is done for the outgoing ICMP request in table 30.

(Packet after encountering table 30)

Source MAC- ROUTER_GW_MAC **Dest MAC-** ROUTER_GW_MAC

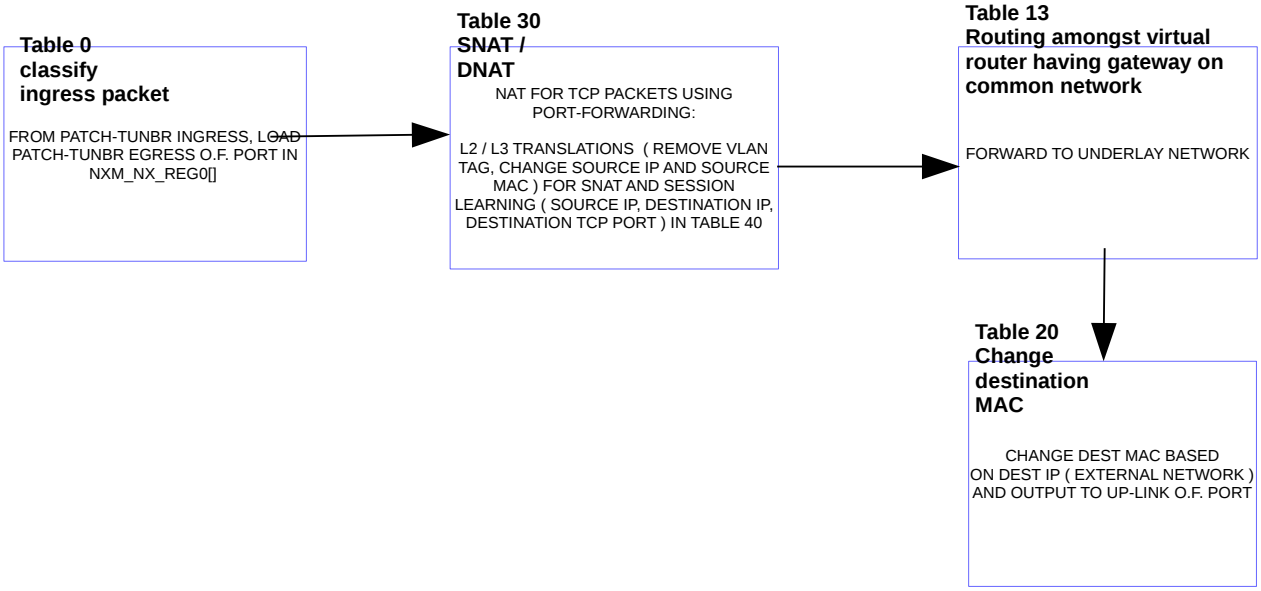
Source IP- SNAT_IP **Dest IP-** 9.121.62.66

The packet is then forwarded 9.121.62.66 by changing the destination MAC to that of 9.121.62.66 in table 20

Source MAC- ROUTER_GW_MAC **Dest MAC-** MAC_OF_9.121.62.66

Source IP- SNAT_IP **Dest IP-** 9.121.62.66

The packet is now forwarded to the up-link port



STEP 5- NATing to overlay IP and MAC on OVS External Bridge for ICMP reply from 9.121.62.66 received on OVS external bridge

Following is the IP/MAC address of the ICMP reply received on the up-link on OVS external bridge

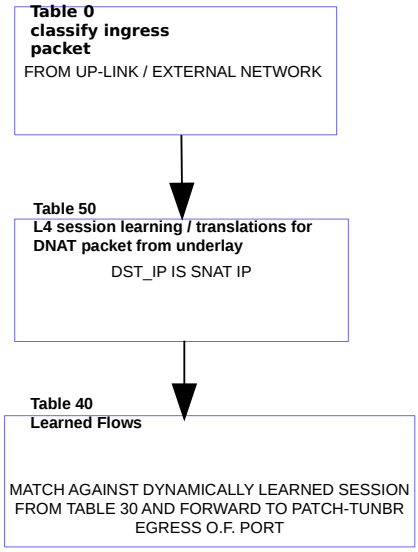
Source MAC- MAC_OF_9.121.62.66 **Dest MAC-** ROUTER_GW_MAC

Source IP- 9.121.62.66 **Dest IP-** SNAT_IP

On receiving the ICMP reply from 9.121.62.66, the learned flows in Table 40 handle the IP/MAC translation from underlay MAC/IP to overlay MAC/IP and VLAN Tagging to that of the external network.

Source MAC- ROUTER_GW_MAC **Dest MAC-** SUBNET_GW_MAC
Source IP- 9.121.62.66 **Dest IP-** VM_IP
VLAN- EXTERNAL_VLAN

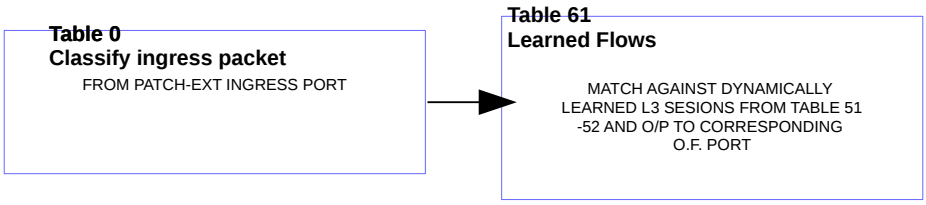
The packet is then sent to OVS Tunnel bridge



The packet on being received by OVS tunnel from OVS external bridge encounters learned flow in Table 61 for ICMP request and is sent directly to the compute node hosting the destination VM after removing VLAN Tag and allocating VXLAN ID for the external network

(Packet after encountering table 61)

Source MAC- ROUTER_GW_MAC **Dest MAC-** SUBNET_GW_MAC
Source IP- 9.121.62.66 **Dest IP-** VM_IP
VXLAN- EXTERNAL_VXLAN



STEP 6 - Forwarding to the VM on compute node

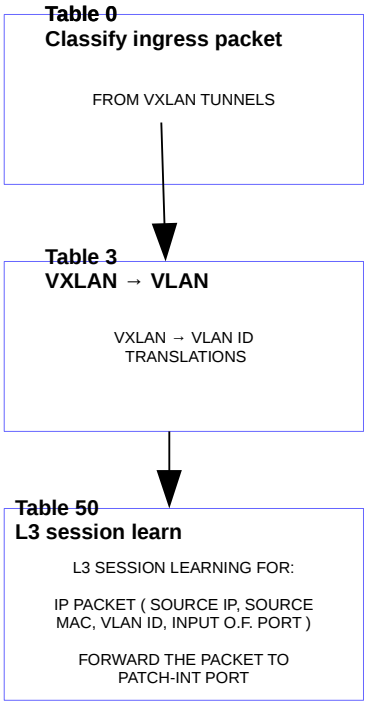
The packet is received by OVS Tunnel on the compute nodes. The packet is VLAN tagged to that of the external network in table 3

Source MAC- ROUTER_GW_MAC **Dest MAC-** SUBNET_GW_MAC

Source IP- 9.121.62.66 **Dest IP-** VM_IP

VLAN- EXTERNAL_VLAN

The packet is than forwarded to the patch port corresponding to the OVS Integration Bridge.



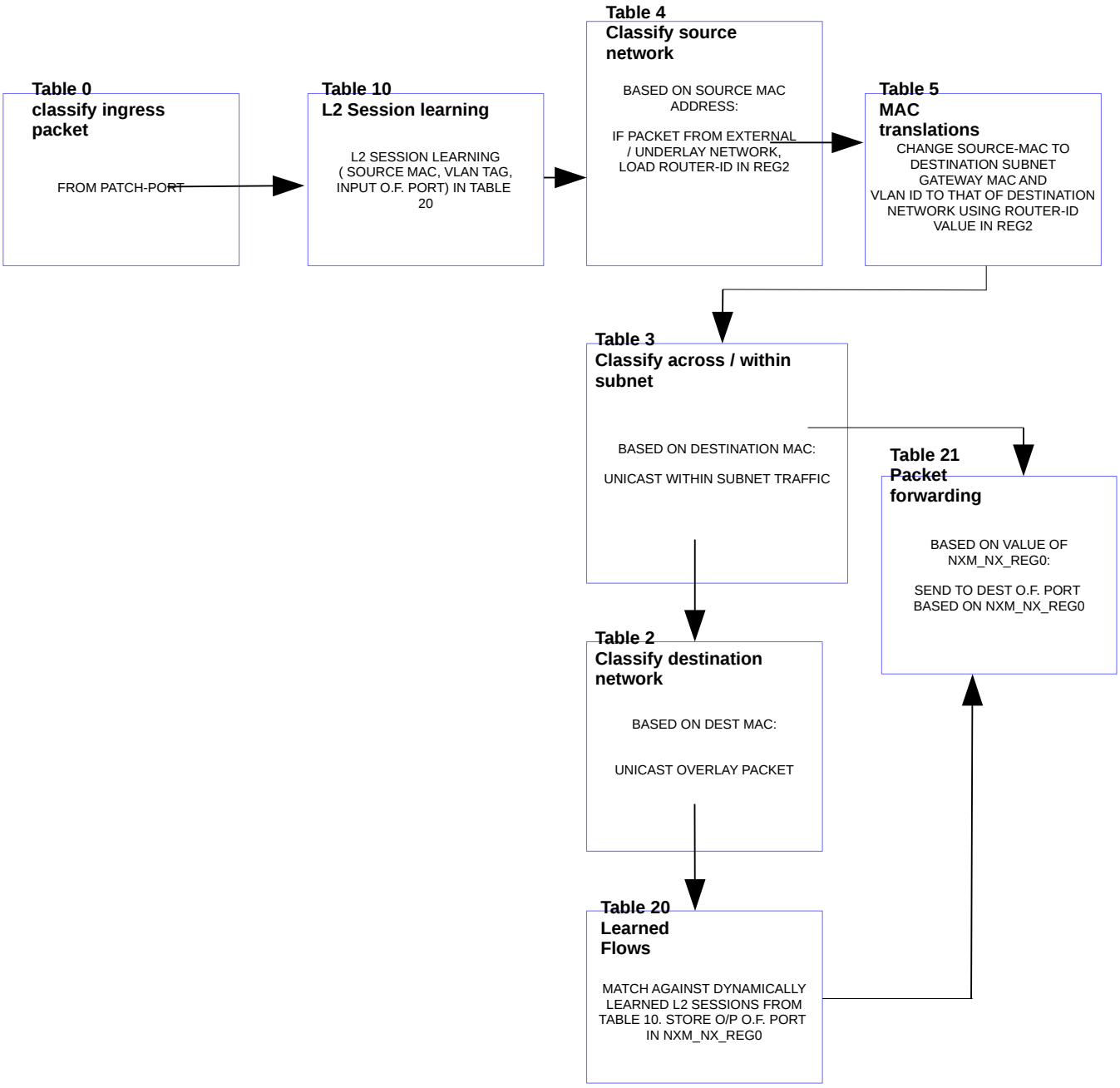
9On receiving the packet on OVS integration bridge, the packets source MAC address is changed from router gateway MAC to subnet gateway MAC in table 5, along with removing VLAN tag for external network and adding VLAN tag for the network of destination VM.

Source MAC- SUBNET_GW_MAC **Dest MAC**- VM_MAC

Source IP- 9.121.62.66 **Dest IP-** VM_IP

VLAN- INTERNAL_VLAN

The packet is then referred to table 20 which contains the flows for forwarding the packet directly
to VM.



CHAPTER 8 - SUMMARY AND CONCLUSION

The POC primarily aimed at leveraging the use of OpenFlow protocol and Open vSwitch for a virtual data center deployment. It does not aims to provide a mechanism to substitute the existing setup but rather proposes a new ideology using Software defined networking. As shown in the performance analysis, the POC has an appreciable bandwidth (reaching 9 Gbps for 10 Gbps links) and fares well against OpenStack setup for SNAT cases and multiple host DNAT and can be used to implement an IP gateway for overlay to underlay network traffic completely using flows defined by Open Flow protocol on Open vSwitch. However the increment is not so high for DNAT as compared to SNAT cases due to issues with MTU negotiation mechanism for DNAT. Although, POC provides better cumulative performance, it still has many components to be implemented, including primarily support for mutation and generation of ICMP packets for ping reply support (like ARP Responder mechanism) and generating ICMP Error messages (to fully virtualize a Neutron Routing instance). Thus, working in this direction involves development in OpenFlow / Open vSwitch for a more Software Defined Network Architecture, reducing the load on Host TCP/IP Stack and Host Kernel and retaining multi-tenancy. One other interesting application that comes out of the POC is implementation of security groups using flows instead of traditrional iptables rules, again reducing the load from Host Kernel.

CHAPTER 9 – LITERATURE CITED

- https://wiki.OpenStack.org/wiki/Neutron/ML2

- https://blueprints.launchpad.net/neutron/+spec/l2-population

- https://docs.google.com/document/d/1sUrvOQ9GIl9IWMGg3qbx2mX0DdXvMiyvCw2Lm6snaWQ/edit

- https://review.openstack.org/#/c/49227

- http://docs.openstack.org/

- http://git.openvswitch.org/cgi-bin/gitweb.cgi?=openvswitch;a=blob_plain;f=tutorial/Tutorial;hb=HEAD

- https://wiki.openstack.org/wiki/Neutron/DVR_L2_Agent

- https://wiki.openstack.org/wiki/Neutron/DVR

- https://blueprints.launchpad.net/neutron/+spec/neutron-ovs-dvr

- https://wiki.openstack.org/wiki/Neutron/DVR/HowTo

- https://docs.google.com/document/d/1iXMAyVMf42FTahExmGdYNGOBFyeA4e74sAO3pvr_RjA/edit

- http://www.borgcube.com/blogs/2011/11/vxlan-primer-part-1/

- https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

- https://www.sdncentral.com/what-is-openflow/

- http://blog.scottlowe.org/2013/04/30/onnetwork-virtualization-and-sdn/

- http://www.businessnewsdaily.com/5791-virtualization-vs-cloud-computing.html

- http://www.ibm.com/developerworks/cloud/library/cl-hypervisorcompare/

- http://ilearnstack.com/2013/05/20/introduction-tocloud-computing-for-newbies/

- http://blog.scottlowe.org/2013/09/09/namespaces-vlans-open-vswitch-and-gre-tunnels/

- http://www.opencloudblog.com/

- http://www.jedelman.com/home/open-vswitch-101

CHAPTER 10 – PUBLICATION

The PBI was a Research and Development (R&D) profile in IBM India Cloud Networking Labs. The entire 6 months research work done has been published on IBM's wiki.

The following have been **published on IBM Distributed Overlay Virtual Ethernet (DOVE) Wiki** for **IBM Cloud Networking community across globe**-

- Research Paper on POC containing detailed documentation
- CookBook on the POC along with related resources
- Presentation giving overview on the POC

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my profound gratitude and deep regard to my mentor and teacher Mr. Rachappa B Goni, Advisory Engineer, Cloud Networking, IBM India Pvt Ltd. for his exemplary guidance, valuable feedback and constant encouragement throughout the duration of the internship. His guidance, well-planned research assignments and planning the entire internship work plan in a pure R&D style kept me working to make this project in a much better way and gave me an opportunity to develop the POC from the scratch in my own innovative way covering both depth and width in terms of knowledge and experience.

I would like to extend my sincere gratitude and deep regard to Mr. Prashanth K Nageshappa, Senior Engineer / Master Inventor, Cloud Networking, IBM India Pvt Ltd for taking interest in my work, keeping an update on my work in regular intervals and for helping me build a solid foundation on Cloud Computing / Virtualization technologies prior to starting work on POC. His expertise in virtualization technologies always motivated and encouraged to me to explore and learn more from this internship.

I would also like to thank and extend my hearty gratitude to Mr. Thomas Domin, Senior Manager, Cloud Networking, IBM India Pvt Ltd and Mr. Gowtham Narasimhaiah, Manager, Cloud Networking, IBM India Pvt Ltd for appreciating the entire work done during the 6 months internship. I am particularly thankful to them for introducing me to IBM Business Model and Business Value. I really appreciate your efforts for exposing me to the non-technical side of industry and how an organization works and organizational structure from business point of view.

I would like to extend my hearty gratitude to Mr. Vaidyanathan Gopalakrishnan, Operations Manager, Cloud Networking, IBM India Pvt Ltd for his constant support, help in organizational formalities and guidance right from the first day till the very end.

I would also like to thank Mr. Nirmalanand Jebakumar, Staff Software Engineer, Cloud Networking, IBM India Pvt Ltd, Mr. Pramod D, Software Engineer, Cloud Networking, IBM India Pvt Ltd and Mr. Ather Qadri, Software Engineer, Cloud Networking, IBM India Pvt Ltd for helping me during the internship and entertaining all my queries throughout.

At last I would like to thank and extend my deep regard for my internal supervisor, Mr. Saket Saurav, Research Engineer, IIITDM, Jabalpur, without whose constant support, guidance and motivation, both during and prior to internship, I would have not been able to get such an innovative and unique opportunity to learn and develop at IBM India Cloud Networking Labs.