Providing Layer 2 Data center Interconnection using Overlay technology – VxLAN

Final Report

Software Defined Networking CSC 591-017

Table of Contents

Abbreviations

VM	Virtual Machine
VxLAN	Virtual Extensible LAN
DC	Data Center
VNI	Virtual Network Identifier
VTEP	Virtual Tunnel Endpoint
L3	Layer 3
L2	Layer 2

1. Problem statement

Title: Providing Layer 2 Data center Interconnection using Overlay technology – VxLAN

Team:

- Ananthram Eklaspuram Lakshmanasamy,
- Shishir Bharadwaj Nagendra
- Madhav Bhatt

Guide:

- Professor Dr. Rudra Dutta

URL: https://sites.google.com/view/sdn-vxlan
Video Demo: https://goo.gl/YfPUAq

1.1. Description of problem:

Problem Statement:

Some use cases that necessitate the need for L2 interconnect between Data Centers and a tunneling technology like VxLAN are:

I. A Cloud Service Provider (like Amazon or Google) intending to give network-as-a service, so that one tenant's resources (like compute nodes and storage) can be spread across multiple sites.
II. A company has multiple data center sites, and wants to make use of the Internet to not be able to send encrypted from a VM in one site to a VM in another, but also be able to perform VM Migrations from one site to another. With Layer 2 interconnect such a migration can be done dynamically without changing IP addresses or Default gateways.
III. A service provider who was previously achieved L2-layer isolation using VLANs, but now needs to scale beyond 4096 users.
IV. Reusing IP addresses between different tenants

The aim of this project is to provide Layer-2 connectivity between physically separated data center sites making use of an underlay Layer-3 network, like the Internet. To achieve this objective, we employ L2-over-L3 tunneling technology, namely VxLAN.

We propose to implement two mechanisms to achieve this objective -

1. A traditional "non-SDN" approach where the gateway OpenvSwitch at each data center site will provide the VTEP Functionality by means of preset rules, individually set at each VTEP. Thus, the functionality will be distributed to all the sites in the network

2. An SDN approach, wherein a centralized controller will set the flow rules at each gateway OpenvSwitch to provide VTEP functionality. These flow rules will be based on the instructions given by a VxLAN service application running on the controller.

2. Components

2.2 Platform(s) for project

Testbed: ExoGeni
Switch and Router: OpenvSwitch
OpenFlow Controller: RYU controller programmable in python
Virtual Machine: Ubuntu 14.04

2.3 Area(s) for project

- Data Center Interconnect
- Layer 2 Tunneling
- OpenFlow
- Software Defined Networking

2.4 Major components decomposition

1. **Tenants:** A tenant is basically a customer of the L2 service provided. It is an abstract component, represented by its host VMs

2. **Datacenter**
   In our project, we are modelling the data center using Linux machines. The Linux machine consists of

   - **Virtual Machines (VM)**: These VMs are created using an open source virtualization application called Virtual Box. Each VM belongs to a Tenant at a data center site.

   - **VTEP:** VTEPs are VxLAN tunnel endpoints, we are realizing VTEP using an OpenvSwitch bridge in the Linux machine. The VTEPs can communicate with the SDN controller through a OpenFlow connection

   - **Tenant ID:** Inside the datacenter, tenant identification can be done by a number of ways discussed in a section below. We plan to do it based on Ports.

   - **OpenvSwitch (OvS): "**Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (e.g. NetFlow, sFlow,

IPFIX, RSPAN, CLI, LACP, 802.1ag)". It supports an OpenFlow interface which we are using to program (set flows) the OvS.(Quoting from [1])

3. SDN Controller (Used only in the centralized approach)

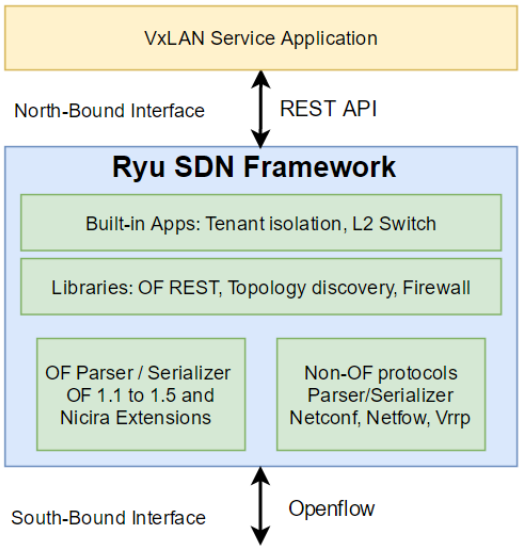


Figure 1 SDN Controller Block Diagram [2]

SDN Controller Components

- **Ryu Framework:** Ryu is a component-based software defined networking framework. Ryu provides a well-defined API that enable developers to create new network management and control applications over it. Ryu supports various protocols for managing network devices, such as OpenFlow, NETCONF, OF-CONFIG, etc.

- **OpenFlow:** According to Open Networking Foundation, "OpenFlow is the first standard communications interface defined between the Control and Forwarding layers of an SDN architecture." It allows access and manipulation of network devices such as switches and routers. (Quote from [3])

- **VxLAN Service Application:** This is a python application written to parse the configuration file input at startup and sets the flows in the VTEPs accordingly. It will also have the functionality to add or remove a tenant from the tunneling service.

4. L3 Backbone

We are using a set of routers to model the L3 Boundary between the (datacenters) Ubuntu machines. Theses routers are realized using basic IP forwarding capabilities of Linux machine, so each router will be an Ubuntu machine in ExoGeni testbed.

2.5. Component Functionality

To simulate the datacenter, we are creating Ubuntu nodes in ExoGeni which run a hypervisor (Virtual Box) to create tenants VMs. All the nodes representing a DC have OvS installed to act as VTEPs. In case of SDN approach, these VTEPs are connected to a SDN controller (RYU) running as a process at another Ubuntu node, the SDN controller is responsible for setting up flows to enable tenant VM communication. In case of traditional approach, we set up flows manually at the OvS. The L3 underlay network is simulated by having Ubuntu node as routing machine by enabling the IP forwarding and all the DCs are connected through this router

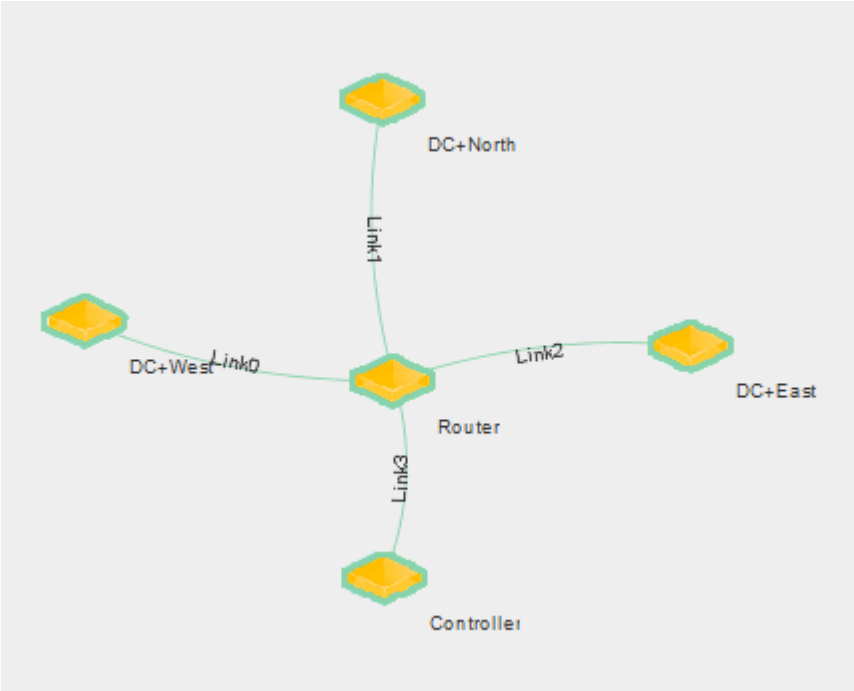


Figure 2 ExoGeni Topology

3. Design and Development plan

Brief description of VxLAN Tunneling Technology

Fundamentally, VXLAN provides mechanisms to aggregate and tunnel multiple layer 2 (sub) networks across a Layer 3 infrastructure. The VXLAN base case is to connect two or more layer three network domains and make them look like a common layer two domains. This allows virtual machines on different networks to communicate as if they were in the same layer 2 subnets.

VxLAN has three main components:

- **Tenant Network** - To the tenant, it must appear like spread across the L3 underlay network is a single L2 domain.
- **VTEP** - The VTEP is a logical endpoint of the VxLAN Tunnel, and this is where encapsulation/de-encapsulation of the VxLAN headers happen.
- **L3 Underlay Network** - This is the network that carries the VxLAN-encapsulated packets from one data center site to the other.

VxLAN Encapsulation Format

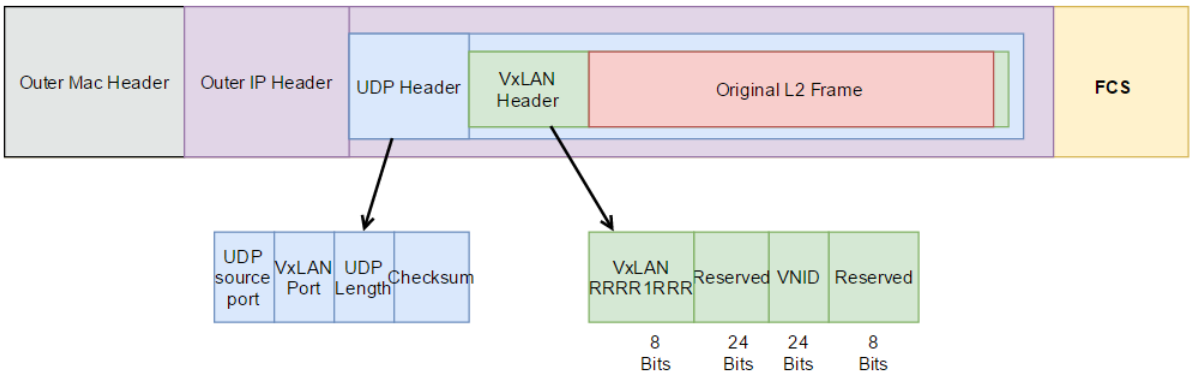


Figure 3 VxLAN Packet format [4]

Design and Implementation Considerations:

1. **Tenant Identification** - The VTEP is the entity that is responsible for adding the VNI tag. To do this, the VTEP must be able to identify which tenant an incoming packet belongs to. There are potentially multiple ways of achieving this-

    i. **IP Address** - This would limit the addressing capabilities of the tenant networks, and is therefore not a feasible option.

    ii. **MAC Address** - In a VM-Environment, MAC Addresses are dynamically allocated. Ensuring uniqueness will be an additional overhead for the network admin. Also, different tenants can have the same mac addresses

    iii. **VLAN Tag** - The advantage VxLAN has over VLAN is that the tag is a 24-bit field, which lets us support 2^{24} tenants, as opposed to $2^{12} = 4096$ with VLAN. If we use the VLAN tag information as our unique identifier, we are again limited by 4096 instances. Thus, this option is feasible, but not the best.

    iv. **Ingress Interface** - If we design such that behind every VTEP Interface, we have only one unique tenant, then this information can have used to pick the VNI ID. Also, since we are using a virtual switch

2. The VxLAN encapsulation header adds 50 bytes to the overall size of an Ethernet frame. Therefore, we will need to consider the fact that our network entities and infrastructure support jumbo frames. We will also have to keep in mind that the link bandwidth can accommodate these large frames [5]. We can also modify the MTU values at the traffic source (Tenant VMs).

3. Layer 2 Broadcasts in the Overlay network are realized as Multicasts between the VTEPs in the Underlay network. It is also required that we map one multicast group to each VNI. This ensures MAC table updates are only sent to VTEPs that require them. It is possible to use only one multicast address for every VNI, but this will effectively flood addresses to VTEPs not needing them and create unnecessary traffic flows in the network. As designers of the system, we will have to ensure that this capability is in place.

3.1. High-level proposed design

Network Topology

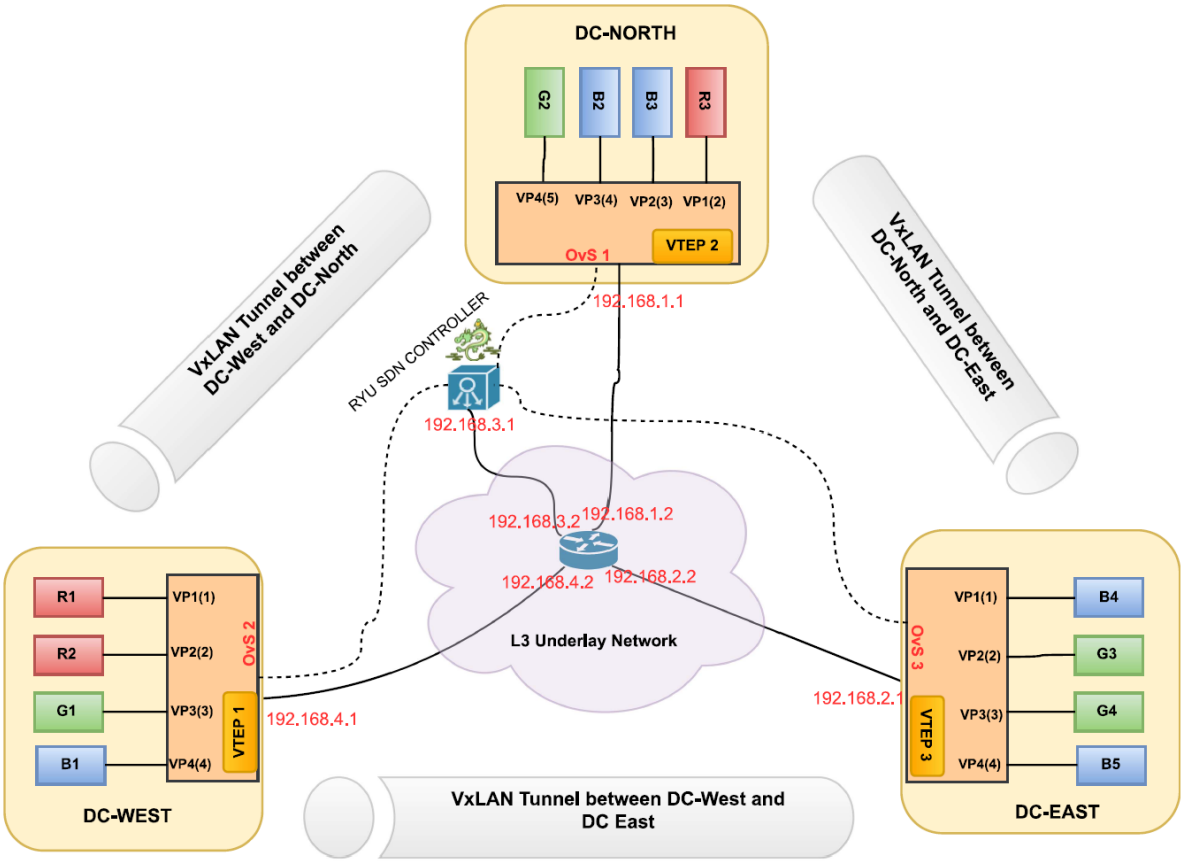


Figure 4 Network Topology for SDN Approach

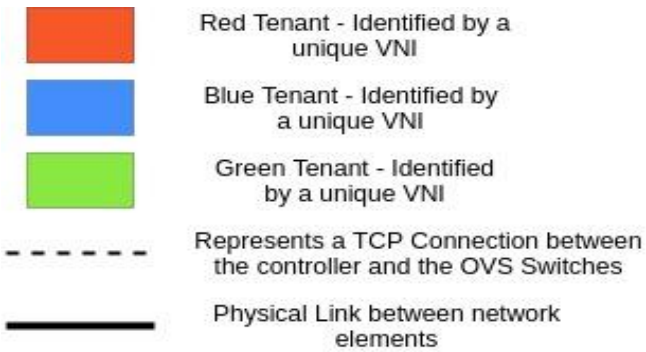


Figure 5 Legend for Topology

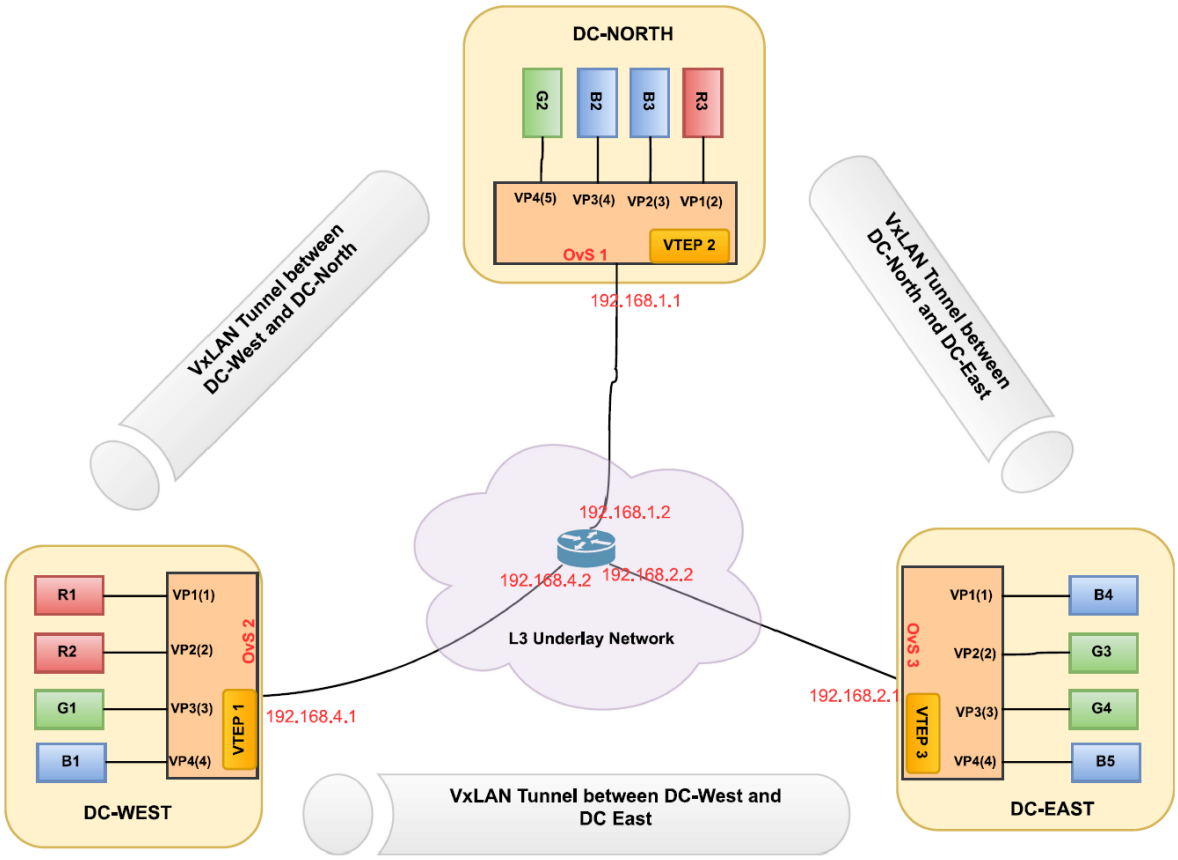


Figure 6 Network Topology Traditional Approach

3.2 Low Level Design

Tenant to VNI mapping

Tenant	VNI
Blue	101
Red	102
Green	103

IP Addressing Scheme

Datacenter	VTEP	Tenants	IP Address
DC - West	VTEP 1	R1	10.0.1.1
		R2	10.0.1.2

		G1	10.0.2.1
		B1	10.0.1.1
DC - North	VTEP 2	R3	10.0.1.3
		B3	10.0.1.3
		B2	10.0.1.2
		G2	10.0.2.2
DC - East	VTEP 3	B4	10.0.1.4
		G3	10.0.2.3
		G4	10.0.2.4
		B5	10.0.1.5

Flows Table Pipeline at OvS

Table No	Functionality
Table 0	• Add VNI tag based on Port to VNI mapping followed by a resubmit rule to table 1 • Default entry to drop packets from unknown ports
Table 1	• Match Destination MAC and VNI to output port • When no match send to controller

OvS Port Configuration

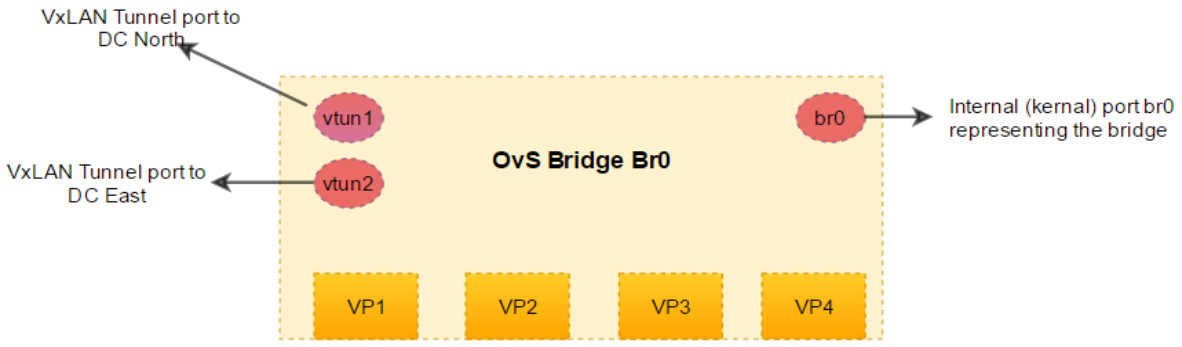


Figure 7 OvS Port Config at VTEP at DC-WEST

Ports VP1, VP2, VP3, VP4 are created in OvS for Tenant VMs and vtun1 an vtun2 are the Tunnel ports

Specifics of SDN Approach:

Controller Logic:

In any SDN implementation, two main Event Handling Methods are defined –

i. Event Handler for New Switches and
ii. Packet – In Handler.

Our design of these two handlers is given below:

New Switch Detected Event:

Trigger

At the OVS we have specified a connection to the Controller by providing an IP Address. So the OVS periodically connects to the controller and triggers the ryu.controller.ofp_event.**EventOFPSwitchFeatures** event.

Action

I. Extract the DPID from the Datapath Object Instance
II. This DPID is used as a Key to extract the matching configuration, which is the associated value in the mapping dictionary that we have created.
III. We now want to install a primary table, where we match the incoming port of an incoming packet to an associated VNI ID, and subsequently resubmit the lookup to a second table. So we iterate through the ports, and install an ACTION to add associated tags. (This achieves **tenant identification**, which we are doing by matching ports)
IV. We install a default rule in Table 1 to forward the packet to the Controller.

Packet In Event:

Trigger

The OVS receives a packet, checks incoming port, adds associated tag, and resubmits to Table, where the instruction is to send it to Controller.

Action:

I. We extract the In_port and the Tag information from the Meta-data associated with the packet.

II. To emulate L2 Mac Learning, we save the In_Port as the value mapped to the VNI, Source MAC Address found in the packet. We now know which port a particular VNI Host is attached to.

III. We check the Destination MAC Address.

IV. If it is a broadcast packet (For example, an ARP Request) received on the VxLAN Tunnel, we broadcast it on all the local ports (without VNI Tag). If it is received on a local port, we broadcast it on all the VxLAN Tunnels with the tag attached, as well as on all local ports without the tag.

V. If it is a unicast packet (For example, an ARP Response), we extract the destination MAC Address from the packet and extract the out_port as the associated value from the stored mapping. We are now in a position to add two flows at this Switch – Match (VNI, Source MAC Address) to the incoming port, and the (VNI, Destination MAC Address) to the output port.

VM/Tenant addition and deletion -

Another key functionality in our project is addition and deletion of VMs/tenants. We achieve this in the following manner.

I. We have a local dictionary object that gets based on a JSON file stored on disk.

II. Every 4 seconds, we check the JSON file to determine if there have been any changes made to it

III. If it detected that changes have taken place, then we reload the JSON file and update the flow tables of the OVS bridges as required.

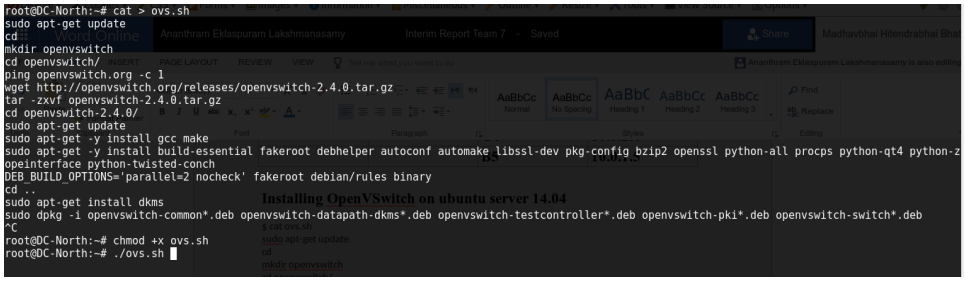
3.3 Implementation

3.3.1. Installing OpenVSwitch on ubuntu server 14.04

cat > ovs.sh

sudo apt-get update
cd
mkdir openvswitch
cd openvswitch/
ping openvswitch.org -c 1
wget http://openvswitch.org/releases/openvswitch-2.4.0.tar.gz
tar -zxvf openvswitch-2.4.0.tar.gz
cd openvswitch-2.4.0/
sudo apt-get update
sudo apt-get -y install gcc make
sudo apt-get -y install build-essential fakeroot debhelper autoconf automake libssl-dev pkg-config bzip2 openssl python-all procps python-qt4 python-zopeinterface python-twisted-conch
DEB_BUILD_OPTIONS='parallel=2 nocheck' fakeroot debian/rules binary
cd ..
sudo apt-get install dkms
sudo dpkg -i openvswitch-common*.deb openvswitch-datapath-dkms*.deb openvswitch-testcontroller*.deb openvswitch-pki*.deb openvswitch-switch*.deb

chmod +x ovs.sh
./ovs.sh



3.3.2. Installing VNC Server on ubuntu server 14.04 [6]

apt-get install gnome-core xfce4 firefox vnc4server –y
addusers user2
usermod –a –G sudo user2
chsh –s /bin/bash

```
root@DC-North:~# adduser user2
Adding user `user2' ...
Adding new group `user2' (1002) ...
Adding new user `user2' (1002) with group `user2' ...
Creating home directory `/home/user2' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:

Retype new UNIX password:

passwd: password updated successfully
Changing the user information for user2
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
root@DC-North:~# su - us
usbmux  user1   user2
root@DC-North:~# usermod -a -G sudo user2
root@DC-North:~# chsh -s /bin/bash user2
```

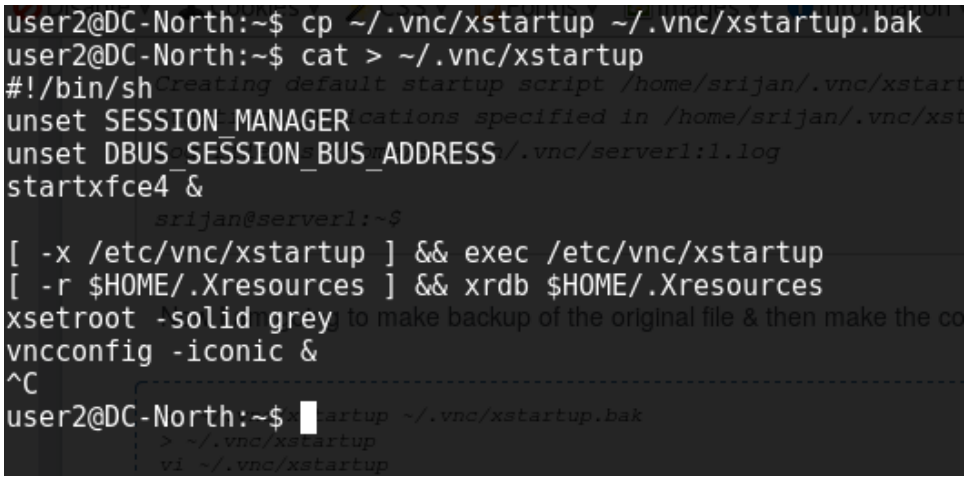
su – user2
vncserver

```
root@DC-North:~# su - user2
user2@DC-North:~$ vncserver

You will require a password to access your desktops.

Password:
Verify:
xauth:  file /home/user2/.Xauthority does not exist

New 'DC-North:3 (user2)' desktop is DC-North:3

Creating default startup script /home/user2/.vnc/xstartup
Starting applications specified in /home/user2/.vnc/xstartup
Log file is /home/user2/.vnc/DC-North:3.log

user2@DC-North:~$
```

cp ~/.vnc/xstartup ~/.vnc/xstartup.bak
cat > ~/.vnc/xstartup

#!/bin/sh
unset SESSION_MANAGER
unset DBUS_SESSION_BUS_ADDRESS
startxfce4 &

[-x /etc/vnc/xstartup] && exec /etc/vnc/xstartup
[-r $HOME/.Xresources] && xrdb $HOME/.Xresources
xsetroot -solid grey
vncconfig -iconic &

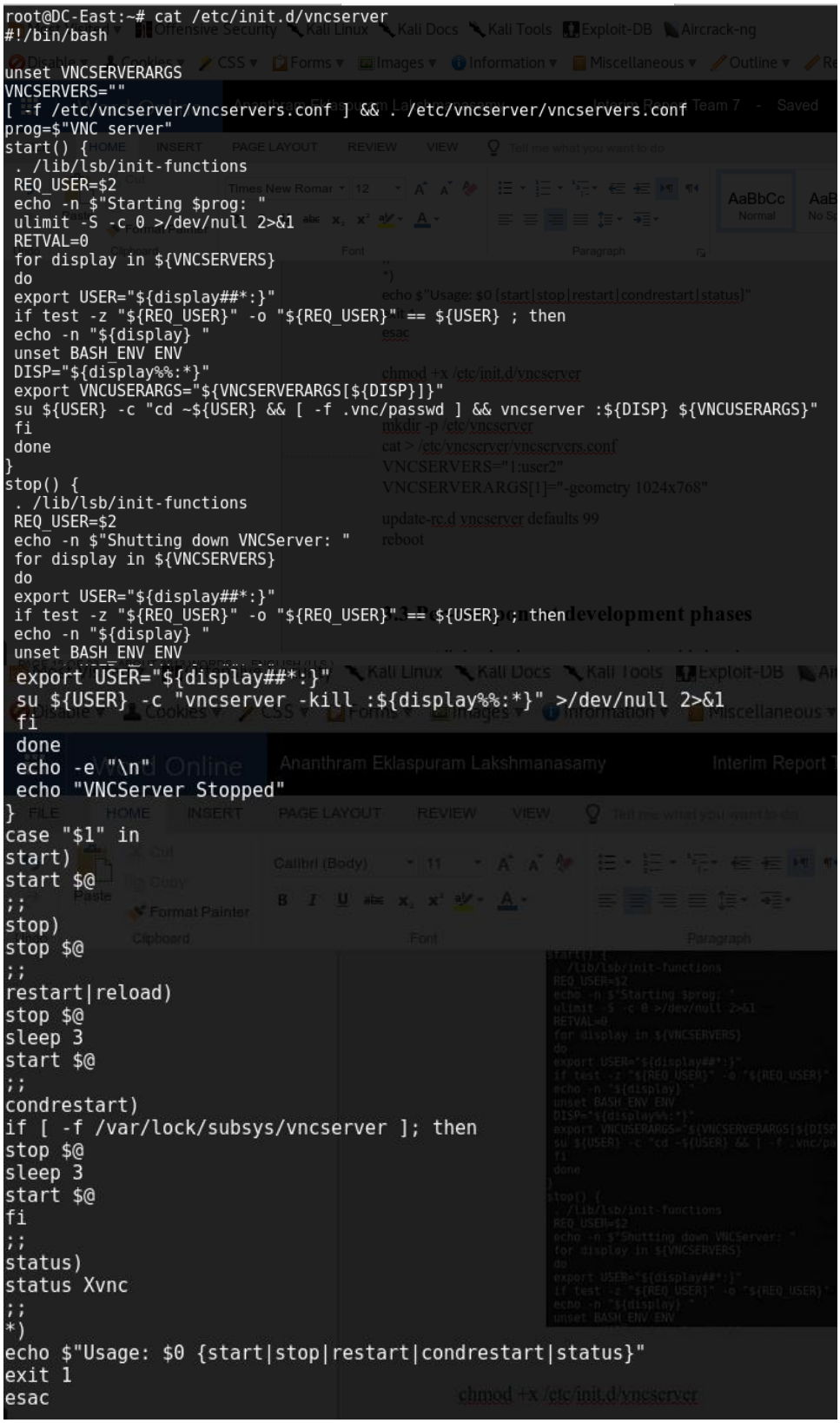


vncserver -kill :1
sudo su
cat > /etc/init.d/vncserver

#!/bin/bash

unset VNCSERVERARGS
VNCSERVERS=""
[-f /etc/vncserver/vncservers.conf] && . /etc/vncserver/vncservers.conf
prog=$"VNC server"
start() {
. /lib/lsb/init-functions
REQ_USER=$2
echo -n $"Starting $prog: "
ulimit -S -c 0 >/dev/null 2>&1
RETVAL=0
for display in ${VNCSERVERS}
do
export USER="${display##*:}"
if test -z "${REQ_USER}" -o "${REQ_USER}" == ${USER} ; then
echo -n "${display} "
unset BASH_ENV ENV
DISP="${display%%:*}"
export VNCUSERARGS="${VNCSERVERARGS[${DISP}]}"
su ${USER} -c "cd ~${USER} && [-f .vnc/passwd] && vncserver :${DISP} ${VNCUSERARGS}"
fi
done
}
stop() {

```
. /lib/lsb/init-functions
REQ_USER=$2
echo -n $"Shutting down VNCServer: "
for display in ${VNCSERVERS}
do
export USER="${display##*:}"
if test -z "${REQ_USER}" -o "${REQ_USER}" == ${USER} ; then
echo -n "${display} "
unset BASH_ENV ENV
export USER="${display##*:}"
su ${USER} -c "vncserver -kill :${display%%:*}" >/dev/null 2>&1
fi
done
echo -e "\n"
echo "VNCServer Stopped"
}
case "$1" in
start)
start $@
;;
stop)
stop $@
;;
restart|reload)
stop $@
sleep 3
start $@
;;
condrestart)
if [ -f /var/lock/subsys/vncserver ]; then
stop $@
sleep 3
start $@
fi
;;
status)
status Xvnc
;;
*)
echo $"Usage: $0 {start|stop|restart|condrestart|status}"
exit 1
esac
```



```
root@DC-East:~# cat /etc/init.d/vncserver
#!/bin/bash
unset VNCSERVERARGS
VNCSERVERS=""
[ -f /etc/vncserver/vncservers.conf ] && . /etc/vncserver/vncservers.conf
prog=$"VNC server"
start() {
 . /lib/lsb/init-functions
 REQ_USER=$2
 echo -n $"Starting $prog: "
 ulimit -S -c 0 >/dev/null 2>&1
 RETVAL=0
 for display in ${VNCSERVERS}
 do
 export USER="${display##*:}"
 if test -z "${REQ_USER}" -o "${REQ_USER}" == ${USER} ; then
 echo -n "${display} "
 unset BASH_ENV ENV
 DISP="${display%%:*}"
 export VNCUSERARGS="${VNCSERVERARGS[${DISP}]}"
 su ${USER} -c "cd ~${USER} && [ -f .vnc/passwd ] && vncserver :${DISP} ${VNCUSERARGS}"
 fi
 done
}
stop() {
 . /lib/lsb/init-functions
 REQ_USER=$2
 echo -n $"Shutting down VNCServer: "
 for display in ${VNCSERVERS}
 do
 export USER="${display##*:}"
 if test -z "${REQ_USER}" -o "${REQ_USER}" == ${USER} ; then
 echo -n "${display} "
 unset BASH_ENV ENV
 export USER="${display##*:}"
 su ${USER} -c "vncserver -kill :${display%%:*}" >/dev/null 2>&1
 fi
 done
 echo -e "\n"
 echo "VNCServer Stopped"
}
case "$1" in
start)
start $@
;;
stop)
stop $@
;;
restart|reload)
stop $@
sleep 3
start $@
;;
condrestart)
if [ -f /var/lock/subsys/vncserver ]; then
stop $@
sleep 3
start $@
fi
;;
status)
status Xvnc
;;
*)
echo $"Usage: $0 {start|stop|restart|condrestart|status}"
exit 1
esac
```

chmod +x /etc/init.d/vncserver
mkdir -p /etc/vncserver
cat > /etc/vncserver/vncservers.conf
VNCSERVERS="1:user2"
VNCSERVERARGS[1]="-geometry 1024x768"

```
root@DC-East:~# cat /etc/vncserver/vncservers.conf
VNCSERVERS="1:user2"
VNCSERVERARGS[1]="-geometry 1024x768"
root@DC-East:~#
```

update-rc.d vncserver defaults 99
reboot

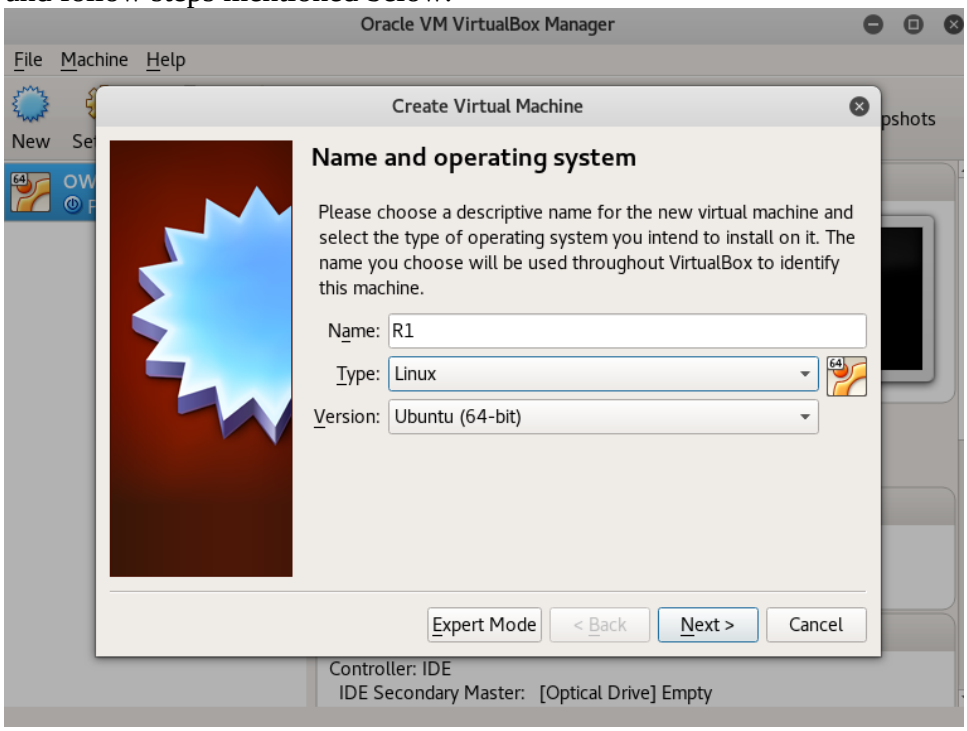
3.3.3. Creating Tenant VMs

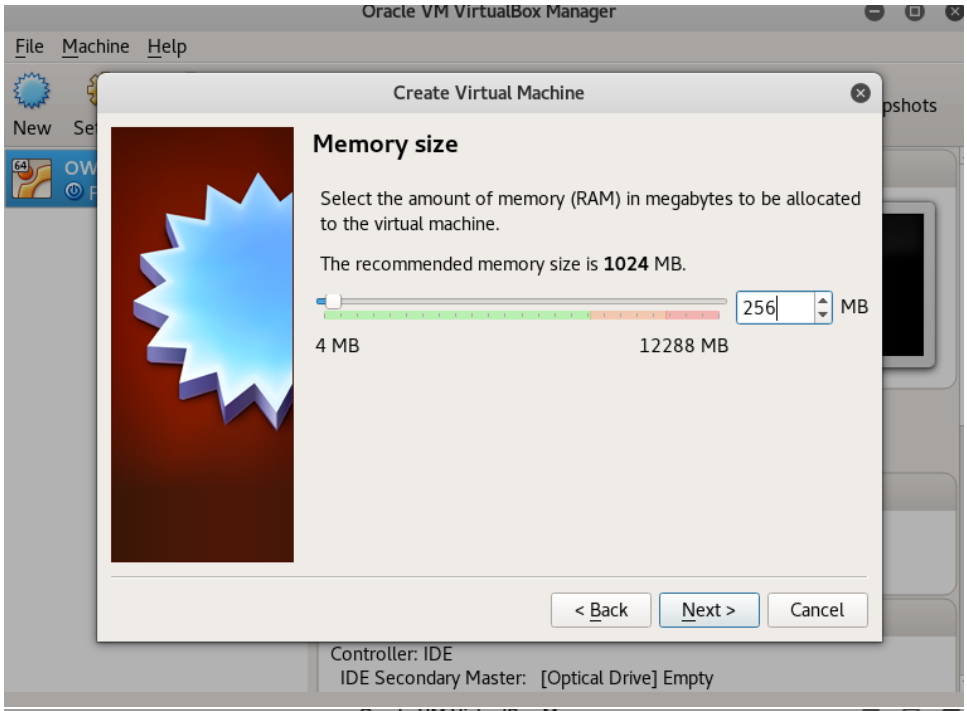
Installing Virtual Box on ubuntu 14.04

sudo apt-get update
sudo apt-get install virtualbox -y

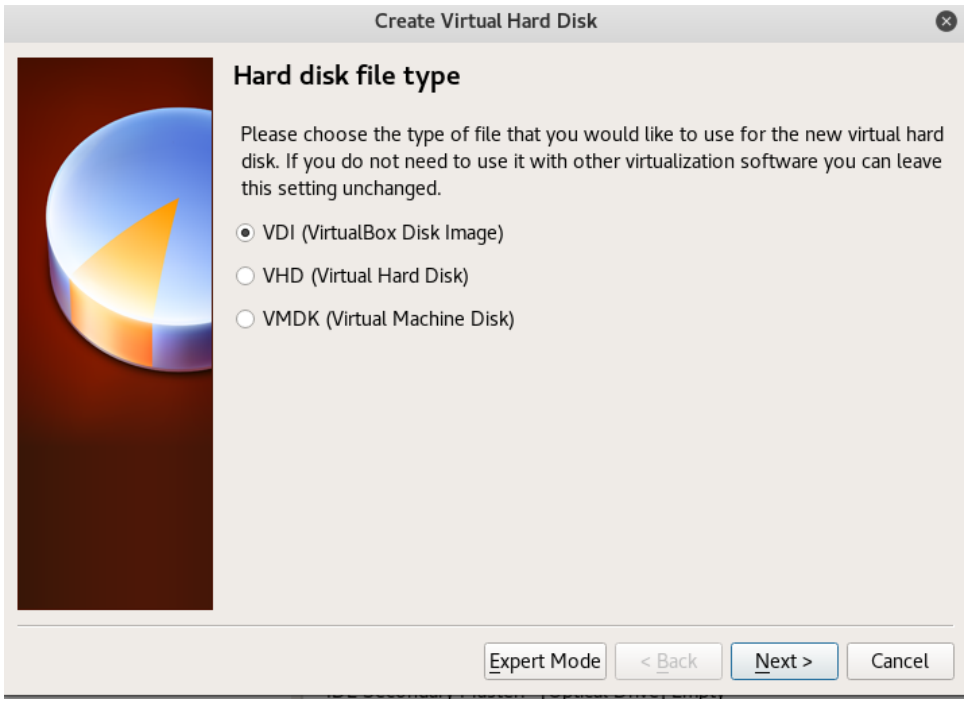
Installing Tenant in Virtual Box

Click New and follow steps mentioned below.

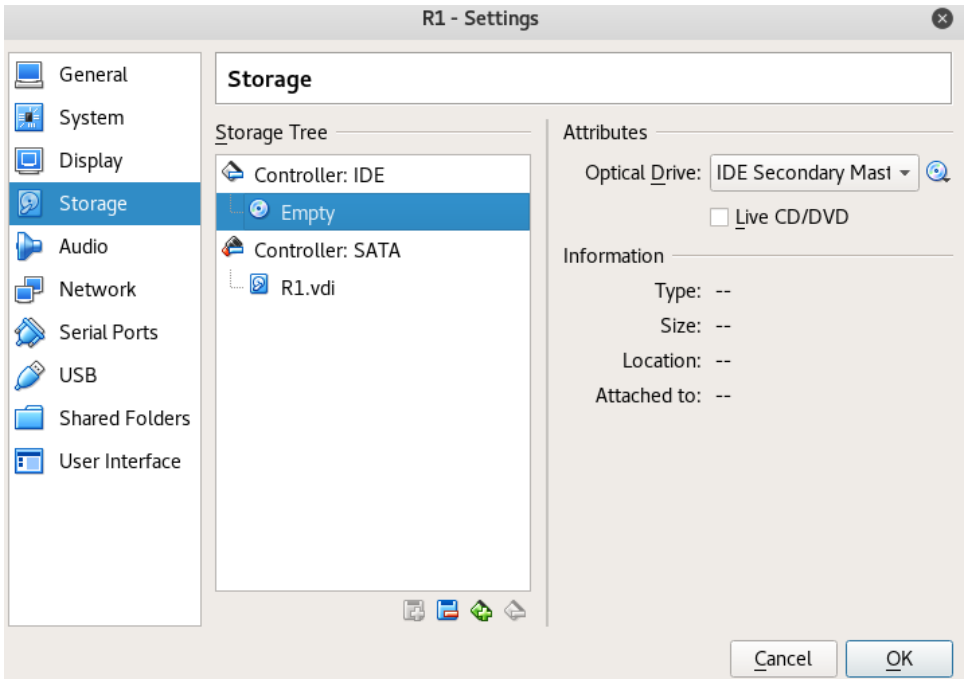



Oracle VM VirtualBox Manager
File Machine Help
New
Create Virtual Machine
Memory size
Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.
The recommended memory size is 1024 MB.
256
MB
4 MB
12288 MB
< Back
Next >
Cancel
Controller: IDE
IDE Secondary Master: [Optical Drive] Empty


Oracle VM VirtualBox Manager
File Machine Help
New
Create Virtual Machine
Hard disk
If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.
If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.
The recommended size of the hard disk is 8.00 GB.
Do not add a virtual hard disk
Create a virtual hard disk now
Use an existing virtual hard disk file
OWASP Broken Web Apps-cl1.vmdk (Normal, 8.00 GB)
< Back
Create
Cancel
Controller: IDE
IDE Secondary Master: [Optical Drive] Empty

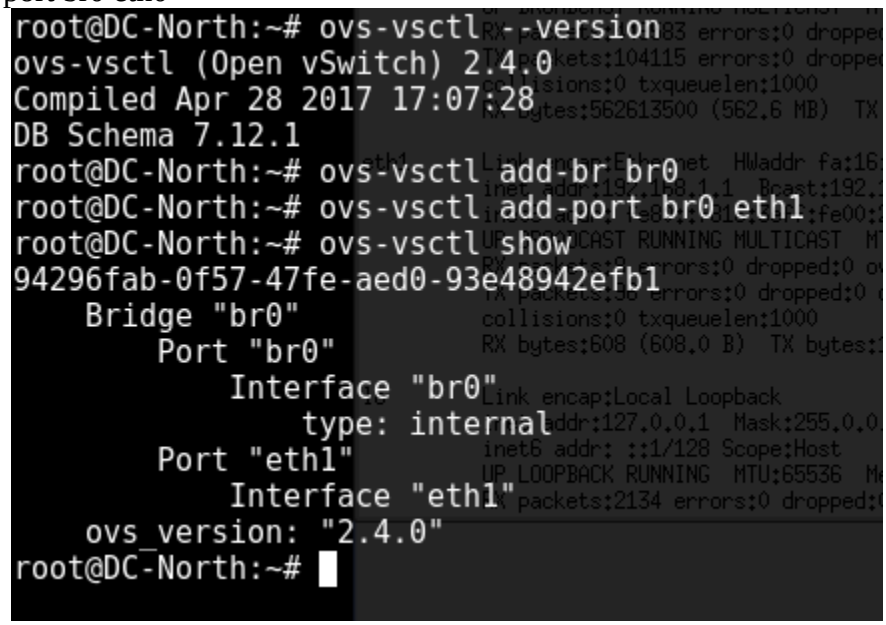


After Machine Gets Created, select the Virtual Machine and go to settings > storage. In Controller IDE, select ubuntu 14.04 iso as Operating System. Afterwards click ok and start VM and follow operating system installation which is out of scope of this document.



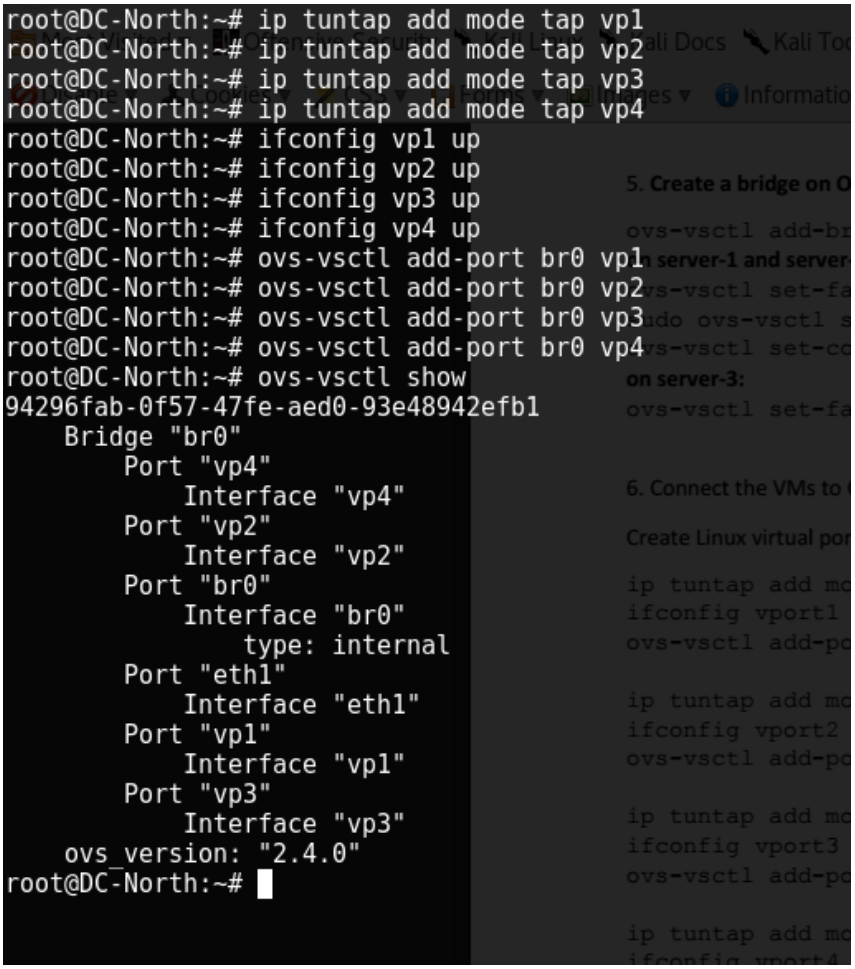
3.3.4. Configuring OpenvSwitch

ovs-vsctl –version
ovs-vsctl add-br br0
ovs-vsctl add-port br0 eth0



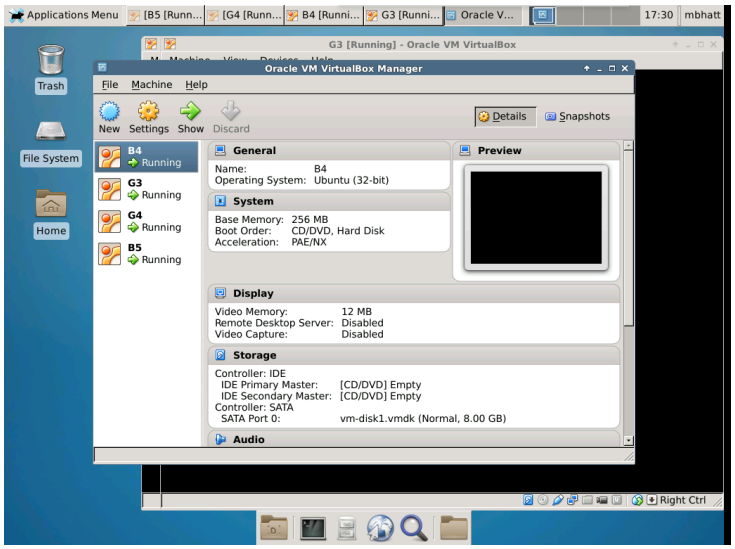
ip tuntap add mode tap vp1
ip tuntap add mode tap vp2
ip tuntap add mode tap vp3
ip tuntap add mode tap vp4
ifconfig vp1 up
ifconfig vp2 up

ifconfig vp3 up
ifconfig vp4 up
ovs-vsctl add-port br0 vp1
ovs-vsctl add-port br0 vp2
ovs-vsctl add-port br0 vp3
ovs-vsctl add-port br0 vp4
ovs-vsctl show

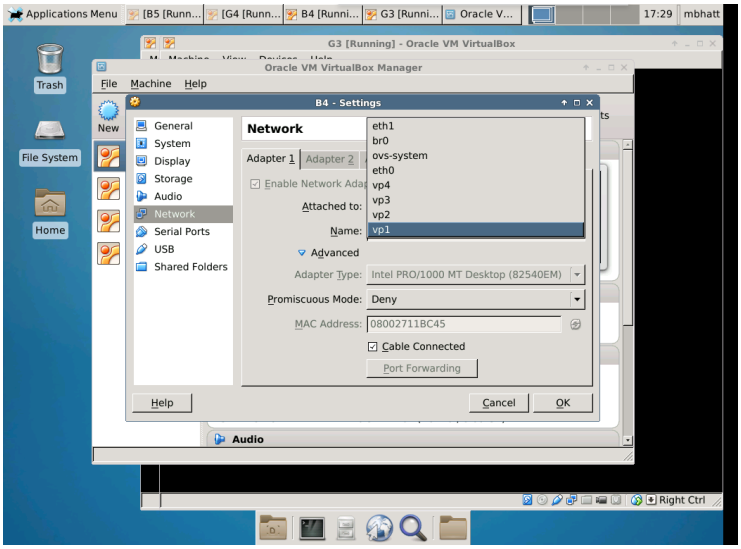


Connecting Tenants with OpenvSwitch

Select Tenant you want to connect to OpenvSwitch.



Go to settings > Network and select VTEP port tenant is supposed to be connected to.



3.3.5. Configuring Routing Table

On DC-North:
ip route add 192.168.2.0/24 via 192.168.1.2
ip route add 192.168.4.0/24 via 192.168.1.2

On DC-East:
ip route add 192.168.1.0/24 via 192.168.2.2
ip route add 192.168.4.0/24 via 192.168.2.2

On DC-West:
ip route add 192.168.1.0/24 via 192.168.4.2
ip route add 192.168.2.0/24 via 192.168.4.2

3.3.6. Configuring VxLAN Tunnels [7] [8]

(Flows are added in Traditional approach only)

Creating VxLAN on DC-North

ovs-vsctl add-port br1 vtun1 --set interface vtun1 type=vxlan option:remote_ip=192.168.2.1 option:key=flow ofport_request=10

ovs-vsctl add-port br1 vtun2 --set interface vtun2 type=vxlan option:remote_ip=192.168.4.1 option:key=flow ofport_request=11

```
cat > northflows.txt
table=0,in_port=2,actions=set_field:102->tun_id,resubmit(,1)
table=0,in_port=3,actions=set_field:101->tun_id,resubmit(,1)
table=0,in_port=4,actions=set_field:101->tun_id,resubmit(,1)
table=0,in_port=5,actions=set_field:103->tun_id,resubmit(,1)
table=0,actions=resubmit(,1)

table=1,tun_id=102,dl_dst=00:00:00:00:00:03,actions=output:2
table=1,tun_id=101,dl_dst=00:00:00:00:00:03,actions=output:3
table=1,tun_id=101,dl_dst=00:00:00:00:00:02,actions=output:4
table=1,tun_id=103,dl_dst=00:00:00:00:00:02,actions=output:5
table=1,tun_id=102,arp,nw_dst=10.0.1.3,actions=output:2
table=1,tun_id=101,arp,nw_dst=10.0.1.3,actions=output:3
table=1,tun_id=101,arp,nw_dst=10.0.1.2,actions=output:4
table=1,tun_id=103,arp,nw_dst=10.0.2.2,actions=output:5

table=1,tun_id=102,dl_dst=00:00:00:00:00:01,actions=output:11
table=1,tun_id=102,dl_dst=00:00:00:00:00:02,actions=output:11
table=1,tun_id=103,dl_dst=00:00:00:00:00:01,actions=output:11
table=1,tun_id=101,dl_dst=00:00:00:00:00:01,actions=output:11
table=1,tun_id=102,arp,nw_dst=10.0.1.1,actions=output:11
table=1,tun_id=102,arp,nw_dst=10.0.1.2,actions=output:11
table=1,tun_id=103,arp,nw_dst=10.0.2.1,actions=output:11
table=1,tun_id=101,arp,nw_dst=10.0.1.1,actions=output:11

table=1,tun_id=101,dl_dst=00:00:00:00:00:04,actions=output:10
table=1,tun_id=103,dl_dst=00:00:00:00:00:03,actions=output:10
table=1,tun_id=103,dl_dst=00:00:00:00:00:04,actions=output:10
table=1,tun_id=101,dl_dst=00:00:00:00:00:05,actions=output:10
table=1,tun_id=101,arp,nw_dst=10.0.1.4,actions=output:10
table=1,tun_id=103,arp,nw_dst=10.0.2.3,actions=output:10
table=1,tun_id=103,arp,nw_dst=10.0.2.4,actions=output:10
table=1,tun_id=101,arp,nw_dst=10.0.1.5,actions=output:10

ovs-ofctl add-flows br0 northflows.txt
```

Creating VxLAN on DC-East

ovs-vsctl add-port br1 vtun2 --set interface vtun2 type=vxlan option:remote_ip=192.168.1.1 option:key=flow ofport_request=10

ovs-vsctl add-port br1 vtun1 --set interface vtun1 type=vxlan option:remote_ip=192.168.4.1 option:key=flow ofport_request=11

cat > eastflows.txt

```
table=0,in_port=2,actions=set_field:101->tun_id,resubmit(,1)
table=0,in_port=3,actions=set_field:103->tun_id,resubmit(,1)
table=0,in_port=4,actions=set_field:103->tun_id,resubmit(,1)
table=0,in_port=5,actions=set_field:101->tun_id,resubmit(,1)
table=0,actions=resubmit(,1)

table=1,tun_id=101,dl_dst=00:00:00:00:00:04,actions=output:2
table=1,tun_id=103,dl_dst=00:00:00:00:00:03,actions=output:3
table=1,tun_id=103,dl_dst=00:00:00:00:00:04,actions=output:4
table=1,tun_id=101,dl_dst=00:00:00:00:00:05,actions=output:5
table=1,tun_id=101,arp,nw_dst=10.0.1.4,actions=output:2
table=1,tun_id=103,arp,nw_dst=10.0.2.3,actions=output:3
table=1,tun_id=103,arp,nw_dst=10.0.2.4,actions=output:4
table=1,tun_id=101,arp,nw_dst=10.0.1.5,actions=output:5

table=1,tun_id=102,dl_dst=00:00:00:00:00:03,actions=output:10
table=1,tun_id=101,dl_dst=00:00:00:00:00:03,actions=output:10
table=1,tun_id=101,dl_dst=00:00:00:00:00:02,actions=output:10
table=1,tun_id=103,dl_dst=00:00:00:00:00:02,actions=output:10
table=1,tun_id=102,arp,nw_dst=10.0.1.3,actions=output:10
table=1,tun_id=101,arp,nw_dst=10.0.1.2,actions=output:10
table=1,tun_id=101,arp,nw_dst=10.0.1.3,actions=output:10
table=1,tun_id=103,arp,nw_dst=10.0.2.2,actions=output:10

table=1,tun_id=102,dl_dst=00:00:00:00:00:01,actions=output:11
table=1,tun_id=102,dl_dst=00:00:00:00:00:02,actions=output:11
table=1,tun_id=103,dl_dst=00:00:00:00:00:01,actions=output:11
table=1,tun_id=101,dl_dst=00:00:00:00:00:01,actions=output:11
table=1,tun_id=102,arp,nw_dst=10.0.1.1,actions=output:11
table=1,tun_id=102,arp,nw_dst=10.0.1.2,actions=output:11
table=1,tun_id=103,arp,nw_dst=10.0.2.1,actions=output:11
table=1,tun_id=101,arp,nw_dst=10.0.1.1,actions=output:11
```

ovs-ofctl add-flows br0 eastflows.txt

Creating VxLAN on DC-West

ovs-vsctl add-port br1 vtun1 --set interface vtun1 type=vxlan option:remote_ip=192.168.1.1 option:key=flow ofport_request=10

ovs-vsctl add-port br1 vtun2 --set interface vtun2 type=vxlan option:remote_ip=192.168.2.1 option:key=flow ofport_request=11

cat > westflows.txt

```
table=0,in_port=2,actions=set_field:102->tun_id,resubmit(,1)
table=0,in_port=3,actions=set_field:102->tun_id,resubmit(,1)
table=0,in_port=4,actions=set_field:103->tun_id,resubmit(,1)
table=0,in_port=5,actions=set_field:101->tun_id,resubmit(,1)
table=0,actions=resubmit(,1)

table=1,tun_id=102,dl_dst=00:00:00:00:00:01,actions=output:2
table=1,tun_id=102,dl_dst=00:00:00:00:00:02,actions=output:3
table=1,tun_id=103,dl_dst=00:00:00:00:00:01,actions=output:4
table=1,tun_id=101,dl_dst=00:00:00:00:00:01,actions=output:5

table=1,tun_id=102,arp,nw_dst=10.0.1.1,actions=output:2
table=1,tun_id=102,arp,nw_dst=10.0.1.2,actions=output:3
table=1,tun_id=103,arp,nw_dst=10.0.2.1,actions=output:4
table=1,tun_id=101,arp,nw_dst=10.0.1.1,actions=output:5

table=1,tun_id=101,dl_dst=00:00:00:00:00:04,actions=output:11
table=1,tun_id=103,dl_dst=00:00:00:00:00:03,actions=output:11
table=1,tun_id=103,dl_dst=00:00:00:00:00:04,actions=output:11
table=1,tun_id=101,dl_dst=00:00:00:00:00:05,actions=output:11
table=1,tun_id=101,arp,nw_dst=10.0.1.4,actions=output:11
table=1,tun_id=103,arp,nw_dst=10.0.2.3,actions=output:11
table=1,tun_id=103,arp,nw_dst=10.0.2.4,actions=output:11
table=1,tun_id=101,arp,nw_dst=10.0.1.5,actions=output:11

table=1,tun_id=102,dl_dst=00:00:00:00:00:03,actions=output:10
table=1,tun_id=101,dl_dst=00:00:00:00:00:03,actions=output:10
table=1,tun_id=101,dl_dst=00:00:00:00:00:02,actions=output:10
table=1,tun_id=103,dl_dst=00:00:00:00:00:02,actions=output:10
table=1,tun_id=102,arp,nw_dst=10.0.1.3,actions=output:10
table=1,tun_id=101,arp,nw_dst=10.0.1.3,actions=output:10
table=1,tun_id=101,arp,nw_dst=10.0.1.2,actions=output:10
table=1,tun_id=103,arp,nw_dst=10.0.2.2,actions=output:10
```

ovs-ofctl add-flows br0 westflows.txt

3.3.8. Setup SDN Controller (Used only in the centralized approach)

RYU SDN Controller

- **Install Ryu in the controller machine (Ubuntu 14.04)** [9]

  sudo apt-get -y install git python-pip python-dev
  sudo apt-get -y install python-eventlet python-routes python-webob python-paramiko
  cd ~/
  git clone --depth=1 https://github.com/osrg/ryu.git
  sudo pip install setuptools –upgrade
  cd ryu;
  sudo python ./setup.py install
  sudo pip install six --upgrade
  sudo pip install oslo.config msgpack-python
  sudo pip install eventlet --upgrade

- **Run the VxLAN service application**

  ryu-manager vxlanservice.py

- **Link the OvS to the controller**

  ovs-vsctl set-controller br0 tcp:16.0.0.2:6633

3.3.9. Setup the Ubuntu Router

- Set appropriate static routes

  sudo route add <network> gw <interface IP>
- enable IP forwarding

  sudo sysctl -w net.ipv4.ip_forward=1

3.3 Per Member Learning Objectives

Team Member	Learning Objective
Shishir	i. Understanding the difference between an SDN vs non-SDN Approach in terms of design, functionality, and ease of configuration ii. Understanding tunneling concepts iii. Trying to figure out if we can enhance the multicast functionality, given that we have a controller that possesses network-wide visibility.
Ananthram	i. Understand the need for L2 interconnects in datacenter and how tunneling technologies like VxLAN solve that ii. Understand the advantages and disadvantages of SDN approach as opposed to a traditional distributed approach iii. Understand OpenFlow, OVSDB, and learn to write SDN controller applications
Madhav	i. Understand the OpenvSwitch implementation ii. Understand how VxLAN encapsulation works iii. Implement the VxLAN network using traditional approach.

3.4. Task Allocation and Proposed Timeline:

S.No	Task	Assignee	Dates
1	Understanding the VxLAN technology by relevant RFCs, the project objectives, the two design approaches and the functional decomposition	Shishir, Ananthram, Madhav	03/20 - 03/24
2	Project Proposal and Website	Shishir, Ananthram	03/24 - 03/26
3	Creating the planned topologies on GENI	Shishir, Ananthram, Madhav	03/27 - 03/29

4	Set up the VMs and OvS for the distributed approach	Madhav	03/29 - 03/30
5	Set up the VMs and OvS for the SDN approach	Shishir, Ananthram	03/29 - 03/30
6	Set up the flow entries for distributed approach and unit testing	Madhav	03/30 - 04/02
7	Designing the algorithm for the SDN Approach, VxLAN service application	Ananthram, Shishir	03/31- 04/02
8	Create flows file to demonstrate tenant addition or deletion	Madhav	04/03 - 04/05
9	Code the SDN application: Incoming packets (to Data Center)	Shishir, Ananthram	04/03 - 04/05
10	Implementing the test plan for the non-SDN Approach	Madhav	04/06-04/08
11	Code the SDN application: Outgoing packets	Shishir, Ananthram	04/06 - 04/08
12	Code the SDN application: Add/Remove Tenants	Shishir, Ananthram	04/09 – 04/11
13	Install the SDN controller framework, run the VxLAN application, any troubleshooting	Shishir, Ananthram	04/12
14	Unit and System testing of Centralized SDN Approach	Ananthram, Shishir	04/13 - 04/15
14	Interim Report, update website	Shishir, Ananthram, Madhav	04/10 - 04/15
16	Preparing the demo	Shishir, Ananthram, Madhav	04/15 - 04/20
17	Final Report, update website	All	04/26 - 04/28

4. Verification and validation plan

1. From the Tenant's Perspective - To verify that the L2-connectivity objective has been achieved, packets can be captured on either side of the underlying L3 network and the fields can be examined. To verify that the overall objective has been met, packets can be captured on either side of the underlying L3 network and the fields can be examined.

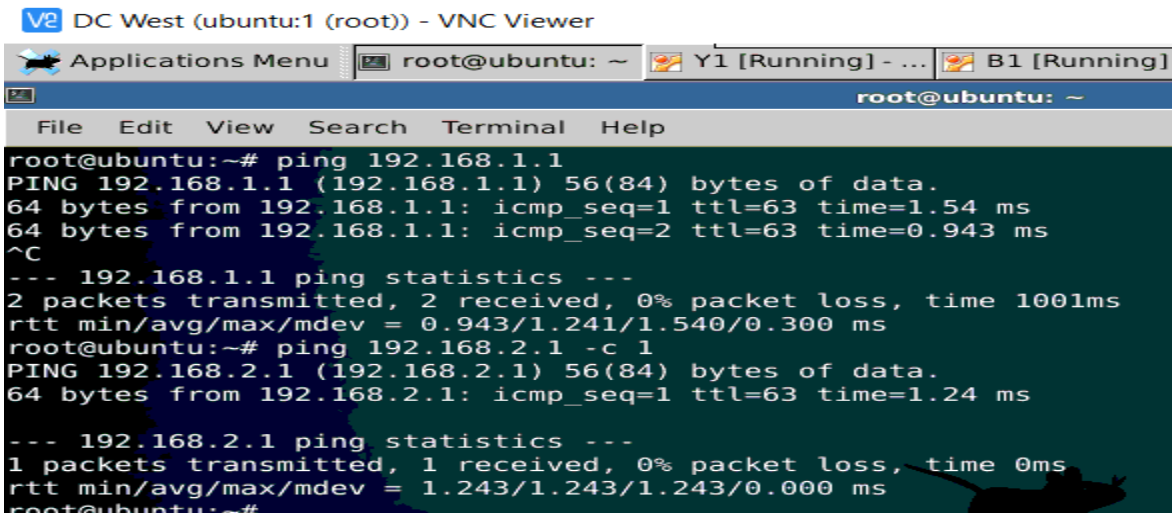
2. To verify the VTEP Functionality, packets will be captured before and after passing through the VTEP. The delta between the two captures must reveal that the VxLAN headers were added at the VTEP.

3. To verify multicast functionality, an ARP request for the MAC Address of a host belonging to a particular tenant, say with VNI X, must be sent to only those VTEPs that are known to have VNI X hosts behind them

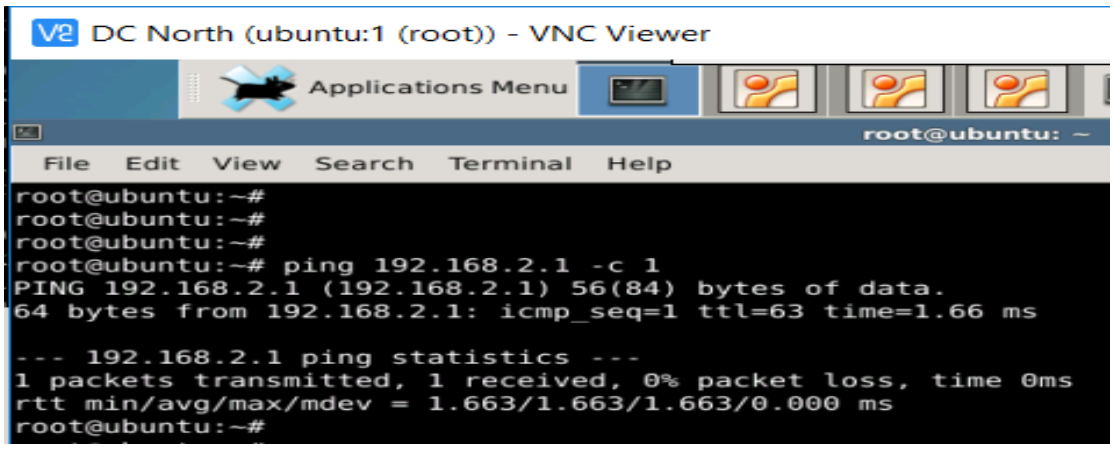
4.1 TEST CASES:

Testing of SDN System

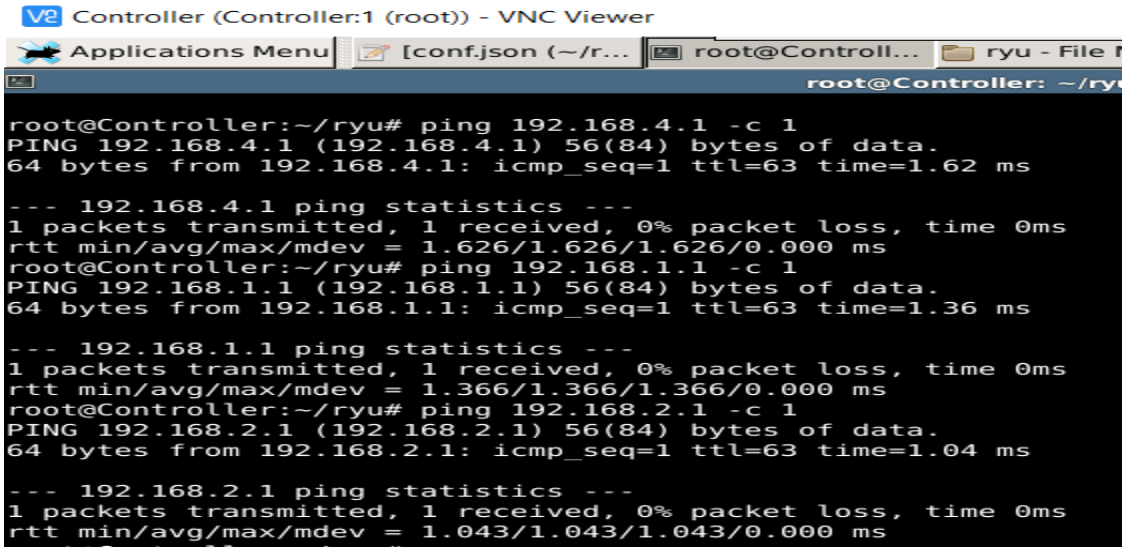
Test ID	**Input and action to System**	**Expected Observation**	**Test pass/failed**	**Conclusion if passed**
T1	Ping the DC-North and DC-East from DC-West. Ping DC-North from DC-East.	Pings are successful.		Connectivity between the Data Centers are ok. Underlay Network is operational.
T1 Result				
Pinging DC-North and DC-East from DC-West				



Ping DC-East from DC-North



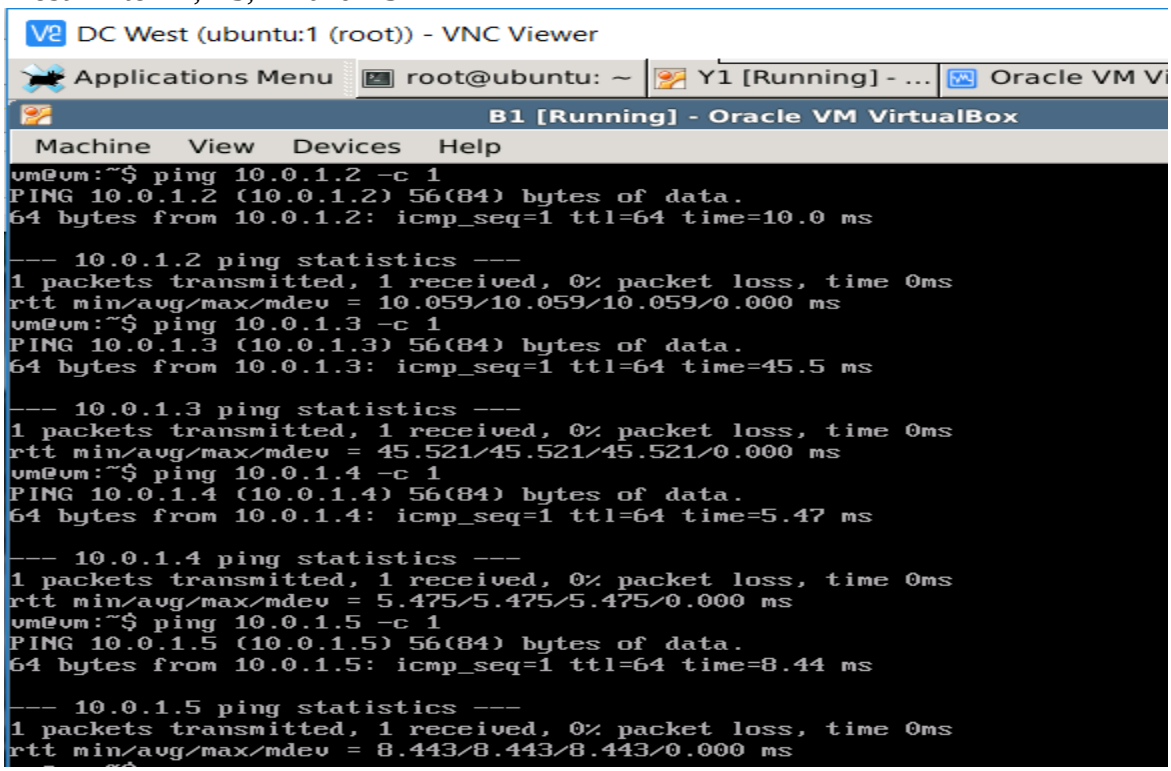
Ping DC-East, DC-North, DC-West from Controller



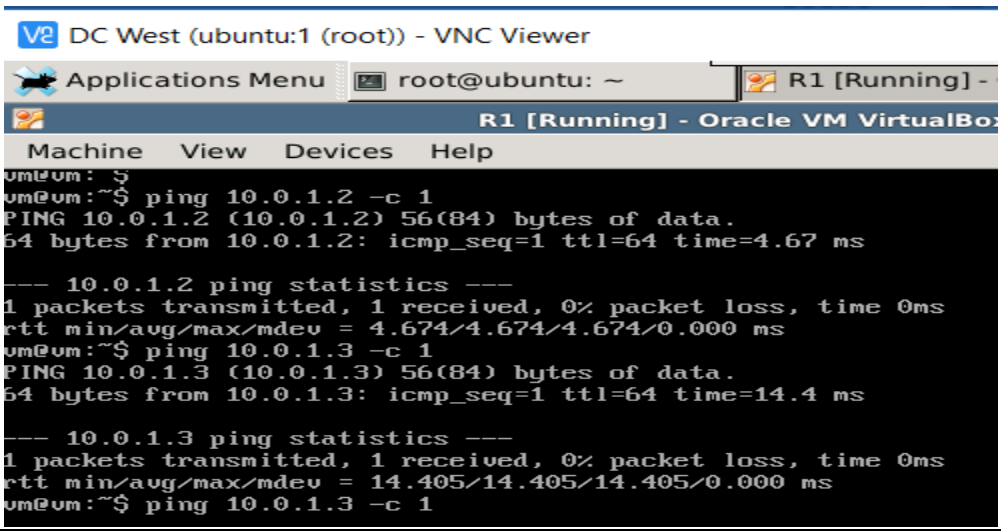
T2	Ping from host B1 to B2, B3, B4 and B5. Ping from host R1 to R2 and R3. Ping from G1 to G2, G3, G4.	Pings are successful.	Pass	Tenants in the same (virtual) L2 domain can communicate with each other. VxLAN service is working.

T2 Result

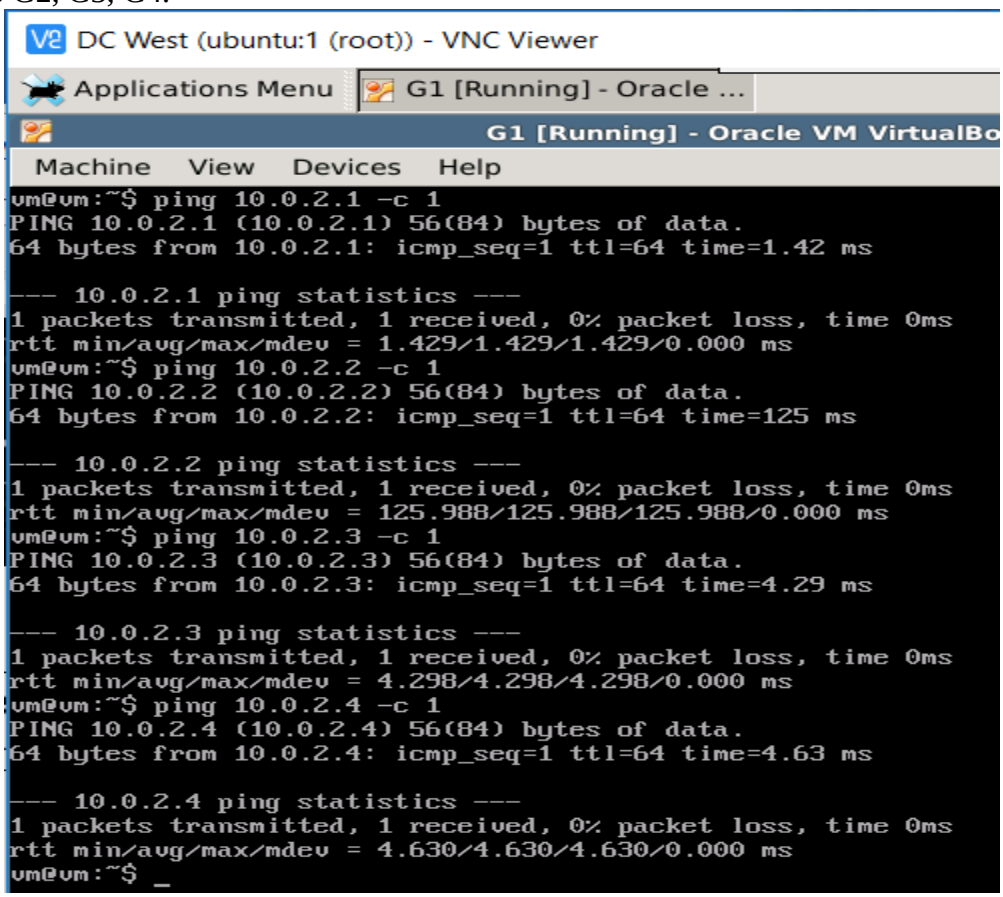
Ping from host B1 to B2, B3, B4 and B5



Ping from host R1 to R2 and R3



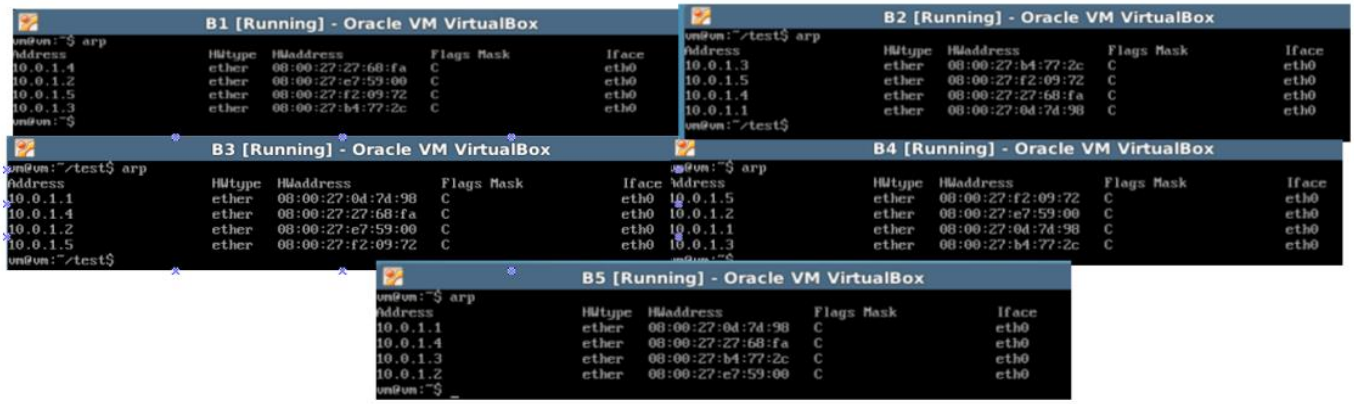
Ping from G1 to G2, G3, G4.



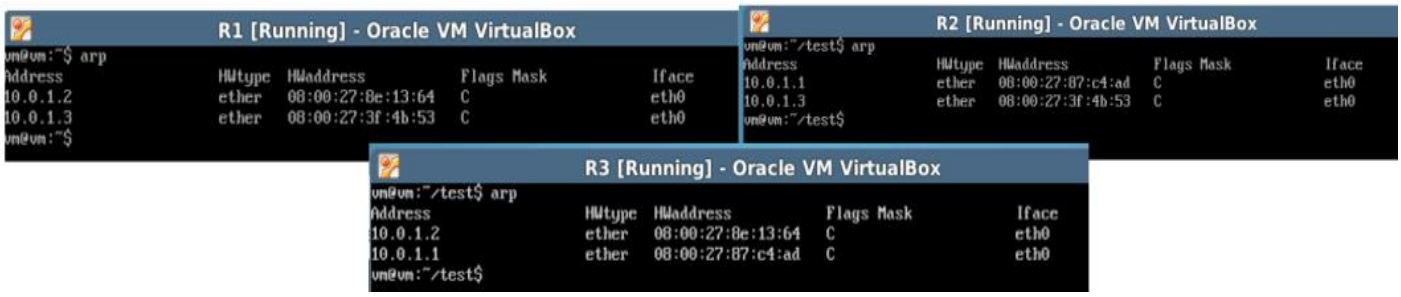
T3	Following T2, and check the ARP tables of the hosts VMs.	The ARP for hosts in one LAN is resolved. {B1, B2, B3, B4, B5} have mac of each other resolved. {R1, R2, R3} have mac of each other resolved. {G1, G2, G3, G4} have mac of each other resolved.	Pass	ARP resolution which is a functionality in L2 networks works properly.

T3 Result

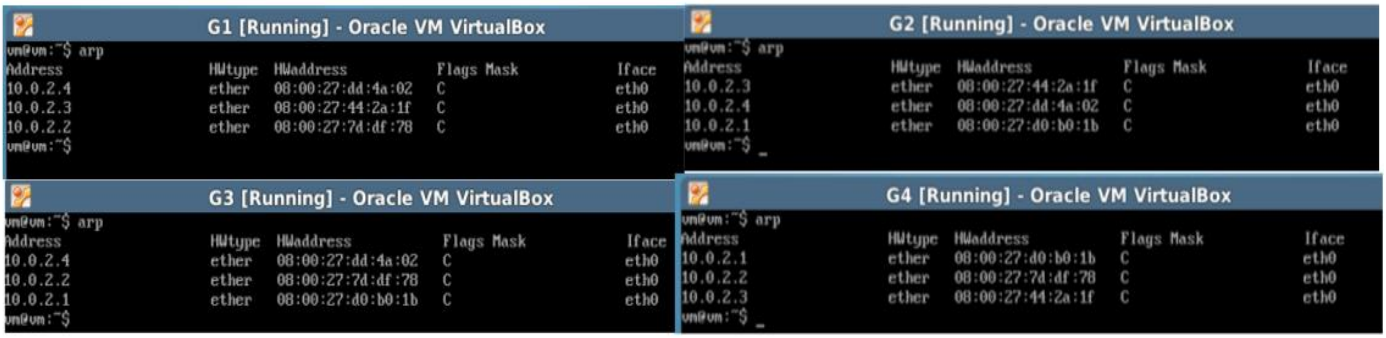
ARP tables at Tenant Blue VMs



ARP tables at Tenant Red VMs



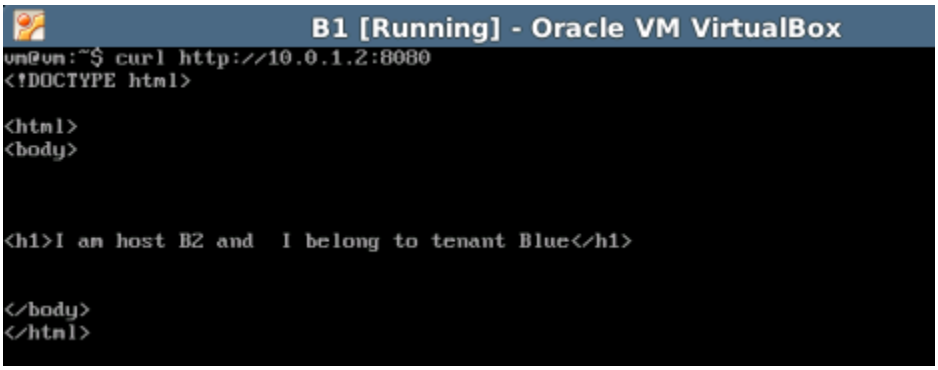
ARP tables at Tenant Green VMs



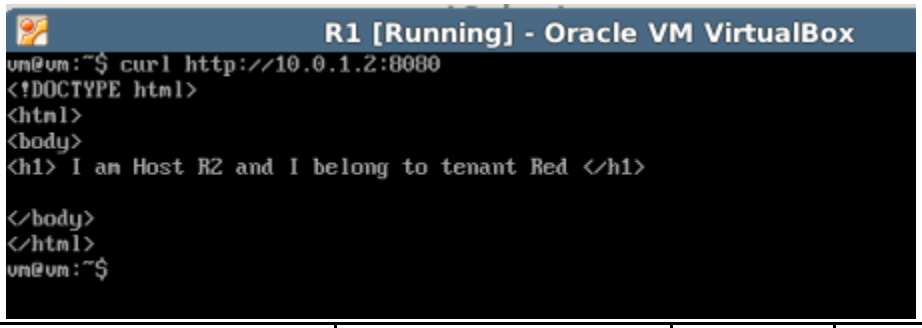
T4	Set up a HTTP web server at R2(IP: 10.0.1.2) and B2 (IP: 10.0.1.2) which would produce HTTP webpage saying, “I am Host R2 and I belong to Tenant Red” and “I am Host B2 and I belong to Tenant Blue” respectively when accessed. Access the webpage http://10.0.1.2:8080 from B1 (IP: 10.0.1.1) and R1(IP: 10.0.1.1)	Different Webpages are loaded at R1 and B1	Pass	Although tenant Blue and tenant Red are having the same IP subnet address space, they are isolated as two different VxLAN broadcast domains. Tenant isolation and address reuse is achieved

T4 Result

Webpage returned at B1

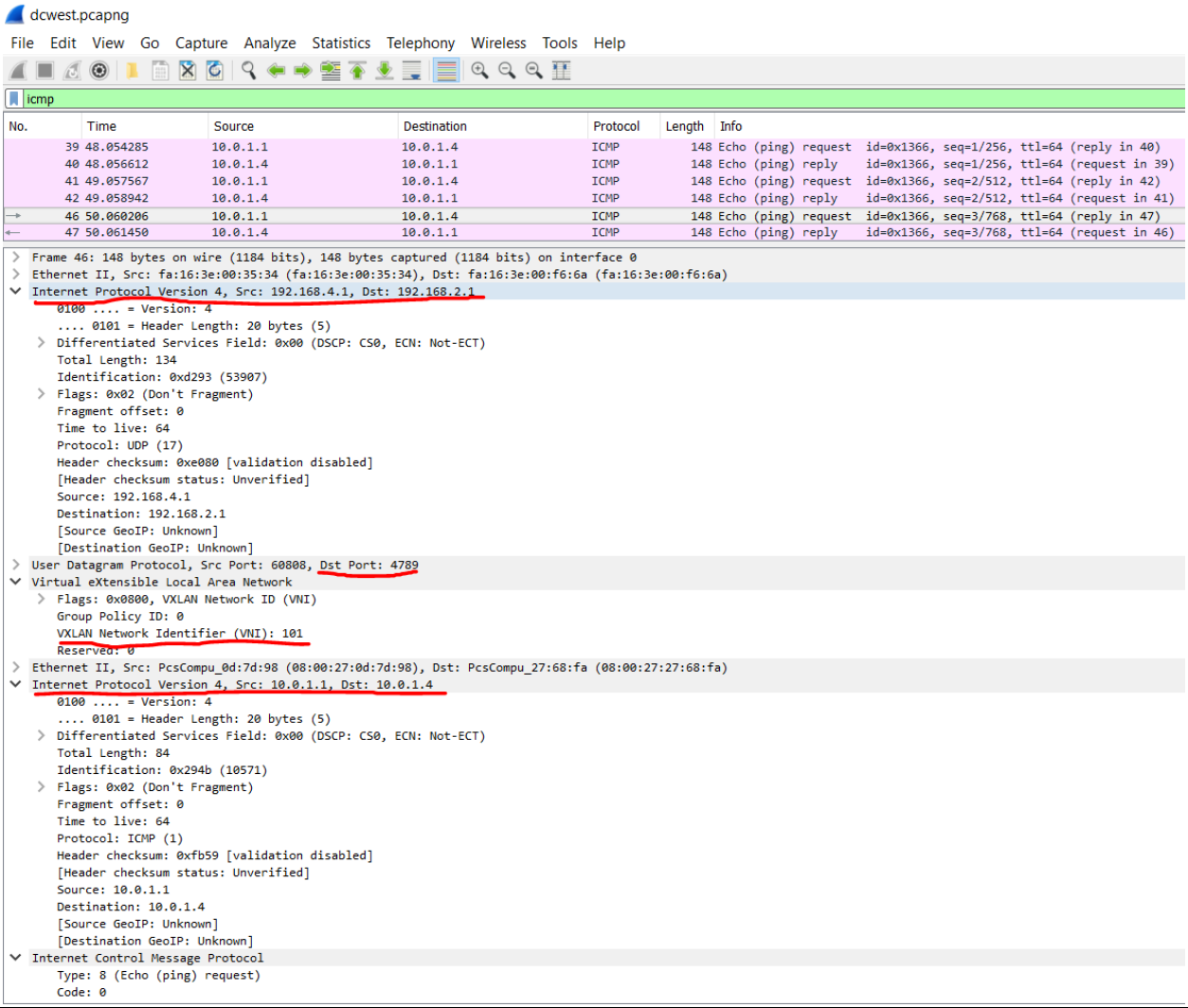


Webpage returned at R1



T5	Ping B4 (DC-East) from B1 (DC-West), setup a Wireshark packet capture at the output interface of DC-West. Observe the fields of ICMP packets between B4 and B1.	The source and destination IP addresses of the packet must be of the source VTEP and Destination VTEP respectively. The packet should have a UDP datagram with payload as the Ethernet frame generated by the VM. The VNI value should be 101.	Pass	The tunneling function is working as expected.

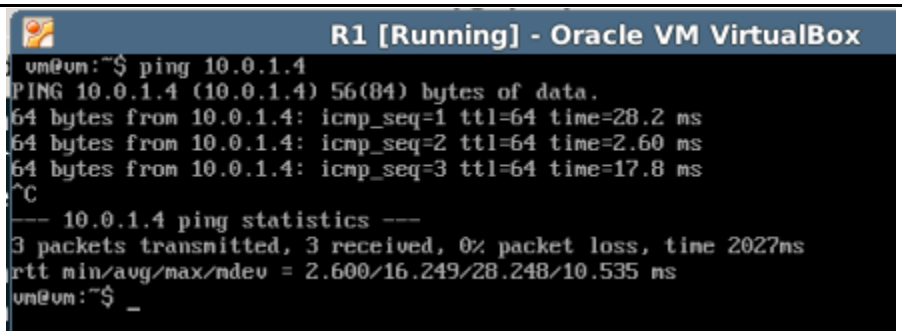
T5 Result



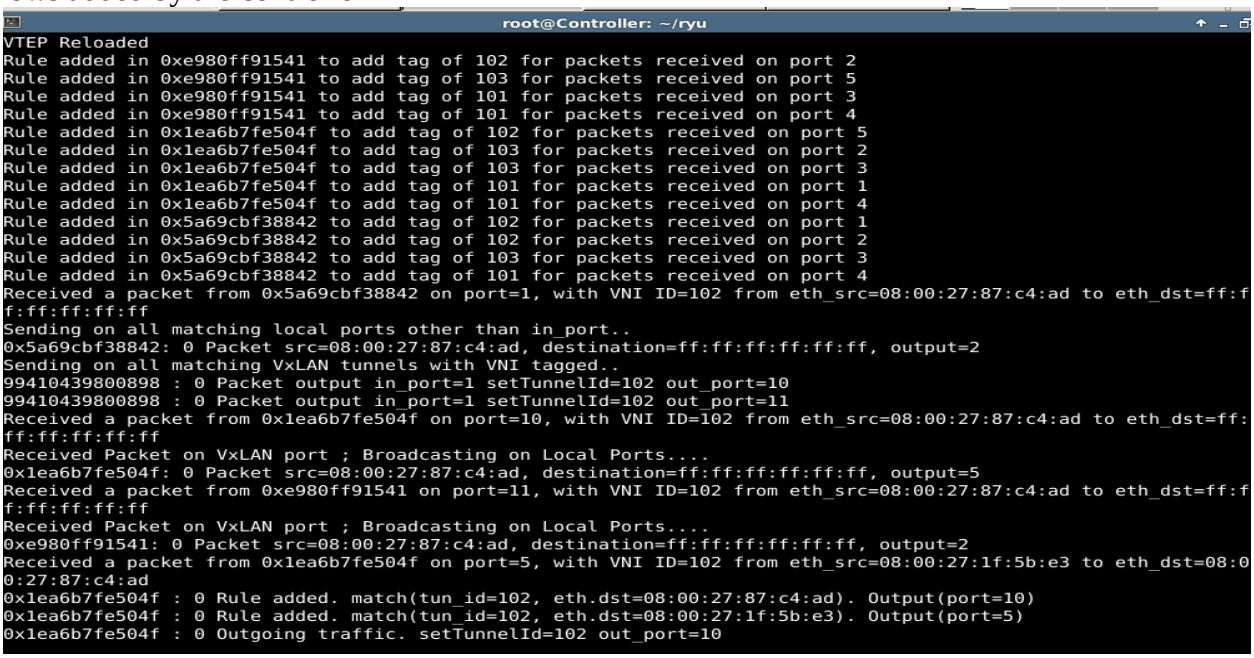
T6	Add a new Host VM (say R4 with IP: 10.0.1.4 at DC-East) at a new port for an existing tenant, update the database of the new addition. Ping this new VM from other existing VM (R1 at DC-West). Note that tenant Red did not have a host with IP 10.0.1.4 before this.	Ping is successful.	Pass	Functionality of adding a new host VM for an existing tenant is working.

T6 Results

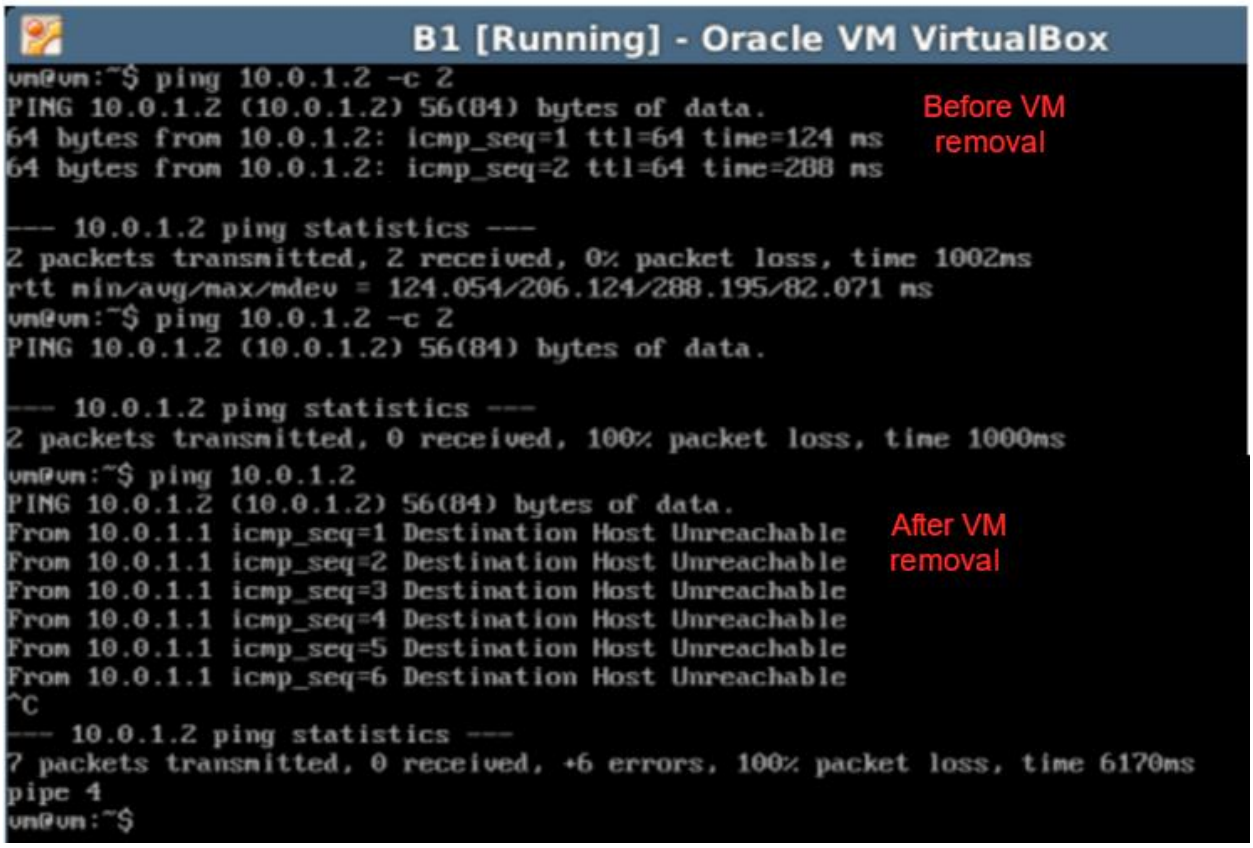
Successful Pings to R4 10.0.1.4



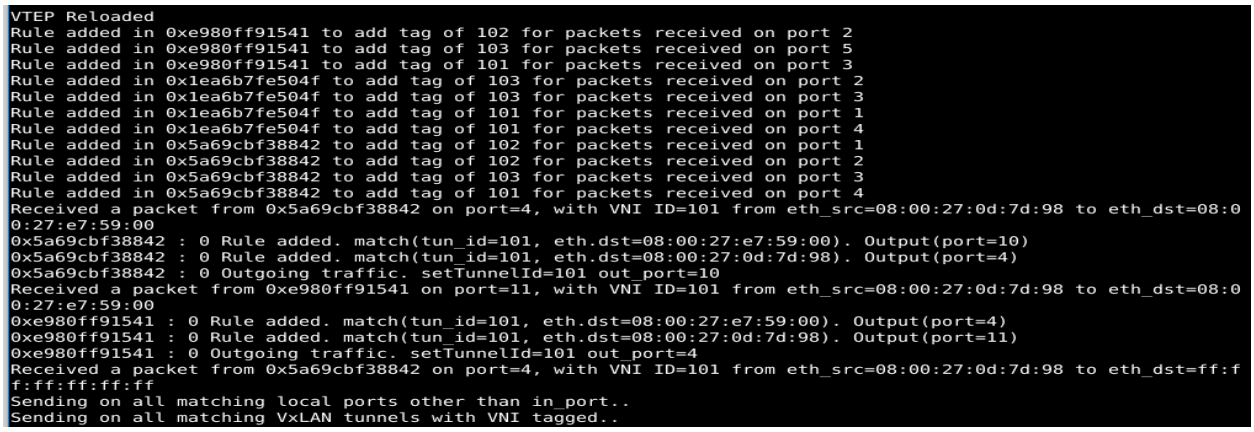
New flows added by the controller



T7	Delete a host VM for an existing tenant (B2 at DC-North with IP 10.0.1.2), update this deletion in the database. Try to ping the deleted host (from B1).	Ping fails with host unreachable ICMP error.		Functionality of deleting a host for a existing tenant is working.
T7 Result				

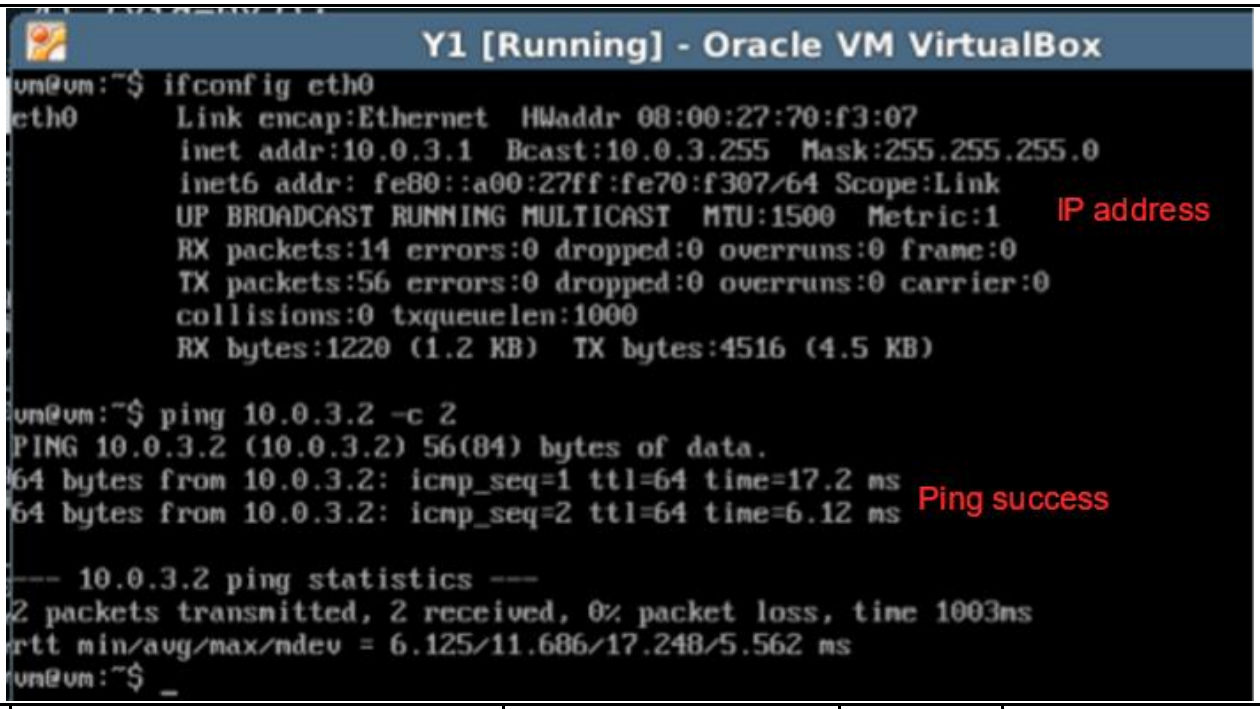


At the controller



T8	Add a new Tenant Yellow with host Y1 (IP: 10.0.3.1) at DC- West and Y2 (IP: 10.0.3.2) at DC-East. Update the addition in the database, ping Y2 from Y1	Ping is successful	Pass	Functionality of adding a tenant is working

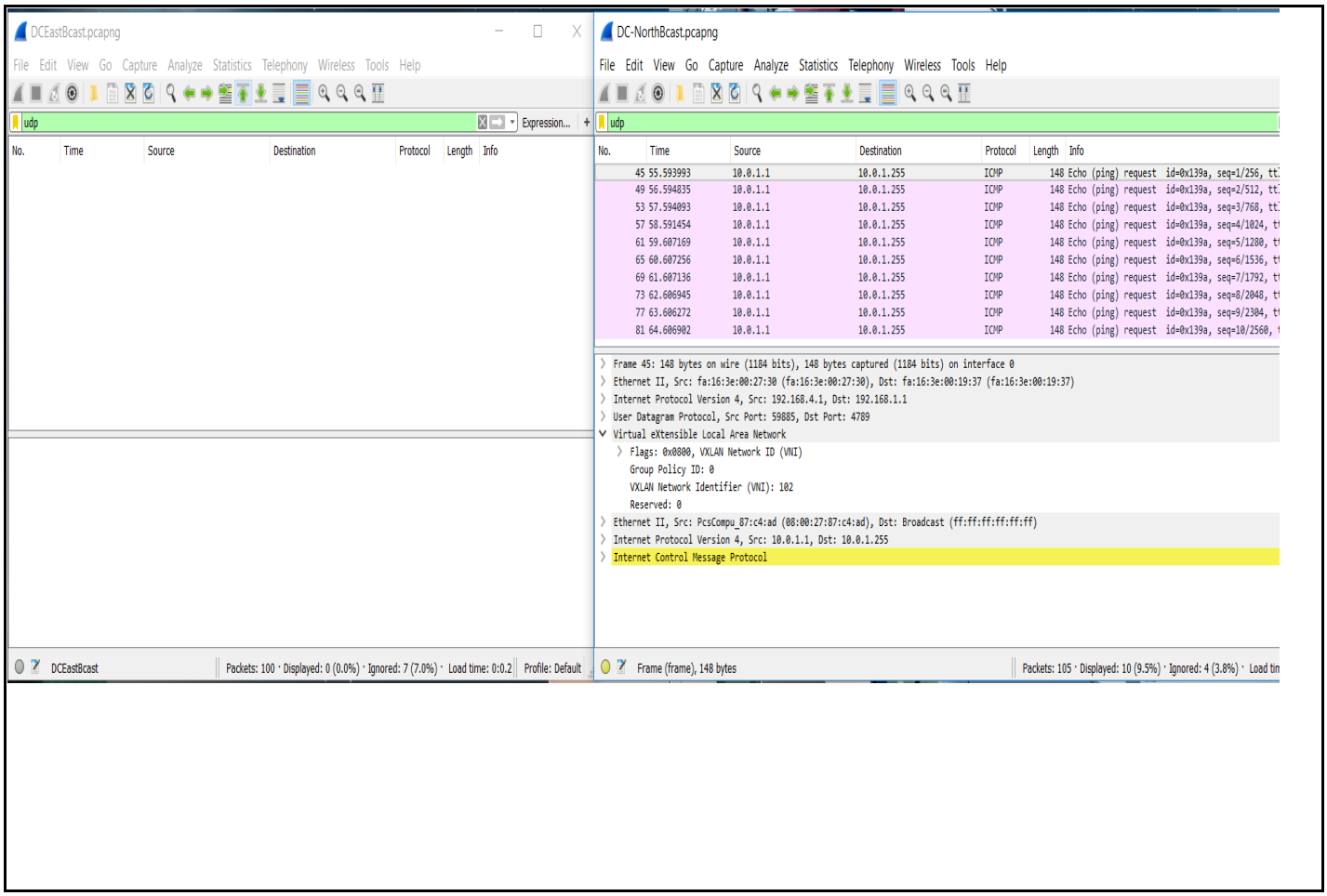
T8 Results



T9	Ping broadcast address from any red tenant VM, say R1 (DC-West). Capture (Wireshark) packets at DC-East and DC-North Interfaces.	The broadcast packet reaches only DC-North		Broadcasting which is achieved as multiple unicast is sent to a DC only if that tenant has presence there.

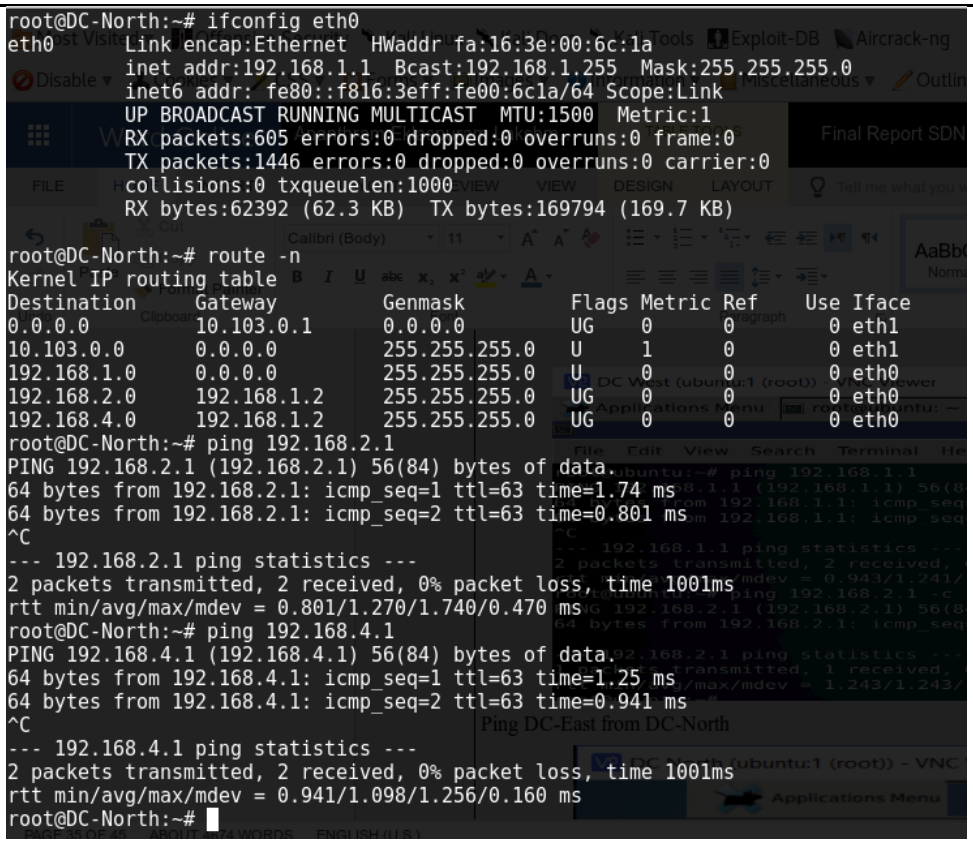
T9 Result

Packet captures at DC-North and DC-East

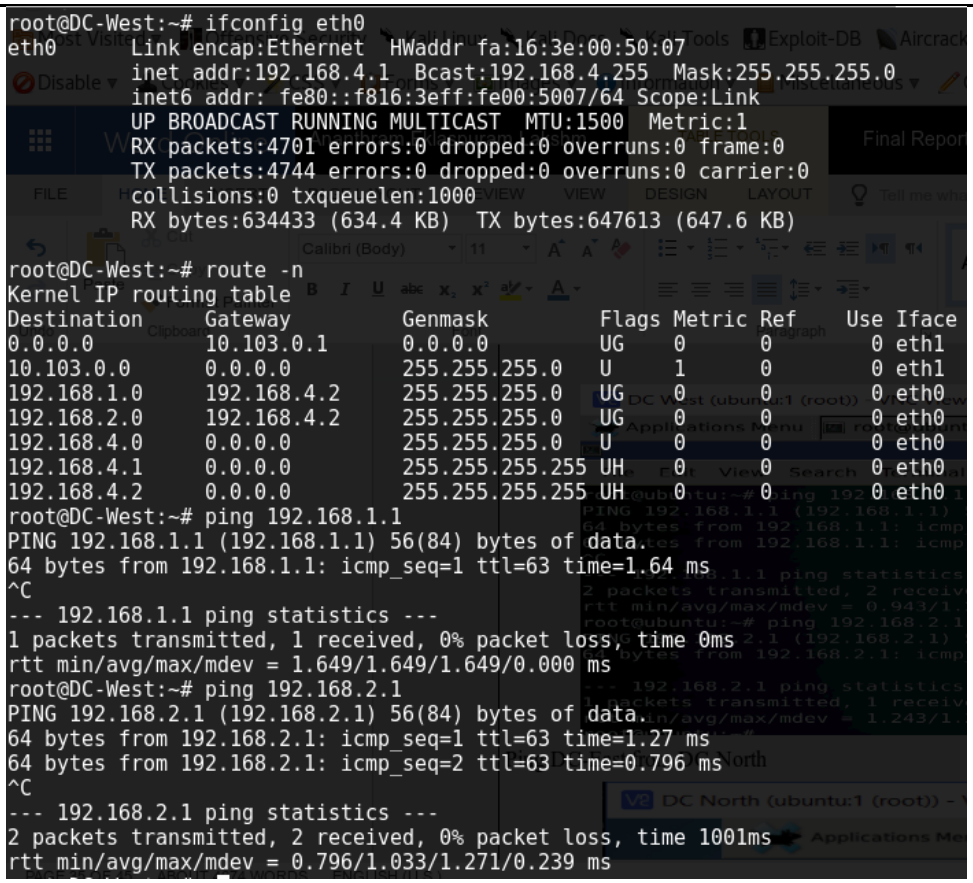


Testing of Traditional Approach

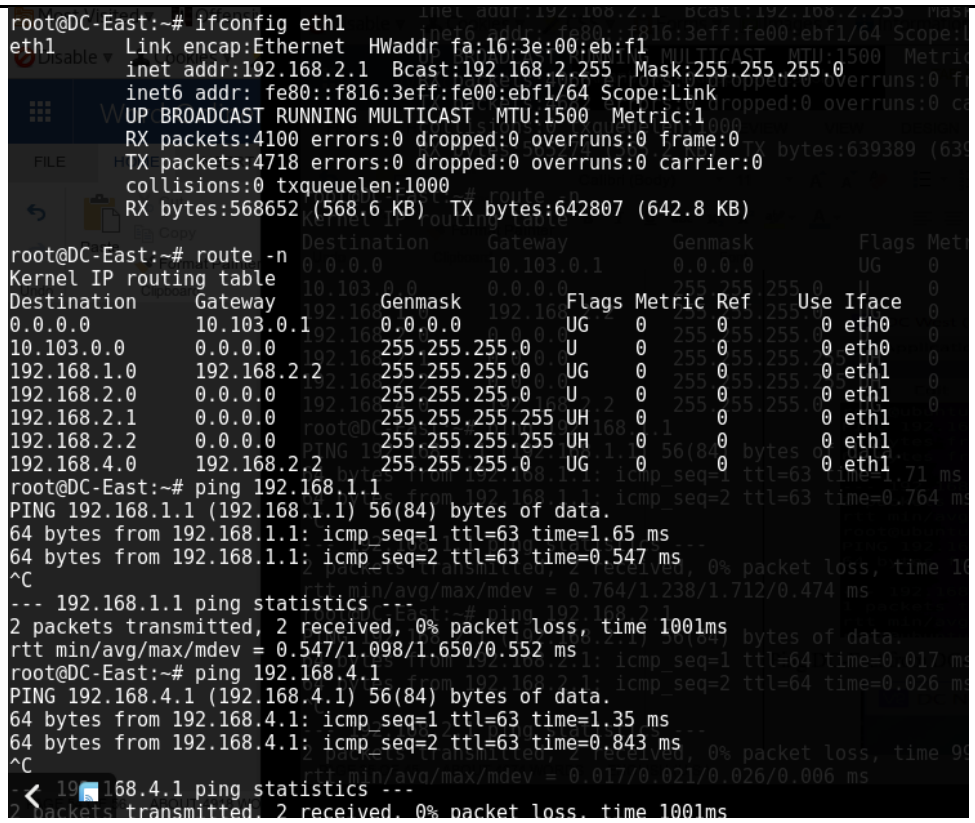
Test ID	Input and action to System	Expected Observation	Test pass/failed	Conclusion if passed
T1	Ping the DC-North and DC-East from DC-West. Ping DC-North from DC-East.	Pings are successful.		Connectivity between the Data Centers are ok. Underlay Network is operational.
T1 Results				
Ping From DC-North				



Ping from DC-West



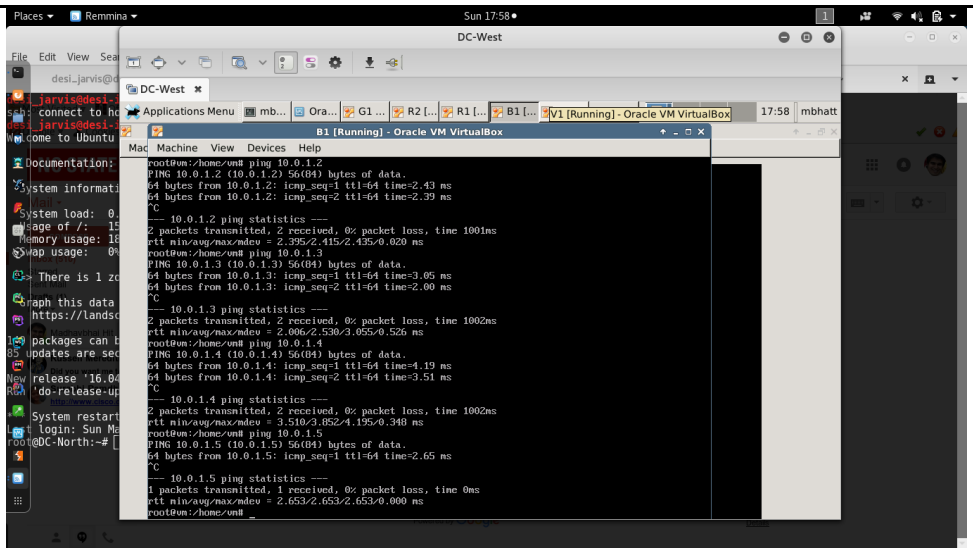
Ping from DC-East



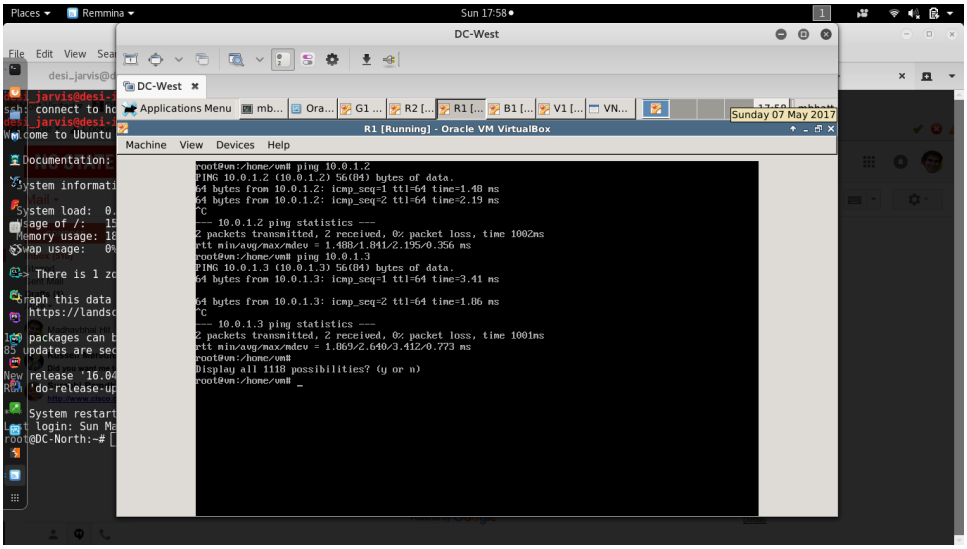
```
root@DC-East:~# ifconfig eth1
eth1      Link encap:Ethernet  HWaddr fa:16:3e:00:eb:f1
          inet addr:192.168.2.1  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::f816:3eff:fe00:ebf1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4100 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4718 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:568652 (568.6 KB)  TX bytes:642807 (642.8 KB)

root@DC-East:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.103.0.1      0.0.0.0         UG    0      0        0 eth0
10.103.0.0      0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.1.0     192.168.2.2     255.255.255.0   UG    0      0        0 eth1
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
192.168.2.1     0.0.0.0         255.255.255.255 UH    0      0        0 eth1
192.168.2.2     0.0.0.0         255.255.255.255 UH    0      0        0 eth1
192.168.4.0     192.168.2.2     255.255.255.0   UG    0      0        0 eth1
root@DC-East:~# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=63 time=1.65 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=63 time=0.547 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.547/1.098/1.650/0.552 ms
root@DC-East:~# ping 192.168.4.1
PING 192.168.4.1 (192.168.4.1) 56(84) bytes of data.
64 bytes from 192.168.4.1: icmp_seq=1 ttl=63 time=1.35 ms
64 bytes from 192.168.4.1: icmp_seq=2 ttl=63 time=0.843 ms
^C
--- 192.168.4.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
```

T2	Ping from host B1 to B2, B3, B4 and B5. Ping from host R1 to R2 and R3. Ping from G1 to G2, G3, G4.	Pings are successful.		Tenants in the same (virtual) L2 domain can communicate with each other. VxLAN service is working.

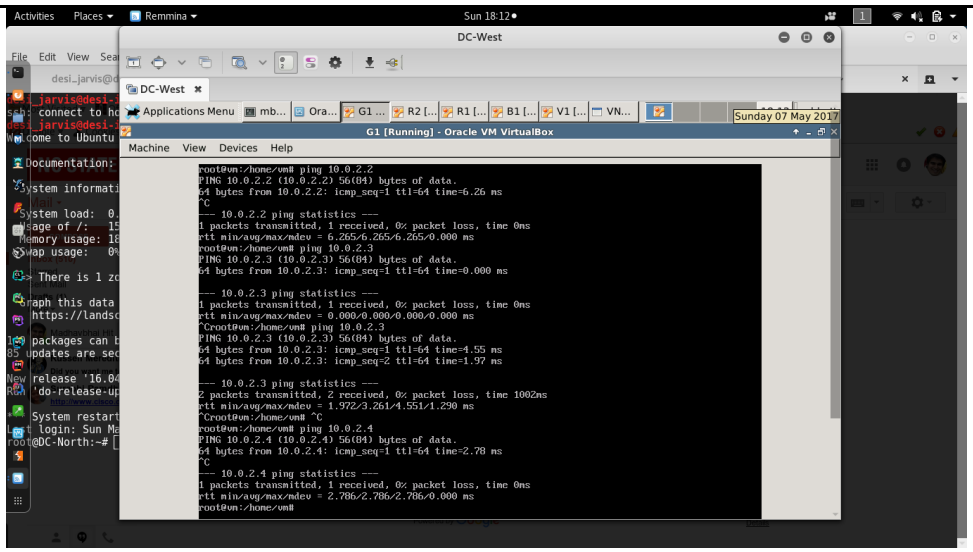
T2 Results
Ping from B1 to B2,B3,B4 and B5



Ping from R1 to R2 and R3



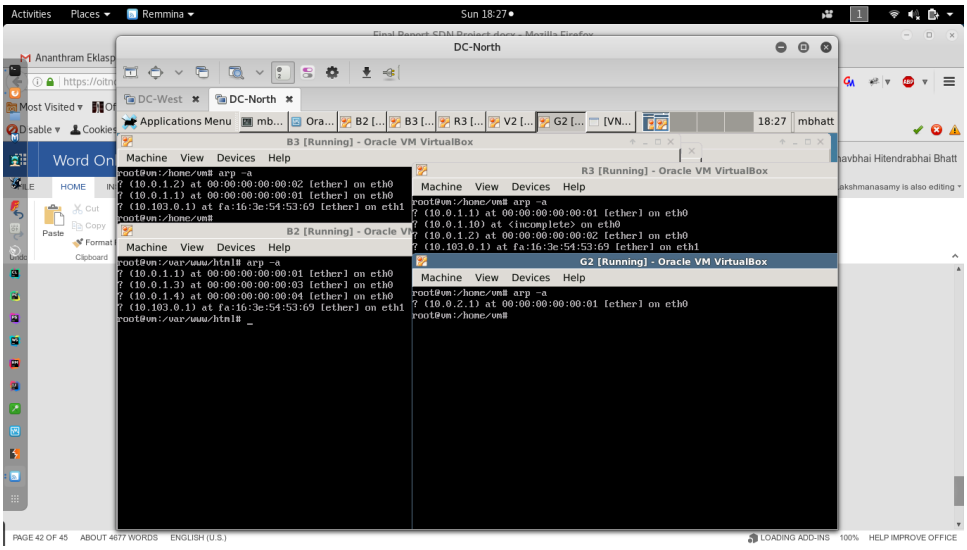
Ping from G1 to G2, G3 and G4



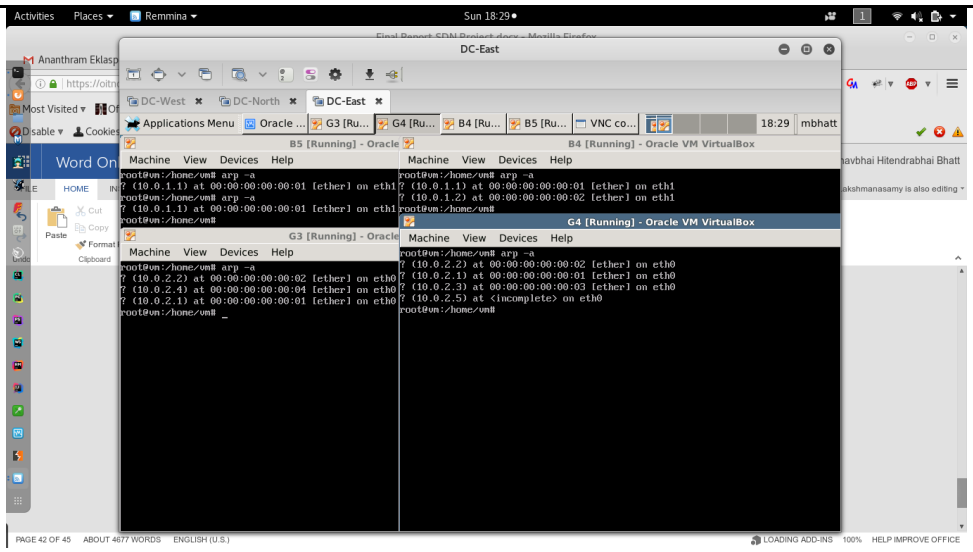
T3	Following test no 2 and check the ARP tables of the hosts VMs	The ARP for all the host (involved in pinging) has been resolved		ARP resolution which is a functionality in L2 networks works properly.

T3 Results

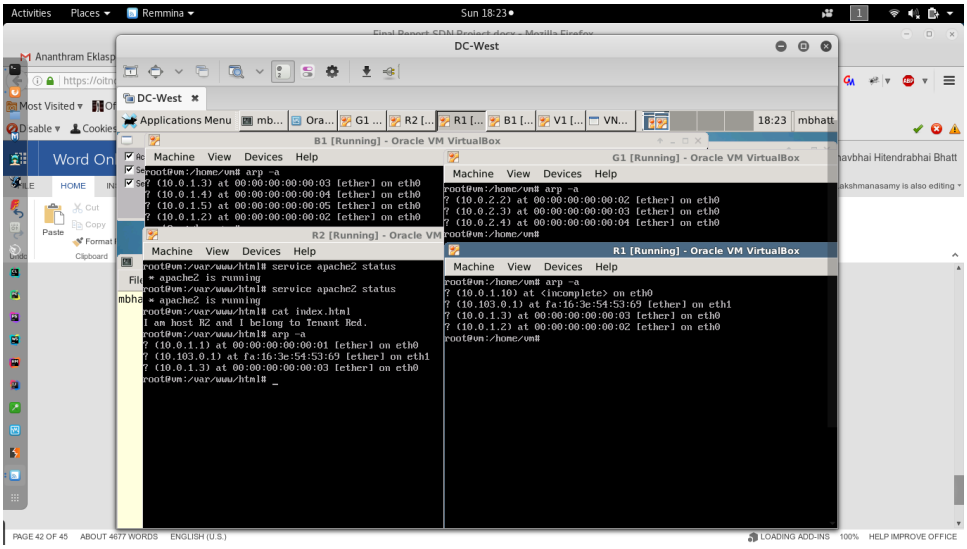
ARP table on DC-North



ARP table on DC-East

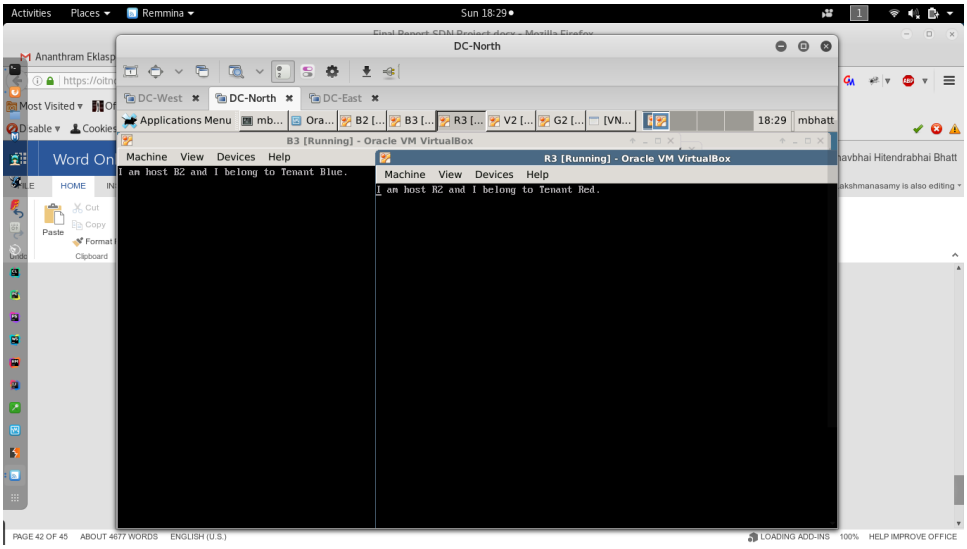


ARP table on DC-West

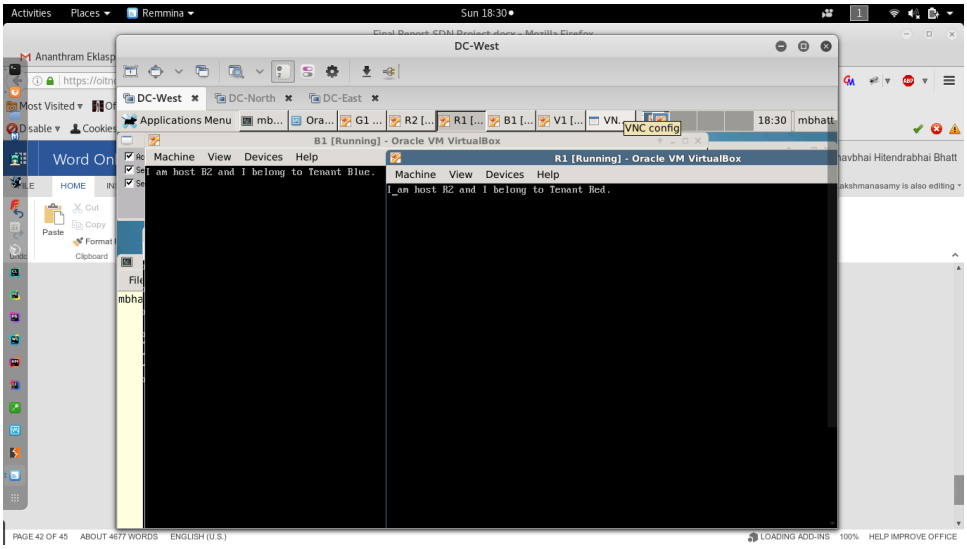


T4	Set up a HTTP web server at R2(IP: 10.0.1.2) and B2 (IP: 10.0.1.2) which would produce HTTP webpage saying “I am Host R2 and I belong to Tenant Red” and “I am Host B2 and I belong to Tenant Blue” respectively when accessed. Access the webpage (http://10.0.1.2:80) from R1 and B2	Different Webpages are loaded at R1 and B1		Although tenant Blue and tenant Red are having the same IP subnet address space, they are isolated as two different VxLAN broadcast domains. Tenant isolation and address reuse is achieved
T4 Results				

HTTP Connection on from DC-North



HTTP Connection on from DC-West

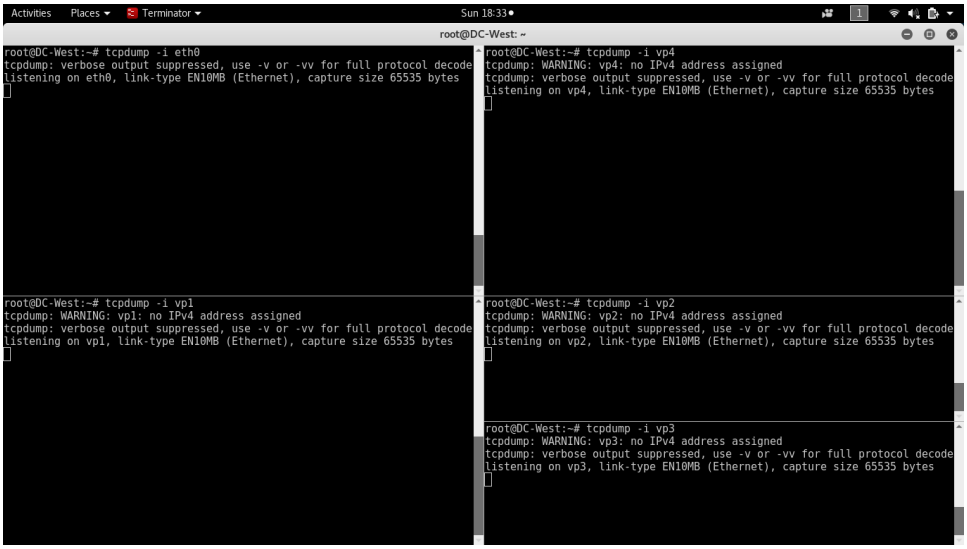


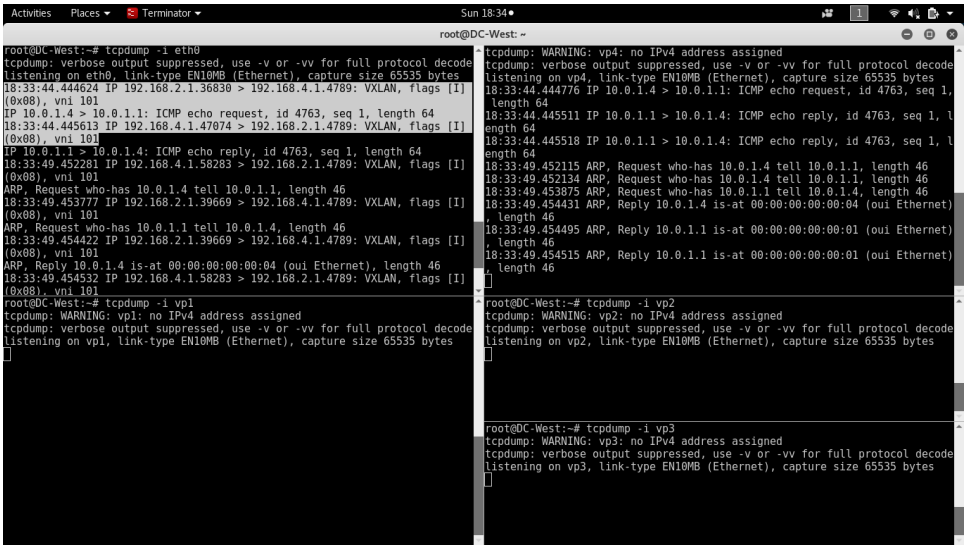
T5	Ping B4 (DC-East) from B1 (DC-West), setup a tcpdump packet capture at the output interface of DC-West.	The source and destination IP addresses of the packet must be of the source VTEP and Destination VTEP respectively, and the packet should have a UDP datagram with payload as the		The tunneling function is working as expected.

		Ethernet frame generated by the VM. The VNI value should be 101.		

T5 Results

tcpdump on eth0, vp4, vp1, vp2, vp3 before and after ping

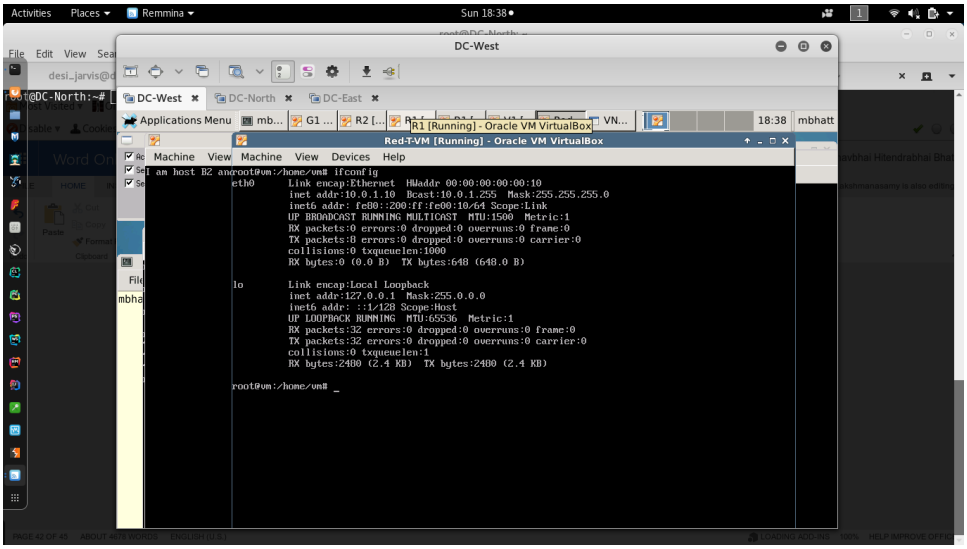


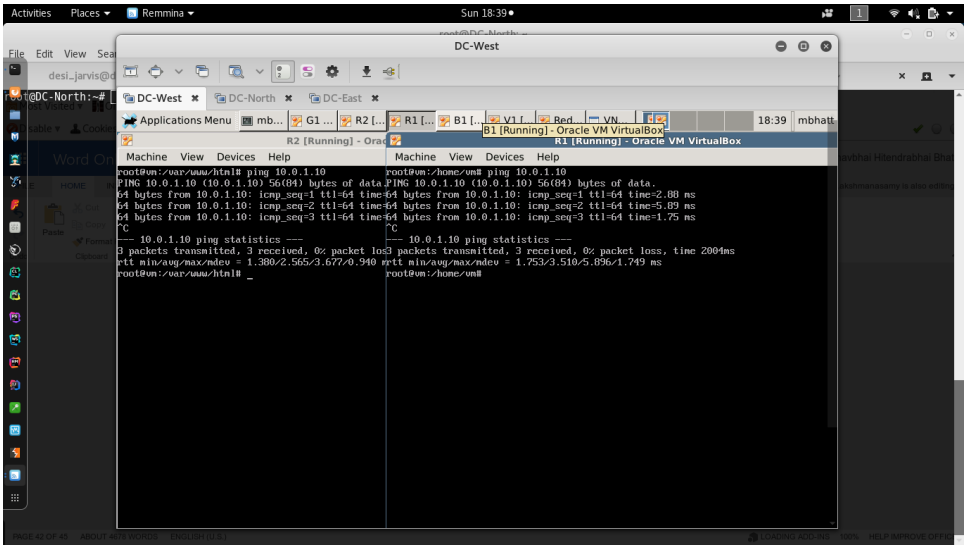


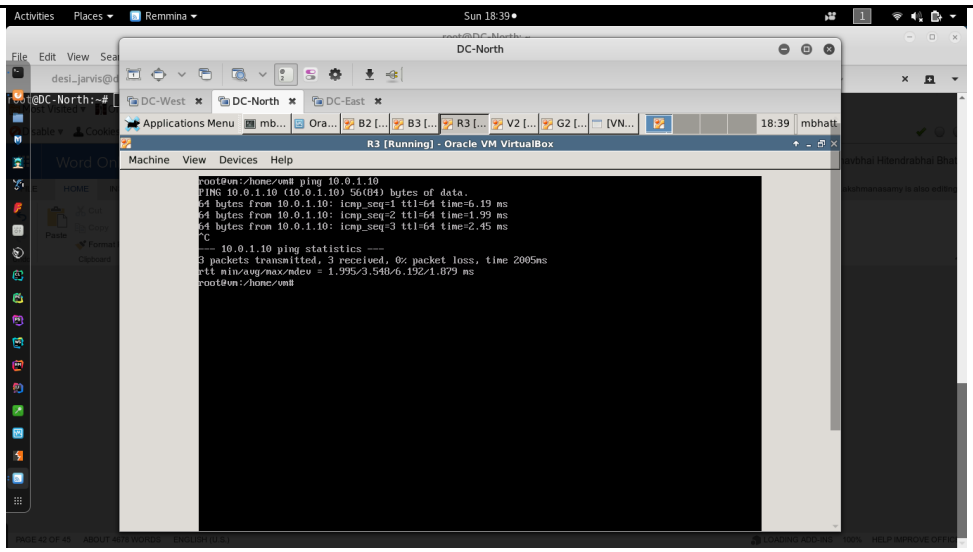
T6	Add a new VM and check if that VM is reachable by other VMs.	Ping is successful		Functionality of adding a new host for a tenant is working.

T6 Results

Ping from R1,R2 and R3 to Red-T-VM






Activities
Places
Remmina
Sun 18:39
DC-North
DC-West
DC-North
DC-East
Applications Menu
R3 [Running] - Oracle VM VirtualBox
Machine
View
Devices
Help
root@vm:/home/vm# ping 10.0.1.10
PING 10.0.1.10 (10.0.1.10) 56(84) bytes of data.
64 bytes from 10.0.1.10: icmp_seq=1 ttl=64 time=6.19 ms
64 bytes from 10.0.1.10: icmp_seq=2 ttl=64 time=1.99 ms
64 bytes from 10.0.1.10: icmp_seq=3 ttl=64 time=2.45 ms
^C
--- 10.0.1.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 1.995/3.540/6.192/1.879 ms
root@vm:/home/vm#

4.2 DEMO PLAYBOOK

Video Demo: https://goo.gl/YfPUAq

Demo ID	Demonstrator Action	Expected Observation	Conclusion
D0 (only SDN approach)	Check the Flow Tables at all the OVS Bridges by entering "ovs-ofctl show br0"	The flow tables are empty	No flows are installed initially and all subsequent additions are going to happen because of the steps we are going to carry out.
D1	Ping B2 from B1. Capture packets at port eth0 of DC-WEST and vp4 of DC-WEST	Packet at eth0 of DC-West is VxLAN encapsulated and packet at vp4 of DC-West is not VxLAN encapsulated	VxLAN encapsulations are working as desired.
D2	Set up a HTTP web server at R2(IP: 10.0.1.2) and B2 (IP: 10.0.1.2) which would produce HTTP webpage saying, "I am Host R2 and I belong to Tenant Red" and "I am Host B2 and I belong to Tenant Blue" respectively when accessed. Access the webpage (http://10.0.1.2:8080) from R1 and B1 using command "curl http://10.0.1.2:8080".	Different Webpages are loaded at R1 and B1.	Although tenant Blue and tenant Red are having the same IP subnet address space, they are isolated as two different VxLAN broadcast domains. Tenant isolation and address reuse is achieved.
D3	**VM Addition:** Add a new Host VM (say R4 with IP: 10.0.1.4 at DC-East) for an existing tenant at DC. Ping this new VM from other existing VM (R1 at DC-West). **VM Deletion:** Delete a VM of a tenant (B2 at DC-North with IP 10.0.1.2). Count the number of admin tasks involved in both traditional approach and SDN approach. Try to ping the deleted host (from B1).	Addition and Deletion of VMs is demonstrated Number of steps is less in SDN approach.	SDN solution is more agile.

D4	**Tenant Addition:** Add a new tenant in all the Datacenters in the Traditional and SDN approach. New Tenant added: Yellow with hosts Y1 (IP: 10.0.3.1) at DC- West and Y2 (IP: 10.0.3.2) at DC-East. Update flows in configs in the both approaches. Ping Y2 from Y1. **Tenant Deletion**: Remove a tenant from all datacenters in both approaches. Count the Number of admin tasks involved in both cases. Delete the previously added tenant Yellow by updating the configs in both approaches.	For both addition and deletion of a tenant the number of steps executed in the SDN approach will be lesser.	SDN solution is more agile.

5. Self-Study Plan

Plan:

S.No	Base Case	Characteristics to observe	Range of Scenarios to Observe
1	ARP Resolution in the Traditional Approach and SDN Approach (or General Flow set up time)	• Time • No of packets dropped	• When Destination is on same VTEP and destination is on remote VTEP • Traffic Load – High load and Low Load (Network)
2	Admin Configuration Overhead in Traditional and SDN approach	• No of steps executed. • Agility of Network operation	• Number of additional VMs for a Tenant • Number of additional tenants

Self-Study Execution

Self-Study 1

Parameter: Ping Response time in the Traditional Approach and SDN Approach (or General Flow set up time)

Characteristics to observe:

- Time taken for ping response
- No of packets dropped

Range of Scenarios:

Note, for the SDN approach all test were started with no flows in the OvS

1: Location of destination

Destination on same DC

Traditional approach

- R1 to R2 : 1.48 ms for first packet and average of 1.29 ms for consecutive packets
- B2 to B3 : 1.90 ms for first packet and average of 1.22 ms for consecutive packets
- B4 to B5 : 3.95 ms for first packet and average of 2.54 ms for consecutive packets
- G3 to G4 : 1.79 ms for first packet and average of 1.61 ms for consecutive packets

SDN approach

- R1 to R2 : 15 ms for first packet and average of 5.3 ms for consecutive packets
- B2 to B3 : 50 ms for first packet and average of 26 ms for consecutive packets
- B4 to B5 : 9.67 ms for first packet and average of 2.5 ms for consecutive packets
- G3 to G4 : 9.75 ms for first packet and average of 1.75 ms for consecutive packets

Destination on different DC

Traditional approach

- R1 to R3: 3.41ms for first packet and average of 1.73 ms for consecutive packets
- B1 to B2: 2.43ms for first packet and average of 2.27 ms for consecutive packets
- B1 to B3: 3.05ms for first packet and average of 1.88 ms for consecutive packets
- B1 to B4: 4.19ms for first packet and average of 2.36 ms for consecutive packets
- B1 to B5: 2.65ms for first packet and average of 2.65 ms for consecutive packets
- G1 to G2: 6.26ms for first packet and average of 2.25 ms for consecutive packets
- G1 to G3: 4.55ms for first packet and average of 3.22 ms for consecutive packets
- G1 to G4: 2.78ms for first packet and average of 1.94 ms for consecutive packets

SDN approach

- R1 to R3: 27 ms for first packet and average of 9 ms for consecutive packets
- B1 to B2: 300 ms for first packet and average of 11.5 ms for consecutive packets
- B1 to B3: 63.8 ms for first packet and average of 20.5 ms for consecutive packets
- B1 to B4: 14.5 ms for first packet and average of 3.34 ms for consecutive packets
- B1 to B5: 19 ms for first packet and average of 7 ms for consecutive packets
- G1 to G2: 41.1 ms for first packet and average of 9.2 ms for consecutive packets
- G1 to G3: 6.79 ms for first packet and average of 3.2 ms for consecutive packets
- G1 to G4: 20.2 ms for first packet and average of 2.6 ms for consecutive packets

Comments

- We see in that in the traditional approach we have a consistent response times and in the SDN approach apart from having a considerable delay for the first packet, the following response times are highly varying. This variance is attributed to the processing delays at the SDN controller.

2. Traffic Load in the Network

Low Load (Base case)
Calculate ping response time B1 and B2 without any other modification to the topology

Traditional Approach

- No packets are dropped without load. Average ping response time is 2.27 ms.

SDN Approach

- Average response time was 8.5 ms with no packet loss

Medium Load

Set up Iperf between R1 and R3, and then calculate the ping response time between B1 and B2

Traditional Approach

- No packets are dropped with medium load. Average ping response time is roughly same 2.27 ms

SDN Approach

- Initial 3 packets are dropped Average ping response time is 12. 3 ms

Heavy load

Set up Iperf between R1-R3 and G1-G3, and then calculate the ping response time between B1 and B2

Traditional Approach

- No packets dropped but response time is 4.92 ms.

SDN Approach

- Initial 9 packets are dropped, Average ping response time is 20 ms

Comments

- We see a degradation is performance with increasing load in the SDN approach and we don't see a big decrease in the performance of traditional approach.
- The performance of the SDN approach is largely dependent on the way the controller logic is implemented, by filtering the duplicate packet events at the controller, we can improve current performance.

Self-Study 2

Parameter: Admin Configuration overhead

Characteristics to observe:

- Number of commands executed to a result
- Agility of Network operation

Range of Scenarios:

1. **Add one new VM of an existing tenant**

   Traditional approach

   - 2 commands per VM per VTEP. It means if we add three VMs on every VTEP we need to execute 6 commands
   - Every VTEP needs to be configured.

   SDN approach

   - Update the data structure in the config file for that particular VTEP where new VM is added. Only if, at VTEP location where the VM is added did not have a VM for that tenant, then update vni_vxlanport variable at other VTEPs

2. **Delete one VM of existing Tenant**

   Traditional approach

   - 2 commands per VM per VTEP. It means if we delete three VMs on every VTEP we need to execute 6 commands
   - Every VTEP needs to be configured.

   SDN approach

   - Update the data structure in the config file for that particular VTEP where new VM is added. If the VM deleted at the VTEP was the only VM for a tenant at that location, then update the vni_vxlanport variable at other VTEPs

3. **Add a Tenant**

   Traditional approach

   - 2 commands per VM per VTEP. It means if we add one tenant with three VMs on every VTEP we need to execute 6 commands.
   - Every VTEP that has tenant from that VM needs to be configured.

   SDN approach

   - Add a new vni_locport and vni_vxlanport object for the tenant using the VNI assigned at each of the VTEP, where the tenant is added

4. **Delete a New Tenant**

   Traditional approach

- 2 commands per VM per VTEP. It means if we delete a tenant with three VMs on every VTEP we need to execute 6 commands
- Every VTEP needs to be configured.

SDN approach

- Delete the vni_locport and vni_vxlanport object for the tenant using the VNI assigned at each of the VTEP, where the tenant is deleted from.

Comments:

- Although it doesn't seem like there is a lot difference in the admin configurations efforts involved, but when we are scaling up with lot more VTEPs and Tenants involved in the design, the SDN approach will be a lot easier to configure and lot more agile in achieving the desired change in the network.

6. Reflection

6.1. Per-member learning experiences

Ananthram: I understood how overlay technology like VxLAN can be used for real world applications. I got to learn to use REST APIs and usefulness of an API in general for application developing. I understood the power a network designer or admin gets while using OpenFlow and Virtual switches like OvS. I learnt to write a SDN controller application and how to model a learning switch when using a SDN controller.

Shishir: I learnt the true definition of an overlay network and how it can be operationalized in an SDN environment. I also learnt the nitty-gritties of working with an API like Ryu, and methodical testing methods, among many other skills.

Madhav: I learnt how to configure VxLAn on an OpenVSwitch. Besides, I learnt how to install VNC server on ubuntu. It is an added value to my experience as a network administrator.

6.2. What went as expected

1. The overall design worked to a degree that was satisfactory.
2. The choice of using RYU as the SDN controller turned out to be very helpful because of the extensive documentation and examples that were available for us, which helped in speeding up our initial learning.
3. We were skeptical about how we would go about adding tenants, but that part worked as expected.

6.3. What went different from expected

1. Initially, we had planned use a Database server for holding all the tenant information but after several attempts to interface our RYU application to access the database server, we decided to write everything on file locally and use that for any changes in the tenant configuration.

2. When it came to deletion of VMs and tenants, our approach was to delete the entire table and refresh it with the updated JSON file. However, this is not a very efficient approach, as a lot of processing power is wasted in doing so, and could potentially slow down operations in a large-scale environment. Thus, we were unable to come up with an efficient deletion strategy. Our initial plan was to have a separate thread running along the RYU app which will act as a User Interface for any additions or deletions but because the way RYU works, it supports only one thread which waits for events and as a result our thread was

blocking all of RYU's event processing. Because of lack of time and lack of expertise in working with threads and events in our team, we decided to get the project working by a different approach, i.e. checking for changes in a file.

3. Although there was an initial hiccup in setting up the VMs which represented the tenants, we were able to set up the topology for testing with several attempts. Some of the tweaks we had to do to get the topology running were using Extra Large nodes in ExoGeni, Using Virtual Network Computing Servers at the ExoGeni nodes instead of x11 forwarding. Using Virtualbox, installing Ubuntu, and booting up was taking a very long time. Therefore, we used Ubuntu Server images for the VMs instead.

References

[1] O. Org, The Linux Foundation, [Online]. Available: http://openvswitch.org/.

[2] N. U. o. Oregon. [Online]. Available: https://nsrc.org/workshops/2014/nznog-sdn/raw-attachment/wiki/WikiStart/Ryu.pdf.

[3] O. N. Foundation, "Openflow," [Online]. Available: https://www.opennetworking.org/sdn-resources/openflow.

[4] Cisco, "Cisco VxLAN white paper," [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-729383.html.

[5] A. Networks, "Arista Networks VXLAN White Paper".

[6] Howtoforge, "Install a vncserver," [Online]. Available: https://www.howtoforge.com/how-to-install-vnc-server-on-ubuntu-14.04.

[7] D. Mahler, "VXLAN overlay networks with Open vSwitch," [Online]. Available: https://www.youtube.com/watch?v=tnSkHhsLqpM.

[8] B. Sailsbury, "Network Static Blog on VxLAN OvS setup," [Online]. Available: http://networkstatic.net/configuring-vxlan-and-gre-tunnels-on-openvswitch/#!prettyPhoto.

[9] olegslavkin, "Github Gist Instructions to install Ryu in Ubuntu 14.04," [Online]. Available: https://gist.github.com/olegslavkin/e01ccc1835396402dc2f.