

Mobile Computing

Category A - Group 3

File hosting / File sharing - VXLAN

Submitted By:

Name

MIR MEHEDI HASAN RAYHAN

Matriculation Number

1324345

Examiner:

Date of Submission:

Contents

1. Introduction	3
2. Elements and Technology	3
3. Implementation and Demonstration	5
3.1 Network Topology	5
3.2 Implementation	7
3.3 Demonstration.....	13
4. Discussion and Problems.....	20
5. Improvemnets and implementaion	20
6. Conclusion	20
7. References	21

1. Introduction

The objective of project is to access the file server from access network through a secure network tunnel. There are three access networks created in our network topology. One host from each access network can communicate with each other through the secure communication channel as like a local network and this can be achieved by the VXLAN. But the access between the hosts in the same access network segment is restricted by packet filtering.

2. Elements and Technology

For achieving this we are using multiple technology and entities. That has been described here.

ContainerNet: Containernet is a fork of the famous Mininet network emulator and allows to use Docker containers as hosts in emulated network topologies. This enables interesting functionalities to build networking/cloud emulators and testbeds. Besides this, it is actively used by the research community, focusing on experiments in the field of cloud computing, fog computing, network function virtualization (NFV), and multi-access edge computing (MEC).

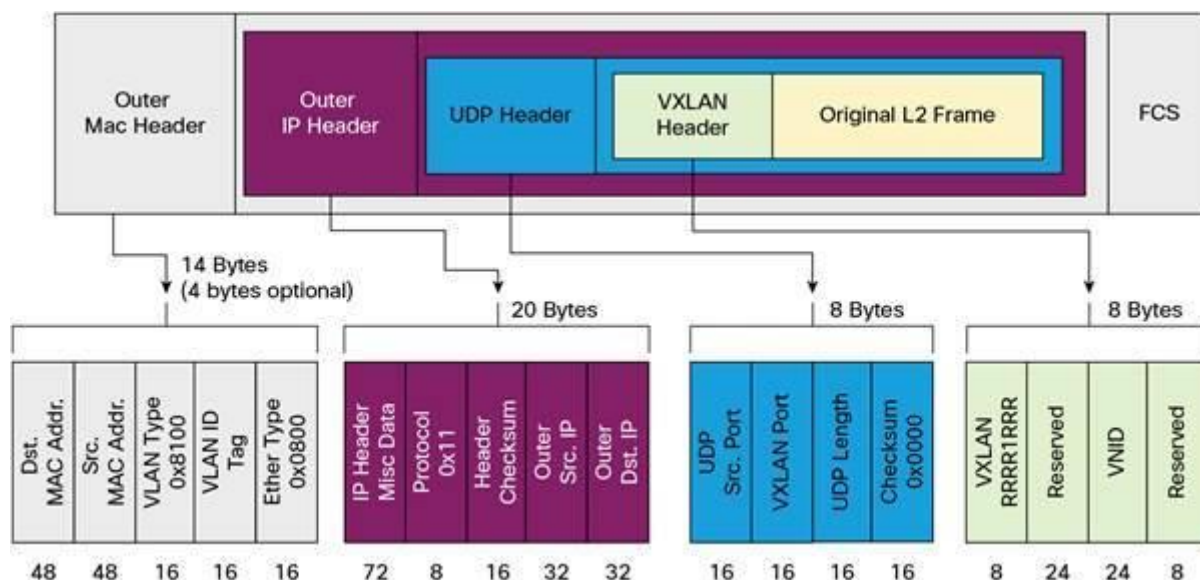
Wireshark: Wireshark is a network packet analyzer and presents captured packet data in as much detail as possible. It is a software that analyses packets sent throughout a network. It will display each packet and the details within the packet. It used for analyzing the network and to troubleshoot an issue if they see a problem. It has many capabilities, some of the key features are that it can catch packets in a real-time network. It can also save packets and the packets are laid out on a very clear GUI.

Quagga: Quagga is a routing software package that provides TCP/IP based routing services with routing protocols support such as RIPv1, RIPv2, RIPng, OSPFv2, OSPFv3, IS-IS, BGP-4, and BGP-4+. Quagga also supports special BGP Route Reflector and Route Server behavior. In addition to traditional IPv4 routing protocols, Quagga also supports IPv6 routing protocols. Quagga uses an advanced software architecture to provide a high quality, multi-server routing engine. Quagga has an interactive user interface for each routing protocol and supports common client commands.

OSPF: OSPF is, mostly, a link-state routing protocol. In contrast to distance-vector protocols, such as RIP or BGP, where routers describe available paths (i.e., routes) to each other, in link-state protocols routers instead describe the state of their links to their immediate neighboring routers. Each router describes their link-state information in a message known as an LSA (Link State Advertisement), which is then propagated through to all other routers in a link-state routing domain, by a process called flooding. Each router thus builds up an LSDB (Link State Database) of all the link-state messages. From this collection of LSAs in the LSDB, each router can then calculate the shortest path to any other router, based on some common metric, by using an algorithm such as Edgser Dijkstra SPF (Shortest Path First).

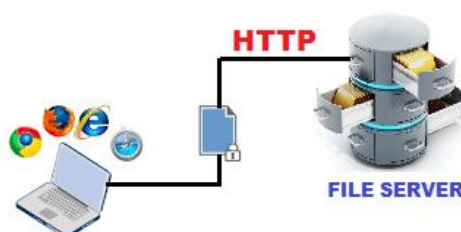
VXLAN: VXLAN stands for Virtual Extensible Local Area Network which is one of the proposed encapsulation protocols which helps tunneling of layer 2 over layer 3 infrastructure. This will help increase the scalability of cloud computing environment while logically separating cloud applications with tenants. One of the major characteristics of VXLAN is that it uses larger naming

space as compared to regular VLAN. A traditional 802.1q VLAN uses 12 bit space which would only allow $2^{12}=4096$ users in a segment. Whereas VXLAN uses 24 bit space that allows for over $2^{24}=16,777,216$ VXLAN identifiers, addressing the problem, where more than a million users can be a part of a same network within the cloud.



Open vSwitch: Open vSwitch (OVS) is an open source OpenFlow capable virtual switch that is typically used with hypervisors to interconnect virtual machines within a host and between different hosts across networks. OVS ties together all the virtual machines within a host residing on a server, which makes it critical component in many SDN deployments. Using OVS for multi-tenant network virtualization is considered a core element of various datacenter SDN deployments. OVS supports many traditional switch features such as VLAN tagging and 802.1q trunking, Standard Spanning Tree Protocol, LACP, port mirroring (SPAN/RSPAN), Flow Export (netflow, sflow, etc), tunneling (GRE, VXLAN, IPSEC), QoS control.

Fileserver: In computer network, file server is a super performing computer system that has responsible for storing and fetching all types of files (audio file, images, video, database, and other documents), and these files are used by all client machines which are linked over the network. A file server allows users to transfer all files over the entire network without using any physical medium of file transfer such as pen drive, floppy diskette or other external storage media. We can set up any computer as host that plays role as a file server. In the other word, file server may be a simple computer system that has abilities to send and retrieve all requests for files over the computer network.



3. Implementation and Demonstration

3.1 Network Topology

This network topology is File hosting / file sharing Using VXLAN. Access the file server (D2) from access network (192.168.3.0/24) through the VXLAN tunnel In this topology (Fig-1) we are using 3 access networks where one is used for placing servers (D1 and D2) and used OSPF routing protocol.

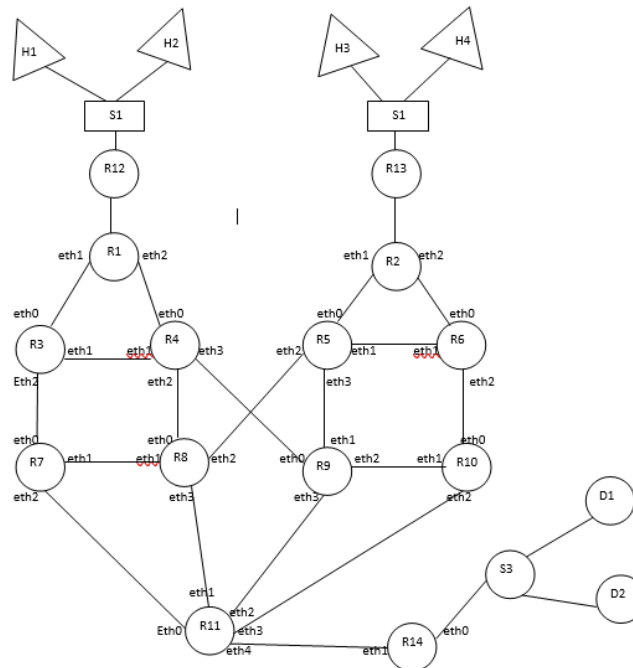


Fig-1: Network Topology

In access network 1 (S1-R12) there are two hosts, and we need to refrain these two hosts to communicate with each other in same subnet. User in access network 1 should communicate with user in access network 2 (S1-R13) and user in access network 2 should communicate with user in access network 3 (S3-R14). This access should be achieved by doing VXLAN Tunneling. Similarly, other user in access network 1 should communicate with other user in access network 2 and same user in access network 2 is communication with other user in access network 3. For both users in access network 1 to not communicate with each other we did network slicing. With network slicing we are able to make both users in access network 1 apart from each other and thus not both users will not be able to communicate with each other.

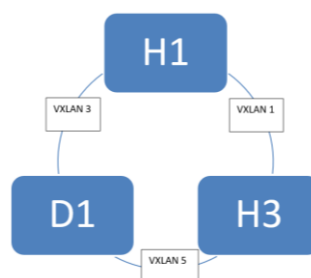


Fig: Logical VXLAN Tunnel between 3 Hosts from 3 access networks

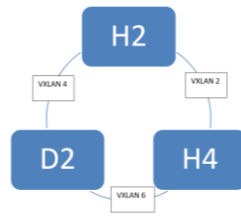


Fig: Logical VXLAN Tunnel between 3 Hosts from 3 access networks

Device	Category	Eth0	Eth1	Eth2	Eth3	Eth4
H1	Host	192.168.1.100/24				
H2	Host	192.168.1.200/24				
H3	Host	192.168.2.100/24				
H4	Host	192.168.2.200/24				
D1	Host	192.168.3.100/24				
D2	Host	192.168.3.200/24				
S1	Switch					
S2	Switch					
R1	Router	10.10.10.1/30	100.10.1.1/30	101.20.1.1/30		
R2	Router	20.20.20.1/30	116.170.1.2/30	117.180.1.2/30		
R3	Router	100.10.1.2/30	102.30.1.1/30	103.40.1.1/30		
R4	Router	101.20.1.2/30	102.30.1.2/30	104.50.1.1/30	105.60.1.1/30	
R5	Router	116.170.1.1/30	115.160.1.1/30	108.90.1.2/30	113.140.1.2/30	
R6	Router	117.180.1.1/30	115.160.1.2/30	114.150.1.2/30		
R7	Router	103.40.1.1/30	106.70.1.1/30	107.80.1.1/30		
R8	Router	104.50.1.2/30	106.70.1.2/30	108.90.1.1/30	109.100.1.1/30	
R9	Router	105.60.1.2/30	113.140.1.1/30	112.130.1.1/30	110.110.1.2/30	
R10	Router	114.150.1.1/30	112.130.1.2/30	111.120.1.2/30		

R11	Router	107.80.1.2/30	109.100.1.2/30	110.100.1.1/30	111.120.1.1/30	150.50.50.1/30
R12	Router	192.168.1.1/24	10.10.10.2/30			
R13	Router	192.168.2.1/24	20.20.20.2/30			
R14	Router	192.168.3.1/24	150.50.50.2/30			

Fig: IPv4 details

3.2 Implementation

Here is the explanation for the python code which used to build and execute the network.

```
#!/usr/bin/python
from mininet.net import Containernet
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import Host
from mininet.node import Node
from mininet.node import OVSKernelSwitch, UserSwitch, OVSSwitch
from mininet.node import IVSSwitch
from mininet.log import setLogLevel, info
from mininet.cli import CLI
from mininet.link import Link, TCLink
import time
import glob, os
```

Fig-1

Fig-1: At first all the required libeies files are imported for different purpose.

```
# Configure the router's IPV4 forwarding
class LinuxRouter( Node ):
    "A Node with IP forwarding enabled."

    def config( self, **params ):
        super( LinuxRouter, self ).config( **params )
        # Enable
        self.cmd( 'sysctl net.ipv4.ip_forward=1' )

    def terminate( self ):
        # Disable
        self.cmd( 'sysctl net.ipv4.ip_forward=0' )
        super( LinuxRouter, self ).terminate()
```

Fig-2

Fig-2: They are two fuction are implemented in this code. The config fuction that will enbale IPv4 forwarding for each router when the main fuction will be exceuted. The Terminate fuction will terminate the IPv4 forwarding

```
class ospfTopo():
    # Add the Container net and define the switch category
    net = Containernet( controller=Controller, link=TCLink, switch=OVSKernelSwitch )
    info('Adding controller\n')
    c0 = net.addController( name='c0', controller=Controller, ip='127.0.0.1', protocol='tcp', port=6633 )
```

Fig-3

Fig-3: Here is intiation main class called “ospfTopo” and adding the Cointainernet and explian the type of switch used in the network, IP addresses, protocol, and port number.

```

#Define Hosts

# Define DHCP Servers
d3 = net.addDocker( name='d3', ip='192.168.1.10/24', defaultRoute='via 192.168.1.1', dimage='dhcp1', mac="04:a3:82:14:30:31")

h1 = net.addDocker( name='h1', ip='192.168.1.100/24', defaultRoute='via 192.168.1.1', dimage='host', mac="24:ec:5d:83:b1:5d")
h2 = net.addHost( name='h2', ip='192.168.1.200', defaultRoute='via 192.168.1.1')

#===== Host 2 IP Renew =====
#h2.cmd("dhclient -r h2-eth0")
#h2.cmd("dhclient h2-eth0")
#===== End =====

h3 = net.addDocker( name='h3', ip='192.168.2.100/24', defaultRoute='via 192.168.2.1', dimage='host1', mac="95:0e:5e:de:70:e9")
h4 = net.addHost( name='h4', ip='192.168.2.200/24', defaultRoute='via 192.168.2.1', mac="5e:dc:11:8f:e0:5d")

# Define Call Servers
d1 = net.addDocker( name='d1', ip='192.168.3.100/24', defaultRoute='via 192.168.3.1', dimage='callserver', mac="04:a3:82:14:20:31")

# Define File Server
d2 = net.addDocker( name='d2', ip='192.168.3.200/24', defaultRoute='via 192.168.3.1', dimage='vsftpl', mac="00:00:00:00:01:01")

```

Fig-4

Fig-4: Defining the type of Hosts, host's names , IP addresses, default routes, and also the mac address.

```

#Define switches

s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
s3 = net.addSwitch('s3', cls=OVSKernelSwitch)

```

Fig-5

Fig-5: Defing the three different switches and the switch types.

```

#Define Routers
r1 = net.addHost( 'r1', cls=LinuxRouter, ip='10.10.10.1/30' )
r2 = net.addHost( 'r2', cls=LinuxRouter, ip='20.20.20.1/30' )
r3 = net.addHost( 'r3', cls=LinuxRouter, ip='100.10.1.2/30' )
r4 = net.addHost( 'r4', cls=LinuxRouter, ip='101.20.1.2/30' )
r5 = net.addHost( 'r5', cls=LinuxRouter, ip='116.170.1.1/30' )
r6 = net.addHost( 'r6', cls=LinuxRouter, ip='117.180.1.1/30' )
r7 = net.addHost( 'r7', cls=LinuxRouter, ip='103.40.1.2/30' )
r8 = net.addHost( 'r8', cls=LinuxRouter, ip='104.50.1.2/30' )
r9 = net.addHost( 'r9', cls=LinuxRouter, ip='105.60.1.2/30' )
r10 = net.addHost( 'r10', cls=LinuxRouter, ip='114.150.1.1/30' )
r11 = net.addHost( 'r11', cls=LinuxRouter, ip='107.80.1.2/30' )
r12 = net.addHost( 'r12', cls=LinuxRouter, ip='192.168.1.1/24' )
r13 = net.addHost( 'r13', cls=LinuxRouter, ip='192.168.2.1/24' )
r14 = net.addHost( 'r14', cls=LinuxRouter, ip='192.168.3.1/24' )

```

Fig-6

Fig-6: Linux Routers are defined with IP addresses.

```

#Define Host to Switch Links
net.addLink(s1, h1)
net.addLink(s1, h2)
net.addLink(s1, d3)
net.addLink(s2, h3)
net.addLink(s2, h4)
net.addLink(s3, d1)
net.addLink(s3, d2)

```

Fig-7

Fig-7: Establishing the links between stwichs and hosts


```
# #Define Switch to Router Links
net.addLink(s1,r12,intfName2='r12-eth0',params2={ 'ip' : '192.168.1.1/24' })
net.addLink(s2,r13,intfName2='r13-eth0',params2={ 'ip' : '192.168.2.1/24' })
net.addLink(s3,r14,intfName2='r14-eth0',params2={ 'ip' : '192.168.3.1/24' })
```

Fig-8

Fig-8: Creating the switch to router links and also setting IP address to the respective ports.

```
#Define Router to Router Links
net.addLink(r1,r12,intfName1='r1-eth0',intfName2='r12-eth1',params1={ 'ip' : '10.10.10.1/30' },params2={ 'ip' : '10.10.10.2/30' })
net.addLink(r1,r3,intfName1='r1-eth1',intfName2='r3-eth0',params1={ 'ip' : '100.10.1.1/30' },params2={ 'ip' : '100.10.1.2/30' })
net.addLink(r1,r4,intfName1='r1-eth2',intfName2='r4-eth0',params1={ 'ip' : '101.20.1.1/30' },params2={ 'ip' : '101.20.1.2/30' })
net.addLink(r3,r4,intfName1='r3-eth1',intfName2='r4-eth1',params1={ 'ip' : '102.30.1.1/30' },params2={ 'ip' : '102.30.1.2/30' })
net.addLink(r3,r7,intfName1='r3-eth2',intfName2='r7-eth0',params1={ 'ip' : '103.40.1.1/30' },params2={ 'ip' : '103.40.1.2/30' })
net.addLink(r4,r8,intfName1='r4-eth2',intfName2='r8-eth0',params1={ 'ip' : '104.50.1.1/30' },params2={ 'ip' : '104.50.1.2/30' })
net.addLink(r4,r9,intfName1='r4-eth3',intfName2='r9-eth0',params1={ 'ip' : '105.60.1.1/30' },params2={ 'ip' : '105.60.1.2/30' })
net.addLink(r7,r8,intfName1='r7-eth1',intfName2='r8-eth1',params1={ 'ip' : '106.70.1.1/30' },params2={ 'ip' : '106.70.1.2/30' })
net.addLink(r7,r11,intfName1='r7-eth2',intfName2='r11-eth0',params1={ 'ip' : '107.80.1.1/30' },params2={ 'ip' : '107.80.1.2/30' })

net.addLink(r8,r11,intfName1='r8-eth3',intfName2='r11-eth1',params1={ 'ip' : '109.100.1.1/30' },params2={ 'ip' : '109.100.1.2/30' })
net.addLink(r11,r9,intfName1='r11-eth2',intfName2='r9-eth3',params1={ 'ip' : '110.100.1.1/30' },params2={ 'ip' : '110.110.1.2/30' })
net.addLink(r9,r10,intfName1='r9-eth2',intfName2='r10-eth1',params1={ 'ip' : '112.130.1.1/30' },params2={ 'ip' : '112.130.1.2/30' })
net.addLink(r11,r10,intfName1='r11-eth3',intfName2='r10-eth2',params1={ 'ip' : '111.120.1.1/30' },params2={ 'ip' : '111.120.1.2/30' })

net.addLink(r2,r13,intfName1='r2-eth0',intfName2='r13-eth1',params1={ 'ip' : '20.20.20.1/30' },params2={ 'ip' : '20.20.20.2/30' })
net.addLink(r5,r2,intfName1='r5-eth0',intfName2='r2-eth1',params1={ 'ip' : '116.170.1.1/30' },params2={ 'ip' : '116.170.1.2/30' })
net.addLink(r6,r2,intfName1='r6-eth0',intfName2='r2-eth2',params1={ 'ip' : '117.180.1.1/30' },params2={ 'ip' : '117.180.1.2/30' })
net.addLink(r5,r6,intfName1='r5-eth1',intfName2='r6-eth1',params1={ 'ip' : '115.160.1.1/30' },params2={ 'ip' : '115.160.1.2/30' })
net.addLink(r8,r5,intfName1='r8-eth2',intfName2='r5-eth2',params1={ 'ip' : '108.90.1.1/30' },params2={ 'ip' : '108.90.1.2/30' })
net.addLink(r9,r5,intfName1='r9-eth1',intfName2='r5-eth3',params1={ 'ip' : '113.140.1.1/30' },params2={ 'ip' : '113.140.1.2/30' })
net.addLink(r10,r6,intfName1='r10-eth0',intfName2='r6-eth2',params1={ 'ip' : '114.150.1.1/30' },params2={ 'ip' : '114.150.1.2/30' })
net.addLink(r11,r14,intfName1='r11-eth4',intfName2='r14-eth1',params1={ 'ip' : '150.50.50.1/30' },params2={ 'ip' : '150.50.50.2/30' })
```

Fig-9

Fig-9: Making the links between all routers by defining port with IP addresses

```
# Define the node name
info( '*** Routing Table on Router:\n' )
s1=net.getNodeByName('s1')
s2=net.getNodeByName('s2')
s3=net.getNodeByName('s3')

r1=net.getNodeByName('r1')
r2=net.getNodeByName('r2')
r3=net.getNodeByName('r3')
r4=net.getNodeByName('r4')
r5=net.getNodeByName('r5')
r6=net.getNodeByName('r6')
r7=net.getNodeByName('r7')
r8=net.getNodeByName('r8')
r9=net.getNodeByName('r9')
r10=net.getNodeByName('r10')
r11=net.getNodeByName('r11')
r12=net.getNodeByName('r12')
r13=net.getNodeByName('r13')
r14=net.getNodeByName('r14')
info('starting zebra and ospfd service:\n')
```

Fig-10

Fig-10: In the our entier network domain these are total number of nodes

```
net.build()
# Start the Controller
c0.start
s1.start( [c0] )
s2.start( [c0] )
s3.start( [c0] )
```

Fig-11

Fig-11: Start the controller and joining with switches.

```
# # Adding MAC on the router interface
info(net['r12']).cmd("ifconfig r12-eth0 hw ether 46:00:43:2c:b0:9e"))
info(net['r13']).cmd("ifconfig r13-eth0 hw ether 00:00:00:00:00:01"))
info(net['r14']).cmd("ifconfig r14-eth0 hw ether 00:00:00:00:00:04"))
```

Fig-12

Fig-12: Configuring the router interfaces with mac addresses.

```
#Create the router APIs
r1.cmd('zebra -f /usr/local/etc/mininet/r1zebra.conf -d -z /usr/local/etc/mininet/r1zebra.api -i /usr/local/etc/mininet/r1zebra.interface')
r2.cmd('zebra -f /usr/local/etc/mininet/r2zebra.conf -d -z /usr/local/etc/mininet/r2zebra.api -i /usr/local/etc/mininet/r2zebra.interface')
r3.cmd('zebra -f /usr/local/etc/mininet/r3zebra.conf -d -z /usr/local/etc/mininet/r3zebra.api -i /usr/local/etc/mininet/r3zebra.interface')
r4.cmd('zebra -f /usr/local/etc/mininet/r4zebra.conf -d -z /usr/local/etc/mininet/r4zebra.api -i /usr/local/etc/mininet/r4zebra.interface')
r5.cmd('zebra -f /usr/local/etc/mininet/r5zebra.conf -d -z /usr/local/etc/mininet/r5zebra.api -i /usr/local/etc/mininet/r5zebra.interface')
r6.cmd('zebra -f /usr/local/etc/mininet/r6zebra.conf -d -z /usr/local/etc/mininet/r6zebra.api -i /usr/local/etc/mininet/r6zebra.interface')
r7.cmd('zebra -f /usr/local/etc/mininet/r7zebra.conf -d -z /usr/local/etc/mininet/r7zebra.api -i /usr/local/etc/mininet/r7zebra.interface')
r8.cmd('zebra -f /usr/local/etc/mininet/r8zebra.conf -d -z /usr/local/etc/mininet/r8zebra.api -i /usr/local/etc/mininet/r8zebra.interface')
r9.cmd('zebra -f /usr/local/etc/mininet/r9zebra.conf -d -z /usr/local/etc/mininet/r9zebra.api -i /usr/local/etc/mininet/r9zebra.interface')
r10.cmd('zebra -f /usr/local/etc/mininet/r10zebra.conf -d -z /usr/local/etc/mininet/r10zebra.api -i /usr/local/etc/mininet/r10zebra.interface')
r11.cmd('zebra -f /usr/local/etc/mininet/r11zebra.conf -d -z /usr/local/etc/mininet/r11zebra.api -i /usr/local/etc/mininet/r11zebra.interface')
r12.cmd('zebra -f /usr/local/etc/mininet/r12zebra.conf -d -z /usr/local/etc/mininet/r12zebra.api -i /usr/local/etc/mininet/r12zebra.interface')
r13.cmd('zebra -f /usr/local/etc/mininet/r13zebra.conf -d -z /usr/local/etc/mininet/r13zebra.api -i /usr/local/etc/mininet/r13zebra.interface')
r14.cmd('zebra -f /usr/local/etc/mininet/r14zebra.conf -d -z /usr/local/etc/mininet/r14zebra.api -i /usr/local/etc/mininet/r14zebra.interface')
```

Fig-13

Fig-13: Creating the router interfaces from “zebra” configuration files.

```
#Create the OSPF interfaces
r1.cmd('ospfd -f /usr/local/etc/mininet/r1ospfd.conf -d -z /usr/local/etc/mininet/r1zebra.api -i /usr/local/etc/mininet/r1ospfd.interface')
r2.cmd('ospfd -f /usr/local/etc/mininet/r2ospfd.conf -d -z /usr/local/etc/mininet/r2zebra.api -i /usr/local/etc/mininet/r2ospfd.interface')
r3.cmd('ospfd -f /usr/local/etc/mininet/r3ospfd.conf -d -z /usr/local/etc/mininet/r3zebra.api -i /usr/local/etc/mininet/r3ospfd.interface')
r4.cmd('ospfd -f /usr/local/etc/mininet/r4ospfd.conf -d -z /usr/local/etc/mininet/r4zebra.api -i /usr/local/etc/mininet/r4ospfd.interface')
r5.cmd('ospfd -f /usr/local/etc/mininet/r5ospfd.conf -d -z /usr/local/etc/mininet/r5zebra.api -i /usr/local/etc/mininet/r5ospfd.interface')
r6.cmd('ospfd -f /usr/local/etc/mininet/r6ospfd.conf -d -z /usr/local/etc/mininet/r6zebra.api -i /usr/local/etc/mininet/r6ospfd.interface')
r7.cmd('ospfd -f /usr/local/etc/mininet/r7ospfd.conf -d -z /usr/local/etc/mininet/r7zebra.api -i /usr/local/etc/mininet/r7ospfd.interface')
r8.cmd('ospfd -f /usr/local/etc/mininet/r8ospfd.conf -d -z /usr/local/etc/mininet/r8zebra.api -i /usr/local/etc/mininet/r8ospfd.interface')
r9.cmd('ospfd -f /usr/local/etc/mininet/r9ospfd.conf -d -z /usr/local/etc/mininet/r9zebra.api -i /usr/local/etc/mininet/r9ospfd.interface')
r10.cmd('ospfd -f /usr/local/etc/mininet/r10ospfd.conf -d -z /usr/local/etc/mininet/r10zebra.api -i /usr/local/etc/mininet/r10ospfd.interface')
r11.cmd('ospfd -f /usr/local/etc/mininet/r11ospfd.conf -d -z /usr/local/etc/mininet/r11zebra.api -i /usr/local/etc/mininet/r11ospfd.interface')
r12.cmd('ospfd -f /usr/local/etc/mininet/r12ospfd.conf -d -z /usr/local/etc/mininet/r12zebra.api -i /usr/local/etc/mininet/r12ospfd.interface')
r13.cmd('ospfd -f /usr/local/etc/mininet/r13ospfd.conf -d -z /usr/local/etc/mininet/r13zebra.api -i /usr/local/etc/mininet/r13ospfd.interface')
r14.cmd('ospfd -f /usr/local/etc/mininet/r14ospfd.conf -d -z /usr/local/etc/mininet/r14zebra.api -i /usr/local/etc/mininet/r14ospfd.interface')
```

Fig-14

Fig-14: Creating the “OSPF” interfaces for routers.

```
#Allow time to create the interfaces
time.sleep(15)
```

Fig-15

Fig-15: Added 15 seconds delay time which help to create interfaces successfully before executing the routing.

```

#===== OVS Forwarding =====

info(net['s1']).cmd("ovs-ofctl add-flow s1 priority=1,arp,actions=flood"))
info(net['s1']).cmd("ovs-ofctl add-flow s1 priority=65535,ip,dst=46:00:43:2c:b0:9e,actions=output:4"))
info(net['s1']).cmd("ovs-ofctl add-flow s1 priority=10,ip,nw_dst=192.168.1.10,actions=output:3"))
info(net['s1']).cmd("ovs-ofctl add-flow s1 priority=10,ip,nw_src=192.168.1.10,nw_dst=192.168.1.200,actions=output:2"))

info(net['s1']).cmd("ovs-ofctl add-flow s1 priority=10,ip,nw_src=192.168.2.100,nw_dst=192.168.1.100,actions=output:1"))
info(net['s1']).cmd("ovs-ofctl add-flow s1 priority=10,ip,nw_src=192.168.3.100,nw_dst=192.168.1.100,actions=output:1"))
info(net['s1']).cmd("ovs-ofctl add-flow s1 priority=10,ip,nw_src=192.168.2.200,nw_dst=192.168.1.200,actions=output:2"))
info(net['s1']).cmd("ovs-ofctl add-flow s1 priority=10,ip,nw_src=192.168.3.200,nw_dst=192.168.1.200,actions=output:2"))

info(net['s2']).cmd("ovs-ofctl add-flow s2 priority=1,arp,actions=flood"))
info(net['s2']).cmd("ovs-ofctl add-flow s2 priority=65535,ip,dst=00:00:00:00:00:01,actions=output:3"))
info(net['s2']).cmd("ovs-ofctl add-flow s2 priority=10,ip,nw_src=192.168.1.100,nw_dst=192.168.2.100,actions=output:1"))
info(net['s2']).cmd("ovs-ofctl add-flow s2 priority=10,ip,nw_src=192.168.3.100,nw_dst=192.168.2.100,actions=output:1"))
info(net['s2']).cmd("ovs-ofctl add-flow s2 priority=10,ip,nw_src=192.168.1.200,nw_dst=192.168.2.200,actions=output:2"))
info(net['s2']).cmd("ovs-ofctl add-flow s2 priority=10,ip,nw_src=192.168.3.200,nw_dst=192.168.2.200,actions=output:2"))

info(net['s3']).cmd("ovs-ofctl add-flow s3 priority=1,arp,actions=flood"))
info(net['s3']).cmd("ovs-ofctl add-flow s3 priority=65535,ip,dst=00:00:00:00:00:04,actions=output:3"))
info(net['s3']).cmd("ovs-ofctl add-flow s3 priority=10,ip,nw_src=192.168.1.100,nw_dst=192.168.3.100,actions=output:1"))
info(net['s3']).cmd("ovs-ofctl add-flow s3 priority=10,ip,nw_src=192.168.2.100,nw_dst=192.168.3.100,actions=output:1"))
info(net['s3']).cmd("ovs-ofctl add-flow s3 priority=10,ip,nw_src=192.168.1.200,nw_dst=192.168.3.200,actions=output:2"))
info(net['s3']).cmd("ovs-ofctl add-flow s3 priority=10,ip,nw_src=192.168.2.200,nw_dst=192.168.3.200,actions=output:2"))

#===== OVS Forwarding END =====

```

Fig-16

Fig-16: Configuring the OVS filtering and network slicing.

```

#===== VXLAN On Linux =====

# For Router 12-13 VXLAN 1-2
r12.cmd ('ip link add vxlan1 type vxlan id 1 remote 20.20.20.2 dstport 4789 dev r12-eth1')
r12.cmd ('ip link add vxlan2 type vxlan id 2 remote 20.20.20.2 dstport 4789 dev r12-eth1')
r13.cmd ('ip link add vxlan1 type vxlan id 1 remote 10.10.10.2 dstport 4789 dev r13-eth1')
r13.cmd ('ip link add vxlan2 type vxlan id 2 remote 10.10.10.2 dstport 4789 dev r13-eth1')

# For Router 12-14 VXLAN 1-2
r12.cmd ('ip link add vxlan3 type vxlan id 3 remote 150.50.50.2 dstport 4789 dev r12-eth1')
r12.cmd ('ip link add vxlan4 type vxlan id 4 remote 150.50.50.2 dstport 4789 dev r12-eth1')
r14.cmd ('ip link add vxlan3 type vxlan id 3 remote 10.10.10.2 dstport 4789 dev r14-eth1')
r14.cmd ('ip link add vxlan4 type vxlan id 4 remote 10.10.10.2 dstport 4789 dev r14-eth1')

# For Router 13-14 VXLAN 1-2
r13.cmd ('ip link add vxlan5 type vxlan id 5 remote 150.50.50.2 dstport 4789 dev r13-eth1')
r13.cmd ('ip link add vxlan6 type vxlan id 6 remote 150.50.50.2 dstport 4789 dev r13-eth1')
r14.cmd ('ip link add vxlan5 type vxlan id 5 remote 20.20.20.2 dstport 4789 dev r14-eth1')
r14.cmd ('ip link add vxlan6 type vxlan id 6 remote 20.20.20.2 dstport 4789 dev r14-eth1')

# For Router 12-13 VXLAN 1-2
r12.cmd ('ip link set vxlan1 up')
r12.cmd ('ip addr add 10.0.0.1/30 dev vxlan1')
r12.cmd ('ip link set vxlan2 up')
r12.cmd ('ip addr add 20.0.0.1/30 dev vxlan2')
r13.cmd ('ip link set vxlan1 up')
r13.cmd ('ip addr add 10.0.0.2/30 dev vxlan1')
r13.cmd ('ip link set vxlan2 up')
r13.cmd ('ip addr add 20.0.0.2/30 dev vxlan2')

# For Router 12-14 VXLAN 3-4
r12.cmd ('ip link set vxlan3 up')
r12.cmd ('ip addr add 30.0.0.1/30 dev vxlan3')
r12.cmd ('ip link set vxlan4 up')
r12.cmd ('ip addr add 40.0.0.1/30 dev vxlan4')
r14.cmd ('ip link set vxlan3 up')
r14.cmd ('ip addr add 30.0.0.2/30 dev vxlan3')
r14.cmd ('ip link set vxlan4 up')
r14.cmd ('ip addr add 40.0.0.2/30 dev vxlan4')

# For Router 13-14 VXLAN 5-6
r13.cmd ('ip link set vxlan5 up')
r13.cmd ('ip addr add 50.0.0.1/30 dev vxlan5')
r13.cmd ('ip link set vxlan6 up')
r13.cmd ('ip addr add 60.0.0.1/30 dev vxlan6')
r14.cmd ('ip link set vxlan5 up')
r14.cmd ('ip addr add 50.0.0.2/30 dev vxlan5')
r14.cmd ('ip link set vxlan6 up')
r14.cmd ('ip addr add 60.0.0.2/30 dev vxlan6')

# VXLAN Static Route
r12.cmd('ip route add 192.168.2.100/32 via 10.0.0.2 encap ip dev vxlan1')
r12.cmd('ip route add 192.168.3.100/32 via 30.0.0.2 encap ip dev vxlan3')
r12.cmd('ip route add 192.168.2.200/32 via 20.0.0.2 encap ip dev vxlan2')
r12.cmd('ip route add 192.168.3.200/32 via 40.0.0.2 encap ip dev vxlan4')

r13.cmd('ip route add 192.168.1.100/32 via 10.0.0.1 encap ip dev vxlan1')
r13.cmd('ip route add 192.168.3.100/32 via 50.0.0.2 encap ip dev vxlan5')
r13.cmd('ip route add 192.168.1.200/32 via 20.0.0.1 encap ip dev vxlan2')
r13.cmd('ip route add 192.168.3.200/32 via 60.0.0.2 encap ip dev vxlan6')

r14.cmd('ip route add 192.168.1.100/32 via 30.0.0.1 encap ip dev vxlan3')
r14.cmd('ip route add 192.168.2.100/32 via 50.0.0.1 encap ip dev vxlan5')
r14.cmd('ip route add 192.168.1.200/32 via 40.0.0.1 encap ip dev vxlan4')
r14.cmd('ip route add 192.168.2.200/32 via 60.0.0.1 encap ip dev vxlan6')

#===== End =====

```

Fig-17

Fig-17: Configuring the VXLAN tunnel between 3 different access networks which need to communicate each other.

```

CLI{ net }
net.stop()
os.system("killall -9 ospfra")
os.system("rm -f /usr/local/etc/mininet/*api*")
os.system("rm -f /usr/local/etc/mininet/*interface*")
os.system("/etc/init.d/quagga restart")
os.system("ifconfig vxlan_sys_4789 down")

# Stop and delete running dockers
os.system("docker stop $(docker ps -a -q)")
os.system("docker rm $(docker ps -a -q)")
#os.system("service networking restart")

# Clean the mininet enviroment
os.system("mn -c")

if __name__ == '__main__':
    setLogLevel( 'info' )
    ospfTopo()

```

Fig-18

Fig-18: All APIs and interfaces deleted that were created; stop and remove backend dockers and restart the Quagga service.

3.3 Demonstration

Here is the providing of the wireshark snapshots for demonstration.

No.	Time	Source	Destination	Protocol	Length	Info
52	139.978767348	150.50.50.1	224.0.0.5	OSPF	134	LS Update
53	140.093855005	150.50.50.1	224.0.0.5	OSPF	82	Hello Packet
54	140.059335276	150.50.50.2	224.0.0.5	OSPF	82	Hello Packet
55	140.835570617	150.50.50.2	224.0.0.5	OSPF	78	LS Acknowledge
56	149.970899466	150.50.50.1	224.0.0.5	OSPF	134	LS Update
57	149.993667847	150.50.50.1	224.0.0.5	OSPF	98	LS Update
58	150.094063182	150.50.50.1	224.0.0.5	OSPF	82	Hello Packet
59	150.056746874	150.50.50.2	224.0.0.5	OSPF	82	Hello Packet
60	150.056910231	150.50.50.2	224.0.0.5	OSPF	98	LS Update
61	150.848186347	150.50.50.1	224.0.0.5	OSPF	78	LS Acknowledge
62	150.848463698	150.50.50.2	224.0.0.5	OSPF	98	LS Acknowledge

Frame 53: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0
 Ethernet II, Src: 96:25:f5:cd:c7:c3 (96:25:f5:cd:c7:c3), Dst: IPv4mcast_05 (01:00:5e:00:00:05)
 Internet Protocol Version 4, Src: 150.50.50.1, Dst: 224.0.0.5
 Open Shortest Path First
 OSPF Header
 Version: 2
 Message Type: Hello Packet (1)
 Packet Length: 48
 Source OSPF Router: 150.50.50.1
 Area ID: 0.0.0.0 (Backbone)
 Checksum: 0xdacd [correct]
 Auth Type: Null (0)
 Auth Data (none): 0000000000000000
 OSPF Hello Packet
 Network Mask: 255.255.255.252
 Hello Interval [sec]: 10
 Options: 0x02, (E) External Routing
 Router Priority: 1
 Router Dead Interval [sec]: 40
 Designated Router: 150.50.50.2
 Backup Designated Router: 150.50.50.1
 Active Neighbor: 150.50.50.2

Fig-19

Fig-18: OSPF network Building between the routers. (Router 14 populating the routing database).

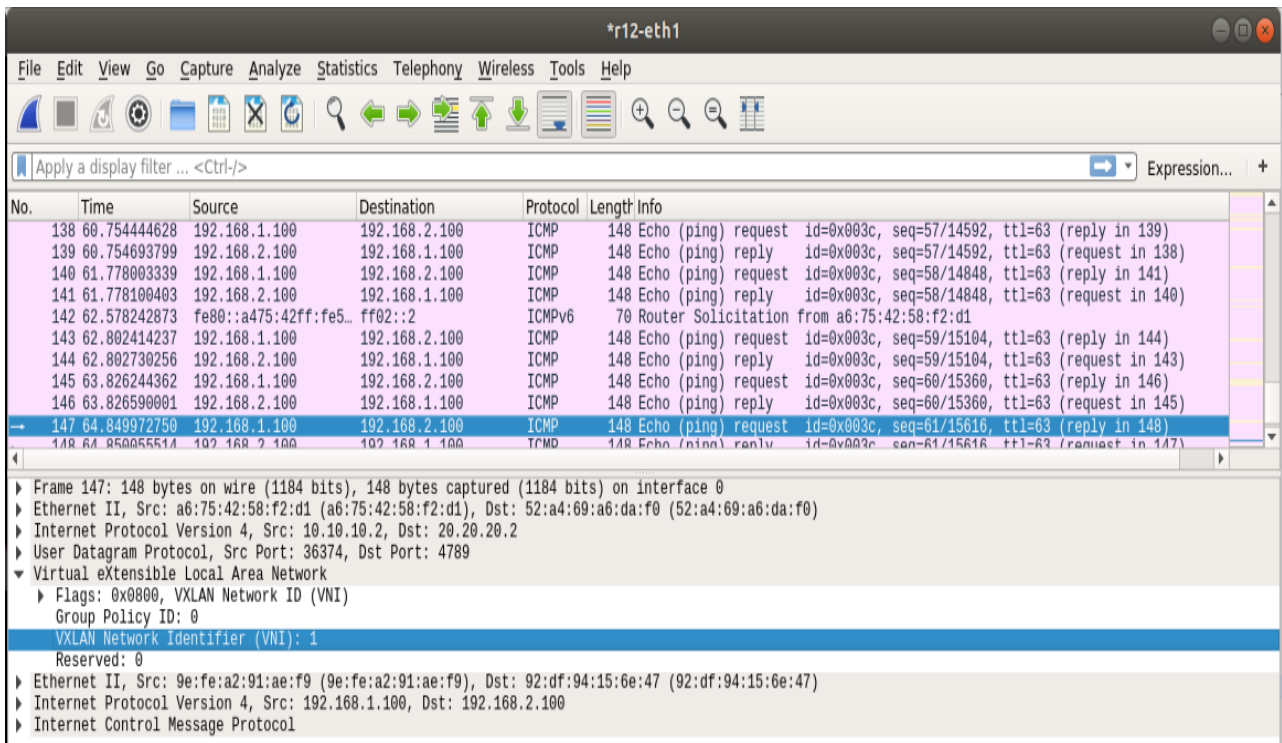


Fig-20

Fig-20: Host 1 and Host 3 successfully communicated with each other through VXLAN using VNI 1.

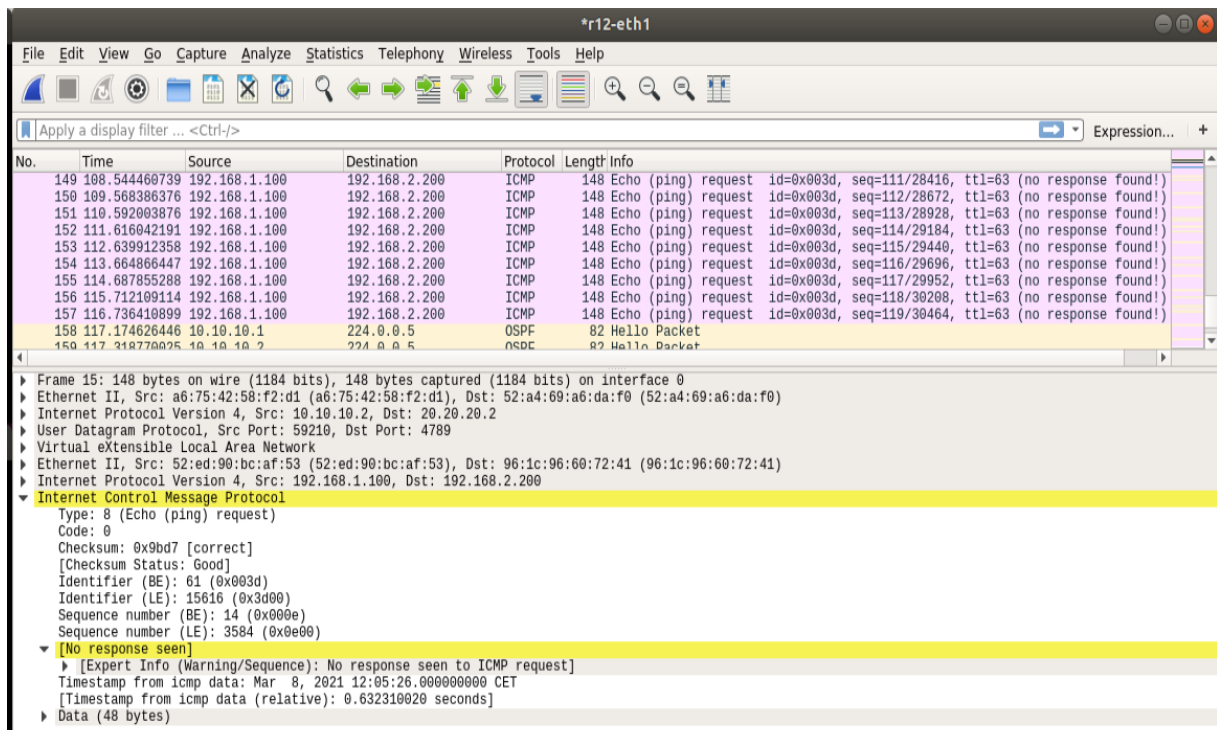


Fig-21

Fig-21: Host 1 is not able to communicate with Host 4 because of the VXLAN

```
root@d2: /
root@d2:/# service dnsmasq start
dnsmasq: unrecognized service
root@d2:/# service vsftpd start
* Starting FTP server vsftpd
/usr/sbin/vsftpd already running.

root@d2:/# [ OK ]
```

Fig-22

Fig-22: The VSFTP server is up and running.

The image shows a Wireshark packet capture of an FTP session. The packet list on the left shows a series of packets, with packet 214 (FTP Response) selected. The packet details pane on the right shows the structure of the selected packet, including the File Transfer Protocol (FTP) section. The packet bytes pane at the bottom shows the raw data of the selected packet.

No.	Time	Source	Destination	Protocol	Length	Info
208	425.304617022	192.168.2.200	192.168.3.200	TCP	116	45238 → 21 [ACK] Seq=33 Ack=115 Win=42496 Len=0 TSval=3139577774 TSecr=773965104
209	430.011978133	150.50.50.1	224.0.0.5	OSPF	82	Hello Packet
210	430.064758269	150.50.50.2	224.0.0.5	OSPF	82	Hello Packet
211	433.837924912	192.168.3.200	192.168.2.200	TCP	124	45348 → 21 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=3139586310 TSecr=773965104
212	433.837967111	192.168.3.200	192.168.2.200	TCP	124	21 → 45348 [SYN, ACK] Seq=0 Ack=1 Win=43440 Len=0 MSS=1460 SACK_PERM=1 TSval=3139586310 TSecr=773965104
213	433.838097951	192.168.2.200	192.168.3.200	TCP	116	45348 → 21 [ACK] Seq=1 Ack=1 Win=42496 Len=0 TSval=3139586307 TSecr=773965104
214	433.840210257	192.168.3.200	192.168.2.200	FTP	136	Response: 220 (vsFTPd 3.0.3)
215	433.840313506	192.168.2.200	192.168.3.200	TCP	116	45348 → 21 [ACK] Seq=1 Ack=21 Win=42496 Len=0 TSval=3139586310 TSecr=773965104
216	437.255064209	192.168.2.200	192.168.3.200	FTP	129	Request: USER mehedi
217	437.255129807	192.168.3.200	192.168.2.200	TCP	116	21 → 45348 [ACK] Seq=21 Ack=14 Win=43520 Len=0 TSval=773968761 TSecr=3139586310
218	437.255261639	192.168.3.200	192.168.2.200	FTP	150	Response: 331 Please specify the password.
219	437.255307083	192.168.2.200	192.168.3.200	TCP	116	45348 → 21 [ACK] Seq=14 Ack=55 Win=42496 Len=0 TSval=3139589725 TSecr=773965104
220	439.536111695	192.168.2.200	192.168.3.200	FTP	130	Request: PASS mininet
221	439.536163358	192.168.3.200	192.168.2.200	TCP	116	21 → 45348 [ACK] Seq=55 Ack=28 Win=43520 Len=0 TSval=773971042 TSecr=3139586310
222	439.567615723	192.168.3.200	192.168.2.200	FTP	130	Response: 230 Login successful.
223	439.567680722	192.168.2.200	192.168.3.200	TCP	116	45348 → 21 [ACK] Seq=28 Ack=78 Win=42496 Len=0 TSval=3139592037 TSecr=773965104
224	439.567754643	192.168.2.200	192.168.3.200	FTP	122	Request: SYST
225	439.567768355	192.168.3.200	192.168.2.200	TCP	116	21 → 45348 [ACK] Seq=78 Ack=34 Win=43520 Len=0 TSval=773971073 TSecr=3139586310
226	439.569003209	192.168.3.200	192.168.2.200	FTP	135	Response: 215 UNIX Type: L8
227	439.569036906	192.168.2.200	192.168.3.200	TCP	116	45348 → 21 [ACK] Seq=34 Ack=97 Win=42496 Len=0 TSval=3139592038 TSecr=773965104
228	440.012779414	150.50.50.1	224.0.0.5	OSPF	82	Hello Packet
229	440.064846199	150.50.50.2	224.0.0.5	OSPF	82	Hello Packet
230	450.013030528	150.50.50.1	224.0.0.5	OSPF	82	Hello Packet
231	450.065075767	150.50.50.2	224.0.0.5	OSPF	82	Hello Packet

Frame 214: 136 bytes on wire (1088 bits), 136 bytes captured (1088 bits) on interface 0
Ethernet II, Src: b6:87:49:77:62:f2 (b6:87:49:77:62:f2), Dst: 96:25:f5:cd:c7:c3 (96:25:f5:cd:c7:c3)
Internet Protocol Version 4, Src: 150.50.50.2, Dst: 20.20.20.2
User Datagram Protocol, Src Port: 55115, Dst Port: 4789
Virtual eXtensible Local Area Network
Ethernet II, Src: ae:b8:c7:2c:79:ff (ae:b8:c7:2c:79:ff), Dst: da:5a:7e:f0:ed:f7 (da:5a:7e:f0:ed:f7)
Internet Protocol Version 4, Src: 192.168.3.200, Dst: 192.168.2.200
Transmission Control Protocol, Src Port: 21, Dst Port: 45348, Seq: 1, Ack: 1, Len: 20
File Transfer Protocol (FTP)
[Current working directory:]

Node: h4
--- 192.168.3.200 ping statistics ---
16 packets transmitted, 16 received, 0% packet loss, time 15113ms
rtt min/avg/max/ndev = 0.130/0.364/0.922/0.202 ms
root@mehedi:/home/mehedi/Desktop/VLAN# ftp -p 192.168.3.200
Connected to 192.168.3.200.
220 (vsFTPd 3.0.3)
Name (192.168.3.200:mehedi): mehedi
331 Please specify the password.
Password:
530 Login incorrect.
Login failed.
ftp> ^Z
[1]+ Stopped ftp -p 192.168.3.200
root@mehedi:/home/mehedi/Desktop/VLAN# ftp -p 192.168.3.200
Connected to 192.168.3.200.
220 (vsFTPd 3.0.3)
Name (192.168.3.200:mehedi): mehedi
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.

Fig-23

Fig-23: This is the Wireshark snapshot of router 14 and it shows that an user has successfully logged in the file server.

```
"Node: h4"

331 Please specify the password.
Password:
530 Login incorrect.
Login failed.
ftp> ^Z
[1]+  Stopped                  ftp -p 192.168.3.200
root@meheidi:/home/meheidi/Desktop/VXLAN# ftp -p 192.168.3.200
Connected to 192.168.3.200.
220 (vsFTPd 3.0.3)
Name (192.168.3.200:meheidi): meheidi
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
227 Entering Passive Mode (192,168,3,200,36,128).

^C
receive aborted
waiting for remote to finish abort
ftp> mkdir test
257 "/home/meheidi/test" created
ftp> █
```

Fig-24

Fig-24: The user named “meheidi” is able to create a directory in ftp server successfully.

The image shows a Wireshark packet capture on the r14-eth1 interface. The packet list on the right shows two distinct FTP sessions. The first session, starting at packet 44, is from 'Node: h2' (192.168.3.200) to the server (192.168.3.200). The second session, starting at packet 54, is from 'Node: h4' (192.168.3.200) to the server (192.168.3.200). Both sessions show successful logins and directory creation. The packet details pane on the right shows the structure of the captured packets, including Ethernet II, Internet Protocol Version 4, and File Transfer Protocol (FTP) fields.

No.	Time	Source	Destination	Protocol	Length	Info
44	0.000000	192.168.3.200	192.168.3.200	FTP	135	Response: 215 UNIX type: L8
54	0.000000	192.168.3.200	192.168.3.200	FTP	135	Response: 215 UNIX type: L8

Fig-25

Fig-25: This snapshot shows that 2 users can login to the FTP server simultaneously and the FTP protocol is initiated.


```
root@d1: /
root@d1: /# /etc/init.d/mysql start
* Starting MySQL database server mysqld [ OK ]
* Checking for tables which need an upgrade, are corrupt or were
not closed cleanly.
root@d1: /# /etc/init.d/kamailio start
* Starting Kamailio SIP server: kamailio
loading modules under /usr/local/lib/kamailio/modules_k:/usr/lib/x86_64-linux-g
nu/kamailio/modules/
Listening on
    udp: 127.0.0.1:5060
    udp: 172.17.0.5:5060
    udp: 192.168.3.100:5060
    tcp: 127.0.0.1:5060
    tcp: 172.17.0.5:5060
    tcp: 192.168.3.100:5060
Aliases:
    tcp: d1:5060
    tcp: localhost:5060
    udp: d1:5060
    udp: localhost:5060
[ OK ]
root@d1: /#
```

Fig-26

Fig-26: The call server is started and ready for initiating SIP session.

```
root@h3: /
root@h3: /# linphonec
2021-03-08 12:57:07:010 ortp-error-Connection to the pulseaudio server failed
Ready
Warning: video is disabled in linphonec, use -V or -C or -D to enable.
linphonec> register sip:ashfaq@192.168.3.100 192.168.3.100
linphonec> Refreshing on sip:ashfaq@192.168.3.100...
linphonec> Registration on sip:192.168.3.100 failed: Unauthorized
linphonec>
Password for ashfaq on 192.168.3.100: mininet
Refreshing on sip:ashfaq@192.168.3.100...
linphonec>
linphonec> Registration on sip:192.168.3.100 successful.
linphonec>
```

Fig-27

Fig-27: This shows that the user “ashfaq” has seccussfully registered on the SIP server.

```

root@h1: /
root@h1:~# linphonec
2021-03-08 16:28:43:506 ortp-error-Connection to the pulseaudio server failed
Ready
Warning: video is disabled in linphonec, use -V or -C or -D to enable.
linphonec> register sip:mehedi@192.168.3.100 192.168.3.100 mehedi
linphonec> Refreshing on sip:mehedi@192.168.3.100...
linphonec> Registration on sip:192.168.3.100 failed: Unauthorized
linphonec>
Password for mehedi on 192.168.3.100: mininet
Refreshing on sip:mehedi@192.168.3.100...
linphonec>
linphonec> Registration on sip:192.168.3.100 successful.
linphonec>

```

Fig-28

Fig-28: This shows that User 2 successfully logged in the SIP server

No.	Time	Source	Destination	Protocol	Length	Info
188	374.870602899	192.168.2.100	192.168.3.100	UDP	96	5060 → 5060 Len=4
189	378.299387573	192.168.1.100	192.168.3.100	UDP	96	5060 → 5060 Len=4
190	380.014384163	150.50.50.1	224.0.0.5	OSPF	82	Hello Packet
191	380.087084207	150.50.50.2	224.0.0.5	OSPF	82	Hello Packet
192	382.182409085	fe80::48e2:d4ff:fea...	ff02::2	ICMPv6	120	Router Solicitation from 4a:e2:d4:a0:bf:c7
193	384.871903486	192.168.2.100	192.168.3.100	UDP	96	5060 → 5060 Len=4
194	388.302097778	192.168.1.100	192.168.3.100	UDP	96	5060 → 5060 Len=4
195	388.323417896	192.168.1.100	192.168.3.100	SIP	494	Request: MESSAGE sip:ashfaq@192.168.3.100 (text/plain)
196	388.324934945	192.168.3.100	192.168.1.100	SIP	521	Status: 407 Proxy Authentication Required
197	388.341637215	192.168.1.100	192.168.3.100	SIP	685	Request: MESSAGE sip:ashfaq@192.168.3.100 (text/plain)
198	388.342121238	192.168.3.100	192.168.2.100	SIP	561	Request: MESSAGE sip:ashfaq@192.168.2.100 (text/plain)
199	388.358643518	192.168.2.100	192.168.3.100	SIP	376	Status: 200 Ok
200	388.358983639	192.168.3.100	192.168.1.100	SIP	314	Status: 200 Ok
201	390.015517127	150.50.50.1	224.0.0.5	OSPF	82	Hello Packet
202	390.087368809	150.50.50.2	224.0.0.5	OSPF	82	Hello Packet
203	393.445920511	4e:49:37:2a:a6:7e	d6:ff:c2:ff:b8:58	ARP	92	Who has 50.0.0.1? Tell 50.0.0.2
204	393.446185749	22:94:2a:32:19:95	96:1c:68:7e:4c:58	ARP	92	Who has 30.0.0.1? Tell 30.0.0.2
205	393.447032129	d6:ff:c2:ff:b8:58	4e:49:37:2a:a6:7e	ARP	92	Who has 50.0.0.2? Tell 50.0.0.1
206	393.447232227	4e:49:37:2a:a6:7e	d6:ff:c2:ff:b8:58	ARP	92	50.0.0.2 is at 4e:49:37:2a:a6:7e
207	393.447329225	d6:ff:c2:ff:b8:58	4e:49:37:2a:a6:7e	ARP	92	50.0.0.1 is at d6:ff:c2:ff:b8:58
208	393.447333366	96:1c:68:7e:4c:58	22:94:2a:32:19:95	ARP	92	30.0.0.1 is at 96:1c:68:7e:4c:58

▶ Ethernet II, Src: 92:8f:9b:bd:ac:94 (92:8f:9b:bd:ac:94), Dst: aa:66:6e:d8:c5:72 (aa:66:6e:d8:c5:72)
 ▶ Internet Protocol Version 4, Src: 10.10.10.2, Dst: 150.50.50.2
 ▶ User Datagram Protocol, Src Port: 36546, Dst Port: 4789
 ▶ Virtual Extensible Local Area Network
 ▶ Ethernet II, Src: 96:1c:68:7e:4c:58 (96:1c:68:7e:4c:58), Dst: 22:94:2a:32:19:95 (22:94:2a:32:19:95)
 ▶ Internet Protocol Version 4, Src: 192.168.1.100, Dst: 192.168.3.100
 ▶ User Datagram Protocol, Src Port: 5060, Dst Port: 5060
 ▶ Session Initiation Protocol (MESSAGE)
 ▶ Request-Line: MESSAGE sip:ashfaq@192.168.3.100 SIP/2.0
 ▶ Message Header
 ▶ Message Body
 ▶ Line-based text data: text/plain (1 lines)
 hello

Fig-29

Fig-29: Now both Users can send message through the SIP server

No.	Time	Source	Destination	Protocol	Length	Info
221	410.085625391	aa:66:6e:d8:c5:72	92:8f:9b:bd:ac:94	ARP	42	150.50.50.2 is at aa:66:6e:d8:c5:72
222	410.087917505	150.50.50.2	224.0.0.5	OSPF	82	Hello Packet
223	414.875093499	192.168.2.100	192.168.3.100	UDP	96	5060 → 5060 Len=4
224	414.948341456	192.168.1.100	192.168.3.100	SIP/SDP	1126	Request: INVITE sip:ashfaq@192.168.3.100
225	414.948588388	192.168.3.100	192.168.1.100	SIP	520	Status: 407 Proxy Authentication Required
226	414.948859985	fe80::a866:6eff:fed::ff02::2	ff02::2	ICMPv6	70	Router Solicitation from aa:66:6e:d8:c5:72
227	414.961387737	192.168.1.100	192.168.3.100	SIP	476	Request: ACK sip:ashfaq@192.168.3.100
228	414.961542508	192.168.1.100	192.168.3.100	SIP/SDP	1317	Request: INVITE sip:ashfaq@192.168.3.100
229	414.961813310	192.168.3.100	192.168.1.100	SIP	395	Status: 100 trying -- your call is important to us
230	414.961919949	192.168.3.100	192.168.2.100	SIP/SDP	1233	Request: INVITE sip:ashfaq@192.168.2.100
231	414.970030551	192.168.2.100	192.168.3.100	SIP	364	Status: 100 Trying
232	414.976357014	192.168.2.100	192.168.3.100	SIP	488	Status: 180 Ringing
233	414.976557761	192.168.3.100	192.168.1.100	SIP	427	Status: 180 Ringing
234	418.326799789	192.168.1.100	192.168.3.100	UDP	96	5060 → 5060 Len=4
235	420.015421129	150.50.50.1	224.0.0.5	OSPF	82	Hello Packet
236	420.088096287	150.50.50.2	224.0.0.5	OSPF	82	Hello Packet
237	424.881334606	192.168.2.100	192.168.3.100	UDP	96	5060 → 5060 Len=4
238	428.329781801	192.168.1.100	192.168.3.100	UDP	96	5060 → 5060 Len=4
239	430.015551560	150.50.50.1	224.0.0.5	OSPF	82	Hello Packet
240	430.080231772	150.50.50.2	224.0.0.5	OSPF	82	Hello Packet
241	434.089728326	192.168.2.100	192.168.3.100	UDP	96	5060 → 5060 Len=4
242	438.332098853	192.168.1.100	192.168.3.100	UDP	96	5060 → 5060 Len=4
243	440.015592671	150.50.50.1	224.0.0.5	OSPF	82	Hello Packet
244	440.037772272	d6:ff:c2:ff:b8:58	4e:49:37:2a:a6:7e	ARP	92	Who has 50.0.0.2? Tell 50.0.0.1
245	440.037800670	4e:49:37:2a:a6:7e	d6:ff:c2:ff:b8:58	ARP	92	50.0.0.2 is at 4e:49:37:2a:a6:7e
246	440.088325840	150.50.50.2	224.0.0.5	OSPF	82	Hello Packet
247	443.365251924	96:1c:68:7e:4c:58	22:94:2a:32:19:95	ARP	92	Who has 30.0.0.2? Tell 30.0.0.1

▶ User Datagram Protocol, Src Port: 36546, Dst Port: 4789
 ▶ Virtual eXtensible Local Area Network
 ▶ Ethernet II, Src: 22:94:2a:32:19:95 (22:94:2a:32:19:95), Dst: 96:1c:68:7e:4c:58 (96:1c:68:7e:4c:58)
 ▶ Internet Protocol Version 4, Src: 192.168.3.100, Dst: 192.168.1.100
 ▶ User Datagram Protocol, Src Port: 5060, Dst Port: 5060
 ▼ Session Initiation Protocol (180)
 ▶ Status-Line: SIP/2.0 180 Ringing
 ▼ Message Header
 ▶ Via: SIP/2.0/UDP 192.168.1.100:5060;branch=29h64bK.jnC49vYW;rport=5060
 ▶ From: <sip:mehedi@192.168.3.100>;tag=HzOmH5onQ
 ▶ To: <sip:ashfaq@192.168.3.100>;tag=fKE5Y2s
 Call-ID: ~bn2R0Dtpo
 ▶ CSeq: 21 INVITE
 User-Agent: Linphonec/3.9.1 (belle-sip/1.4.2)
 Supported: outbound
 ▶ Record-route: <sip:192.168.3.100;lr=on>

Fig-30

```

root@h1: /
root@h1:~# linphonec
2021-03-08 16:28:43:506 ortp-error-Connection to the pulseaudio server failed
Ready
Warning: video is disabled in linphonec, use -V or -C or -D to enable.
linphonec> register sip:mehedi@192.168.3.100 192.168.3.100 mehedi
linphonec> Refreshing on sip:mehedi@192.168.3.100...
linphonec> Registration on sip:192.168.3.100 failed: Unauthorized
linphonec>
Password for mehedi on 192.168.3.100: mininet
Refreshing on sip:mehedi@192.168.3.100...
linphonec>
linphonec> Registration on sip:192.168.3.100 successful.
linphonec> chat ashfaq hello
linphonec> call ashfaq
Establishing call id to sip:ashfaq@192.168.3.100, assigned id 1
Contacting sip:ashfaq@192.168.3.100
linphonec> Call 1 to sip:ashfaq@192.168.3.100 in progress.
linphonec> Remote ringing.
linphonec> Remote ringing...
linphonec> Call 1 to sip:ashfaq@192.168.3.100 ringing.
User is busy.
linphonec> Call 1 with sip:ashfaq@192.168.3.100 error.
[]

root@h3: /
root@h3:~# linphonec
2021-03-08 16:28:50:625 ortp-error-Connection to the pulseaudio server failed
Ready
Warning: video is disabled in linphonec, use -V or -C or -D to enable.
linphonec> register sip:ashfaq@192.168.3.100 192.168.3.100 ashfaq
linphonec> Refreshing on sip:ashfaq@192.168.3.100...
linphonec> Registration on sip:192.168.3.100 failed: Unauthorized
linphonec>
Password for ashfaq on 192.168.3.100: mininet
Refreshing on sip:ashfaq@192.168.3.100...
linphonec>
linphonec> Registration on sip:192.168.3.100 successful.
linphonec> Message received from sip:mehedi@192.168.3.100: hello
sip:mehedi@192.168.3.100 is contacting you.
linphonec> Receiving new incoming call from sip:mehedi@192.168.3.100, assigned id 1
linphonec> Call ended
linphonec> You have missed 1 call.
linphonec> Call 1 with sip:mehedi@192.168.3.100 ended (Not answered).
linphonec>
  
```

Fig-31

Fig-30/31: Now both Users can establish the call through the SIP server

4. Discussion and Problems

While implementing the given task we face following complexities.

- 1) This containernet environment was completely new for us so we were facing issues in configuring our network.
- 2) Facing issues while working on Ubuntu Docker images.
- 4) OSPF needs time to initiate the interface and populate the router tables.
- 5) Containernet does not support any docker from docker hub.
- 6) Facing issue while configuring the VXLAN.
- 7) Having difficulties on the implementation of Containernet.
- 8) Network slicing using Open vSwitch

For solving all of this we go through the internet, reading related blogs. It takes more time of our project to cater these problems. Meanwhile, we are in contact with other groups they also have common issues.

5. Improvemnets and implementaion

Our goal is to implement the DHCP and DNS server in the existing topology. Also, improve the network priority based on the user requirements using network slicing.

6. Conclusion

A host from one access network is successfully communicating with other host in other access network. VXLAN tunneling is successfully performed between all 3 access networks. Multiple independent hosts in access network are sliced with the Open vSwitch technology. We tried to create a complex topology based on the dynamic routing (OSPF). For increasing the functionality to the end users, we implemented two more service, SIP call and Text based chat.

7. References

Containernet: <https://github.com/containernet/containernet>

Wireshark: <https://www.ukessays.com/essays/information-technology/cisco-prime-wireshark-advantages-7229.php>

VSFTP: <https://wiki.ubuntuusers.de/vsftpd/>

Quagga: <https://www.quagga.net/docs/quagga.pdf>

<https://www.kamailio.org/wiki/tutorials/getting-started/main>

VXLAN CISCO overview: <http://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-729383.html>

Arista VXLAN white paper:

https://www.arista.com/assets/data/pdf/Whitepapers/Arista_Networks_VXLAN_White_Paper.pdf

Software Defined Networking: <https://www.sdxcentral.com/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/>