Mobile Computing

Category A - Group 3

File hosting / File sharing - VXLAN

Submitted By:

Name	Matriculation Number
MIR MEHEDI HASAN RAYHAN	1324345

Examiner:

Date of Submission:

Contents

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 Technlogy

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

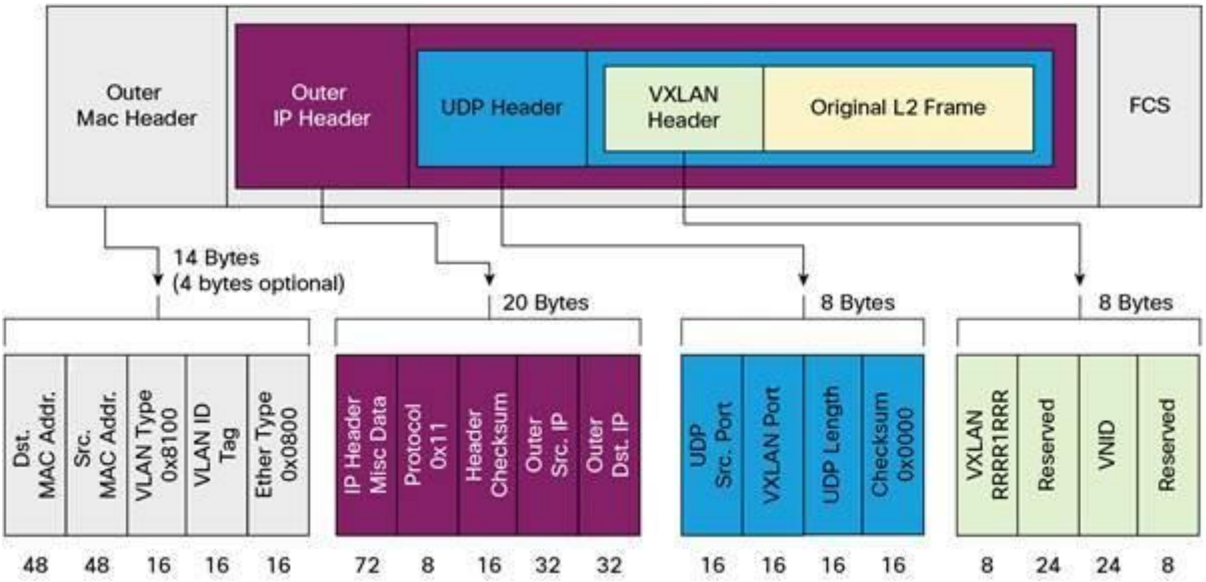
ContainerNet: Containernet is a fork of the famous Mininet network emulator and allows to use Docker containers as hosts in emulated network topologies. This enables interesting functionalities to build networking/cloud emulators and testbeds. Besides this, it is actively used by the research community, 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.

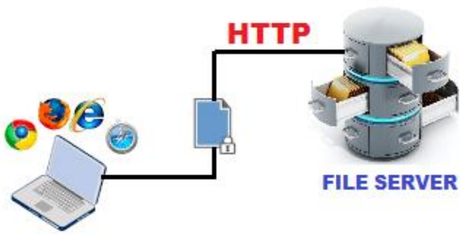
OSPF: OSPF is, mostly, a link-state routing protocol. In contrast to distance-vector protocols, such as RIP or BGP, where routers describe available paths (i.e., routes) to each other, in link-state protocols routers instead describe the state of their links to their immediate neighboring routers. Each 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.

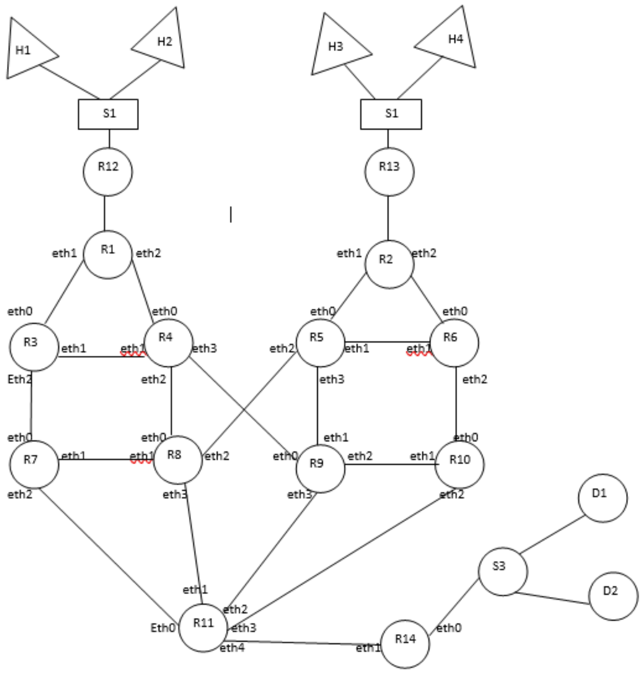


Fig-1: Network Topology

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.

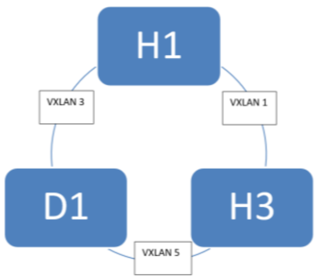


Fig: Logical VXLAN Tunnel between 3 Hosts from 3 access networks

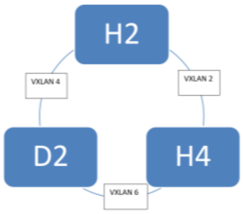


Fig: Logical VXLAN Tunnel between 3 Hosts from 3 access networks

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 libreies 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 fucntion will be exceuted. The Terminate fucntion 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="vsftp1", 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 swtiches 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 swtich to router links and also seting IP address to the recpective 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 swtiches.

```
# # 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 rotuer 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,dl_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,dl_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,dl_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.

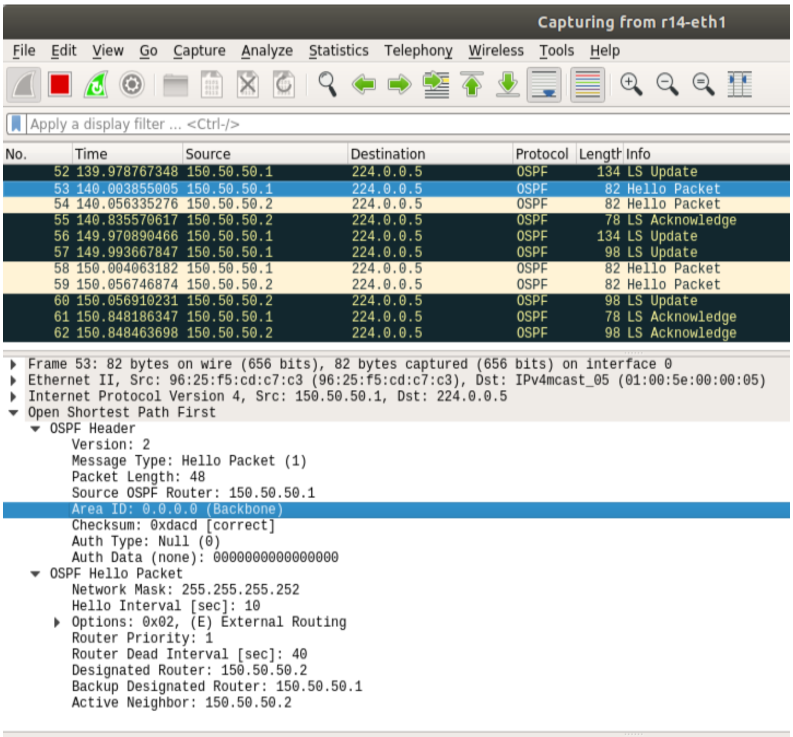


Fig-19

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

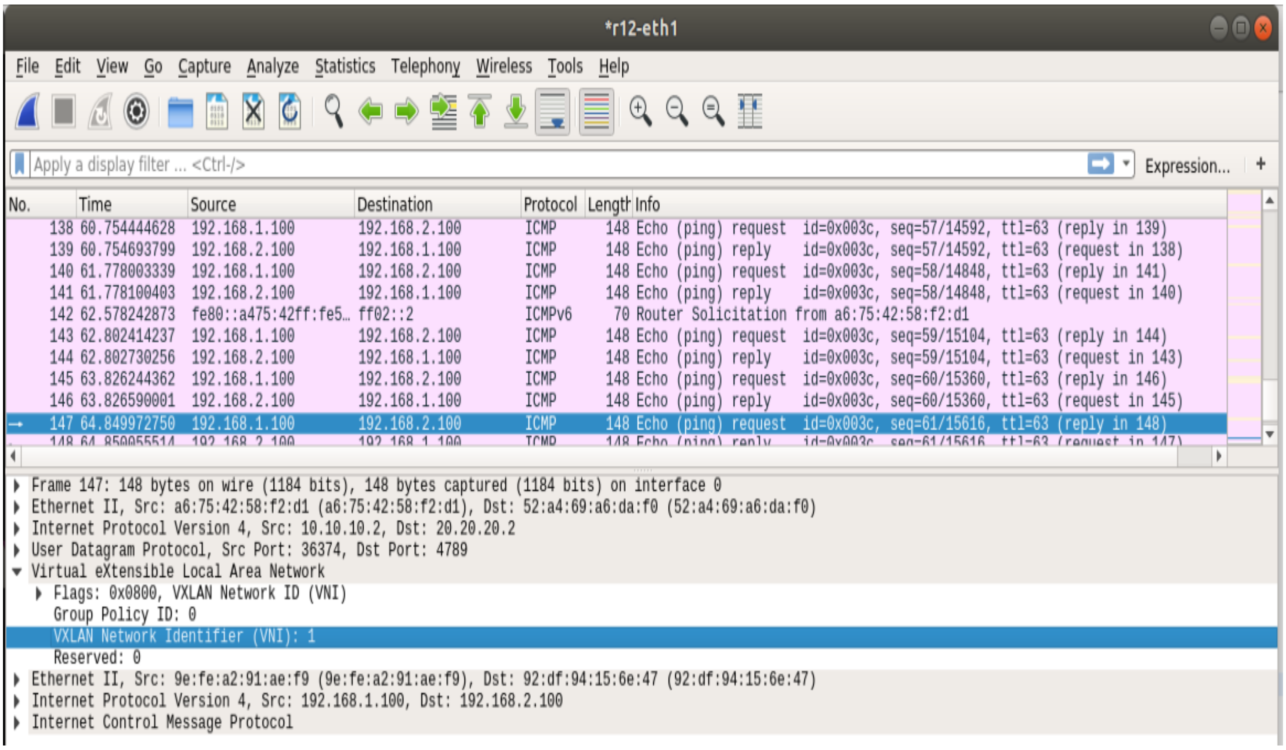


Fig-20

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

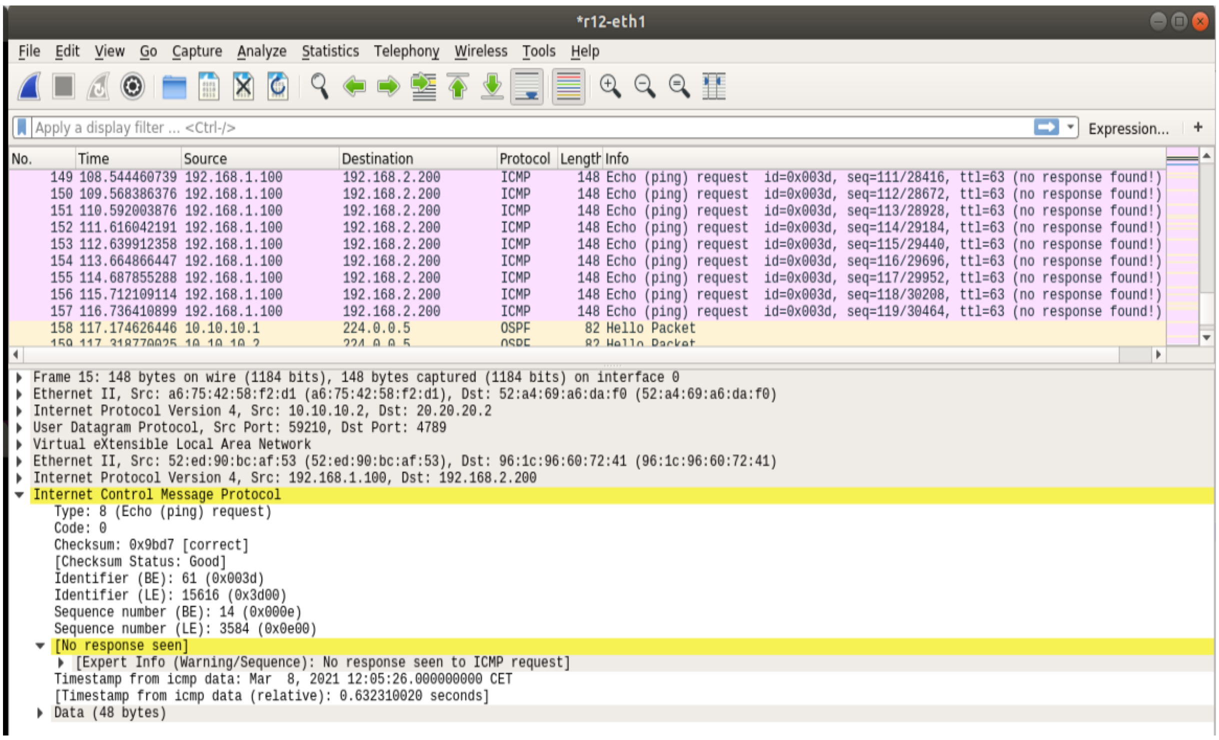


Fig-21

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

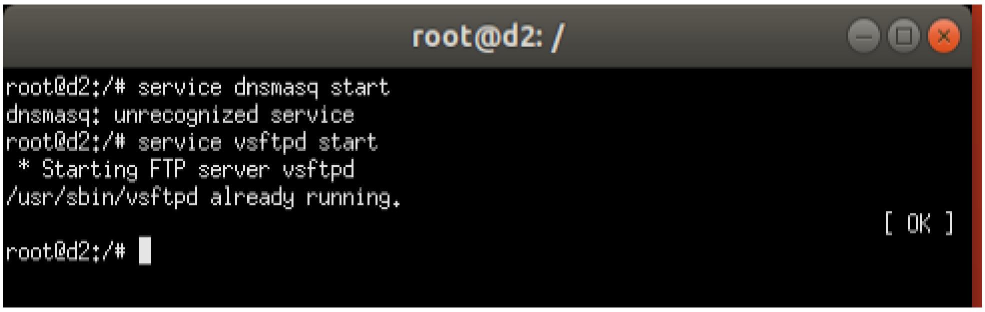


Fig-22

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

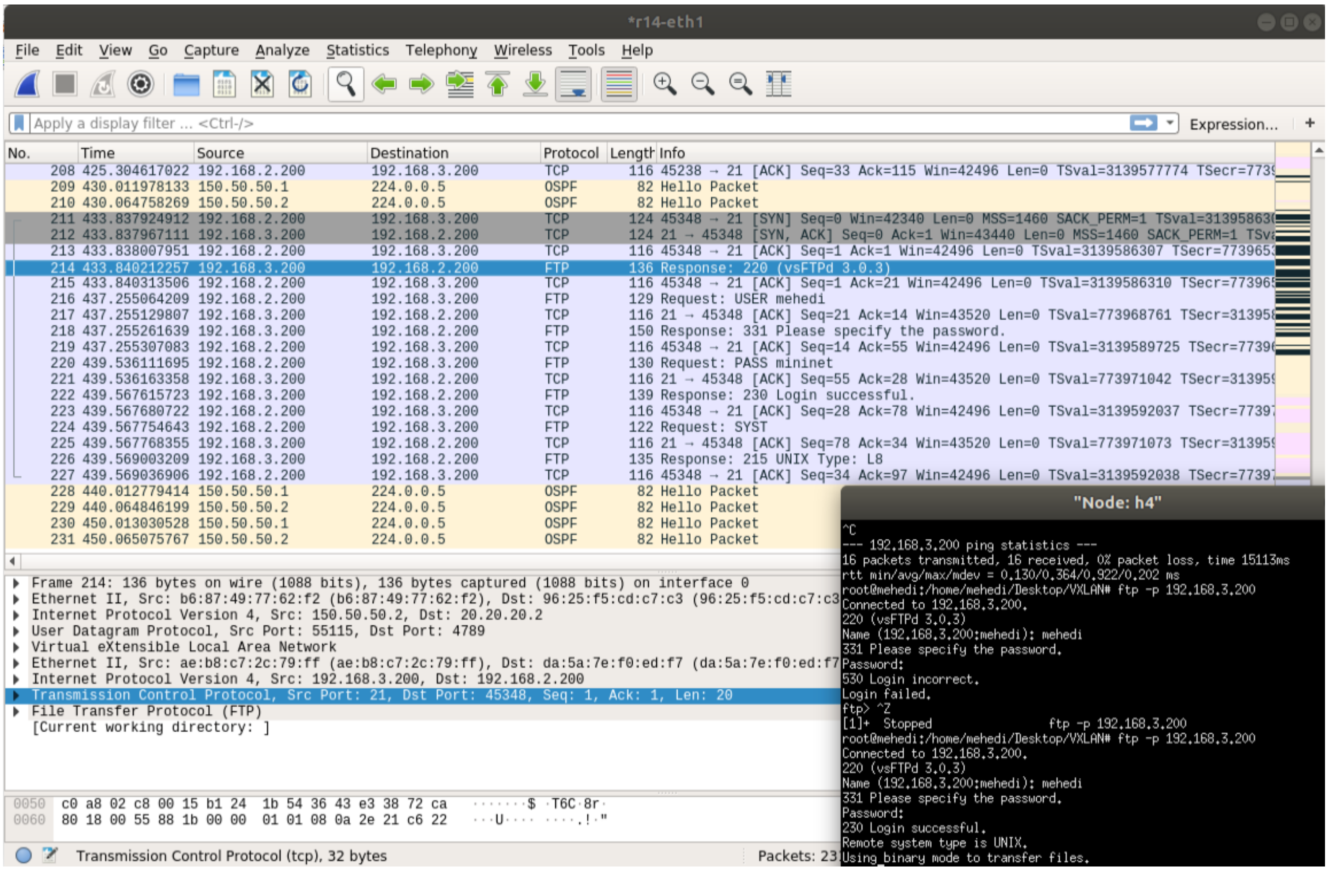


Fig-23

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

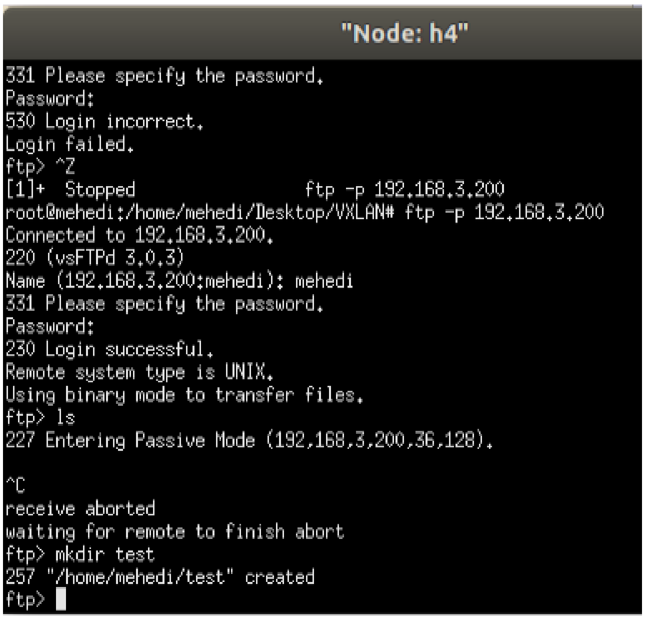


Fig-24

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

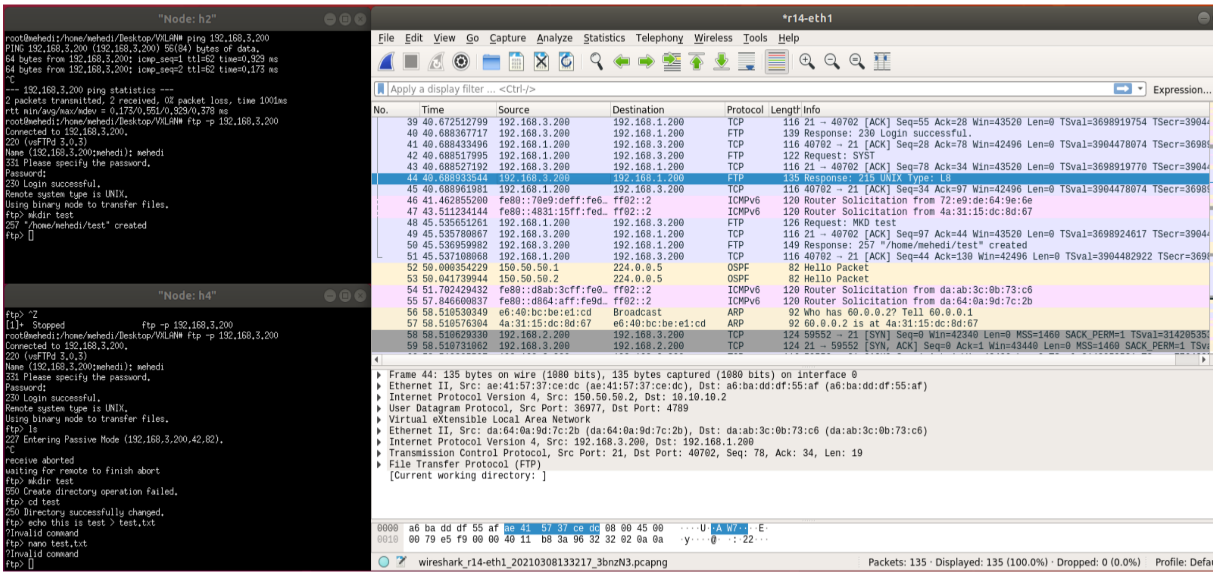


Fig-25

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

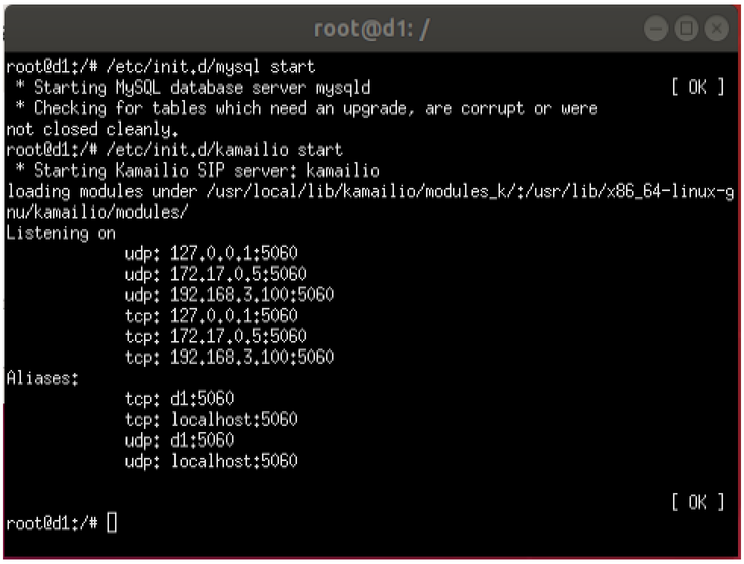


Fig-26

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

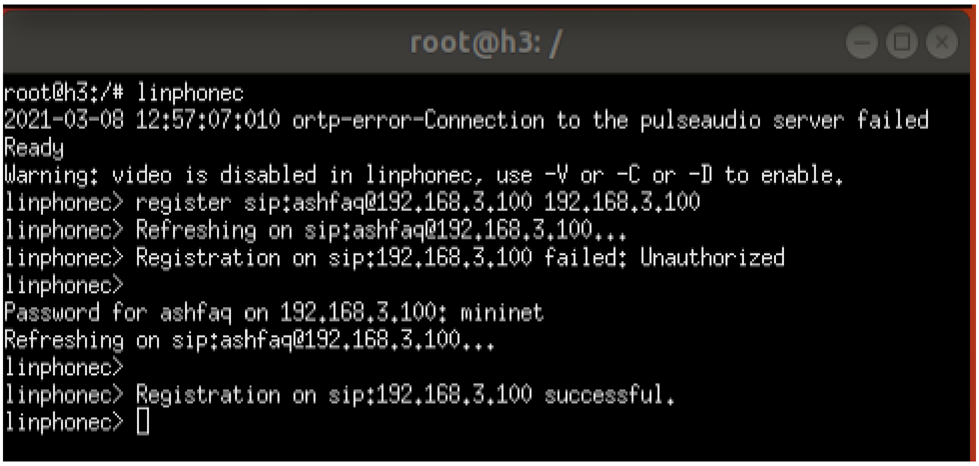


Fig-27

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

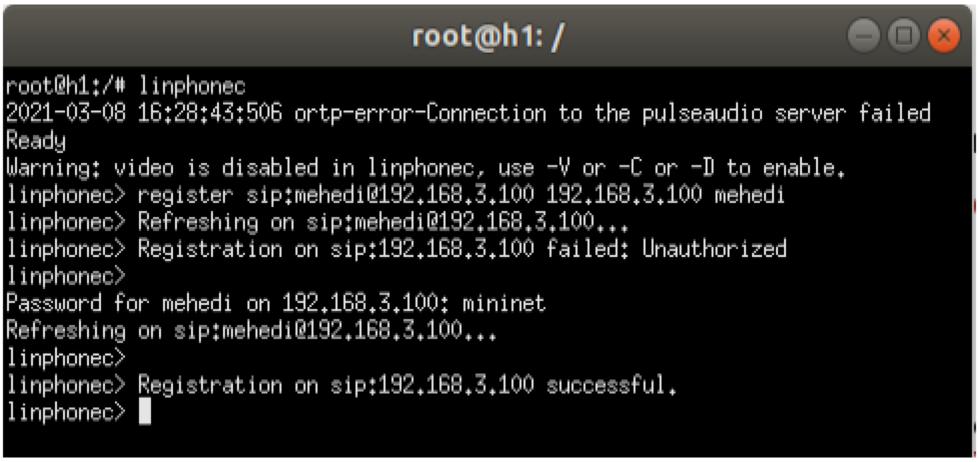


Fig-28

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

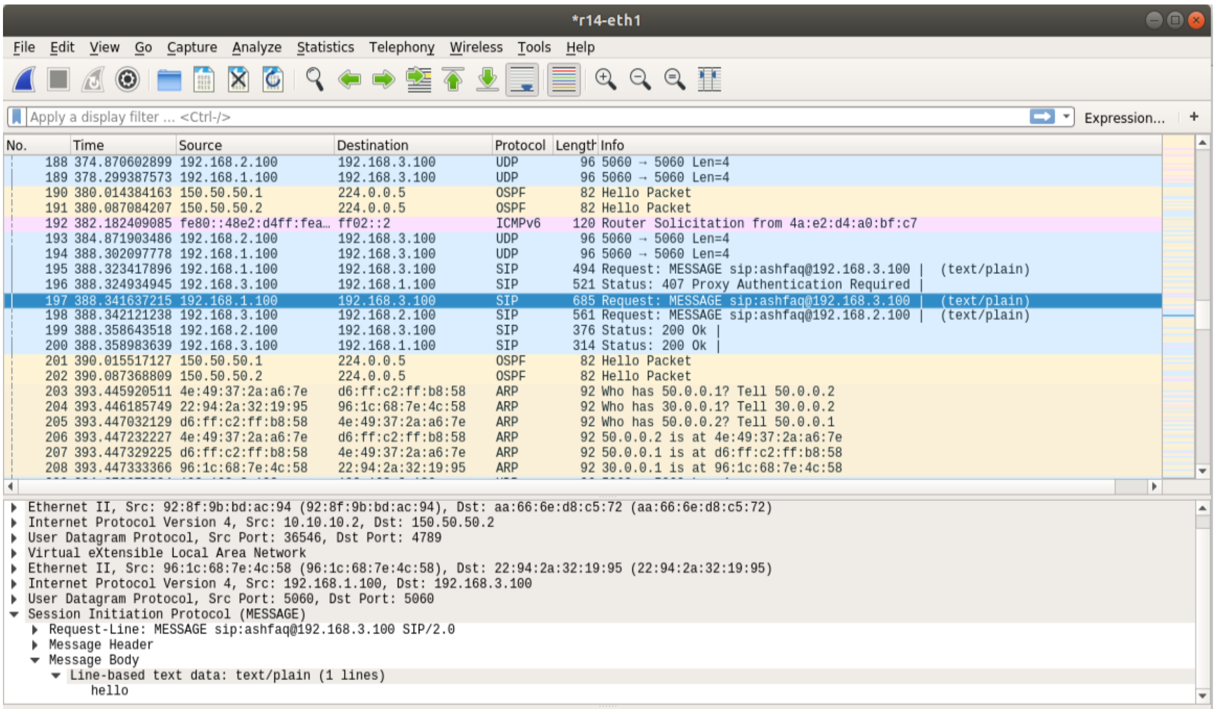


Fig-29

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

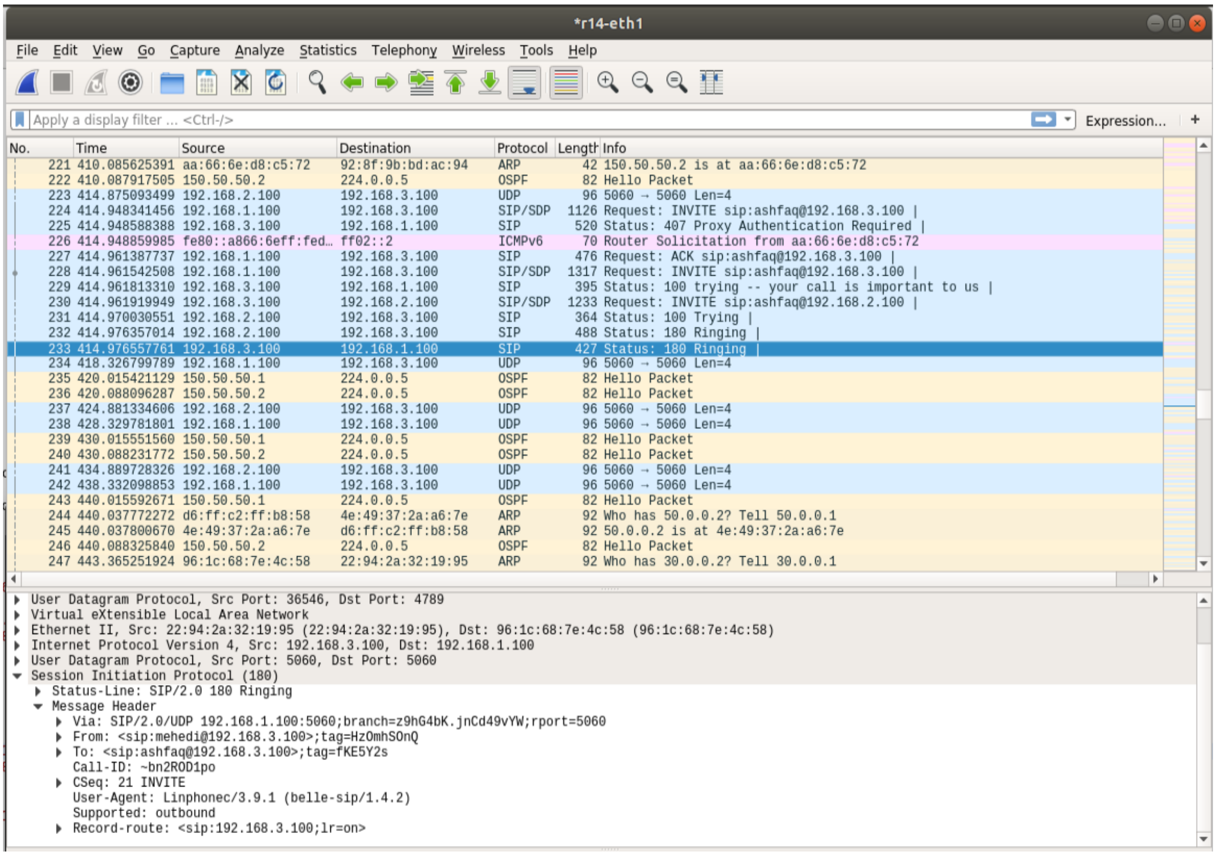


Fig-30

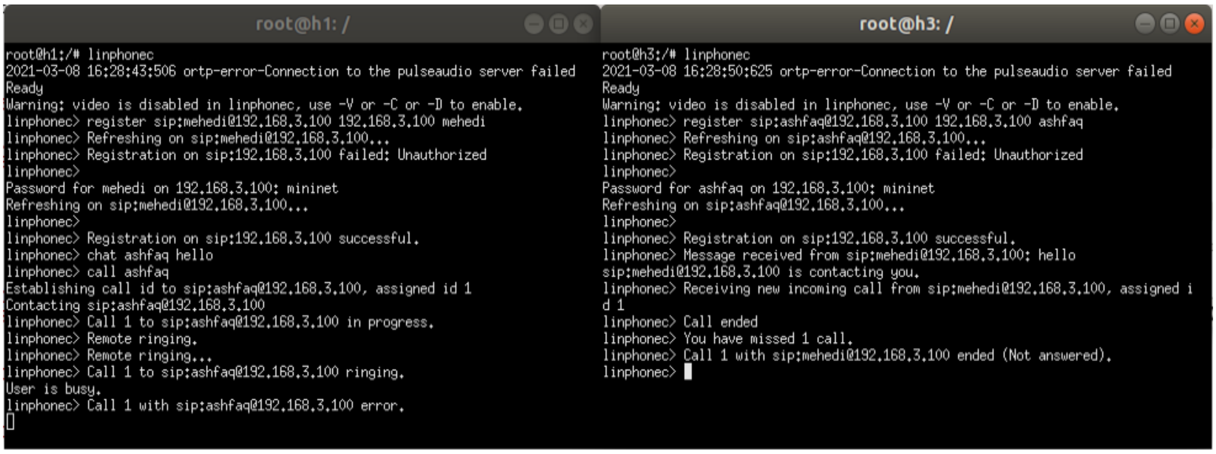


Fig-31

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/