

SPRING 2018 –EE555 PROJECT
Open Flow Protocol using POX and Mininet

Submitted by:

MOKSHITH KUMAR TUMALLAPALLI

SRIRAM UNNIKRISHNAN

ABSTRACT:

In this project, we get a hands-on experience on the OpenFlow protocol which is used to govern the communication between a controller and a switch in a Software Defined Network (SDN) environment. This protocol gives access to the forwarding plane of the switch or router.

SETUP:

To start with the project the following software were installed:

- Oracle VirtualBox
- The tutorial image from Github
- Putty to SSH into the Virtual machine
- Xming to run multiple X servers

Ensure that the Virtual machine has two network interfaces. This can be configured in VirtualBox with one NAT interface and one host-only interface. Start the virtual machine and type "***ifconfig -a***" and observe that there are two interfaces with IP address 10.0.0.x and 192.168.y.z.

From here onwards, you can connect to the Virtual machine by SSH using Putty. Enable X11 forwarding in Putty in order to use X11 applications like Xterm and Wireshark.

Note – the project was implemented in a MAC and Windows system. Neither Putty nor Xming is required to be installed in MAC.

PART 1

Hub Emulation:

The following network is used to emulate the hub behaviour.

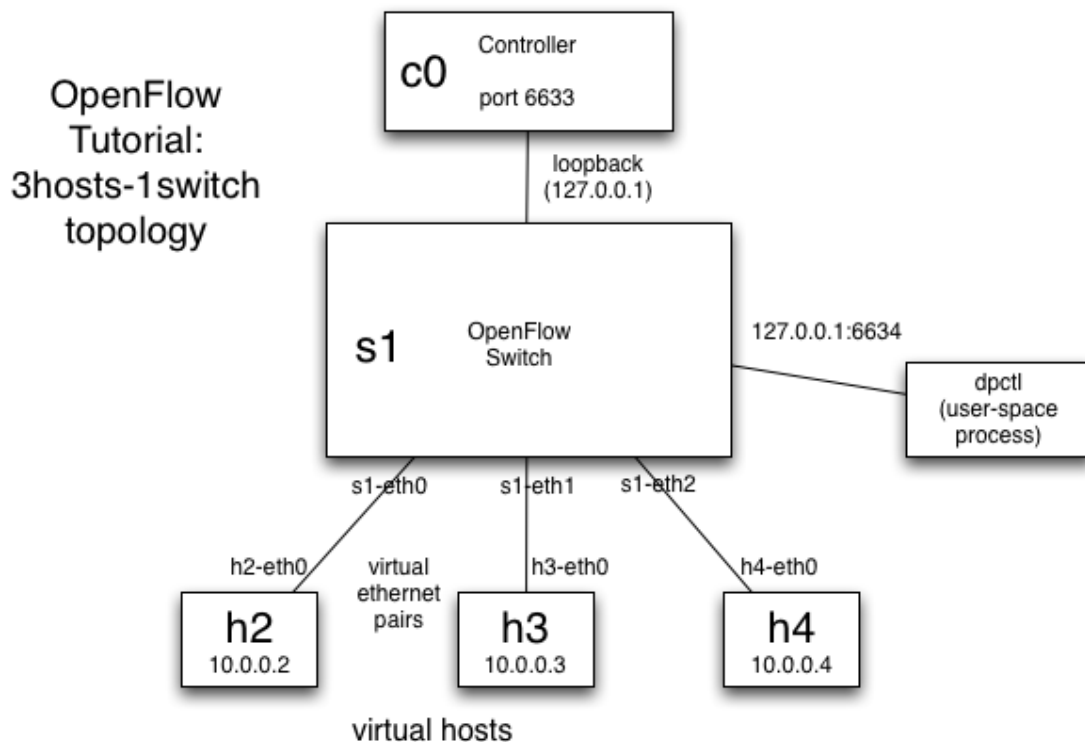


Figure 1: Hub and Self-Learning Switch Topology

The above configuration is created with the following mininet command typed into the terminal.

```
sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

The VM image includes the OpenFlow Wireshark dissector pre-installed. It is best if we open wireshark as a background process. Enter the following command in another terminal.

```
$ sudo wireshark &
```

We can view the openflow messages here by typing “of” in the filter tab and then Apply-ing it. Capture the “lo” interface to observe the openflow messages between the hub and the controller.

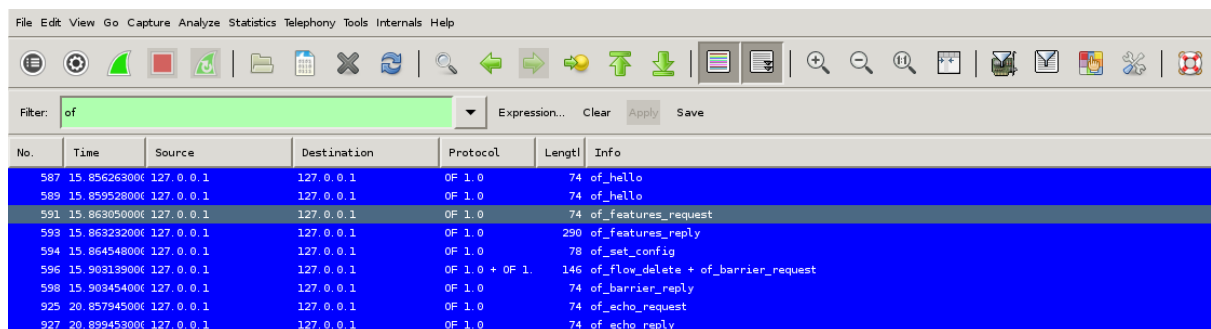
Open another terminal SSHed to the Virtual machine and change the directory to pox. Once inside the pox directory, execute the following command:

```
./pox.py log.level --DEBUG misc.of_tutorial
```

```
mininet@mininet-vm: ~/pox
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG misc.of_tutorial
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.13.0-24-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 1]
```

Figure 2: Starting the Controller

This tells POX to enable verbose logging and to start the of_tutorial component currently acts like a hub. We can also view the openflow messages in wireshark.



No.	Time	Source	Destination	Protocol	Length	Info
587	15.856263000	127.0.0.1	127.0.0.1	OF 1.0	74	of_hello
589	15.859528000	127.0.0.1	127.0.0.1	OF 1.0	74	of_hello
591	15.863050000	127.0.0.1	127.0.0.1	OF 1.0	74	of_features_request
593	15.863232000	127.0.0.1	127.0.0.1	OF 1.0	290	of_features_reply
594	15.864548000	127.0.0.1	127.0.0.1	OF 1.0	78	of_set_config
596	15.903139000	127.0.0.1	127.0.0.1	OF 1.0 + OF 1.	146	of_flow_delete + of_barrier_request
598	15.903454000	127.0.0.1	127.0.0.1	OF 1.0	74	of_barrier_reply
925	20.857945000	127.0.0.1	127.0.0.1	OF 1.0	74	of_echo_request
927	20.899453000	127.0.0.1	127.0.0.1	OF 1.0	74	of_echo_reply

Figure 3: OpenFlow Messages in Wireshark

Now we test the hub behaviour with the following commands in the mininet:

```
xterm h1 h2 h3
```

This opens up three xterms for each of the hosts.

```
mininet@mininet-vm: ~/pox
mininet@mininet-vm:~/pox$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet> xterm h1 h2 h3
mininet> 
```

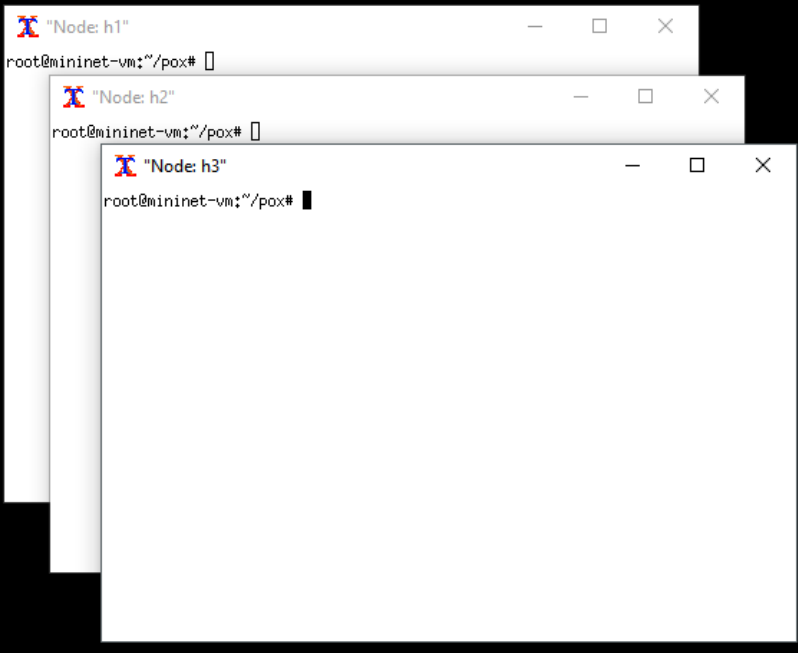
The image shows three overlapping terminal windows. The top window is titled "Node: h1" and shows the mininet CLI prompt. The middle window is titled "Node: h2" and shows the root@mininet-vm prompt. The bottom window is titled "Node: h3" and shows the root@mininet-vm prompt.

Figure 4: Spawning the Configuration

Type the below commands in h2 and h3 xterms to observe the packets seen by that host:

```
tcpdump -XX -n -i h2-eth0
```

```
tcpdump -XX -n -i h3-eth0
```

Now, ping host 2 from host 1, type the command in host 1 xterm to do this:

```
ping -c1 10.0.0.2
```

We observe that the hosts 2 and 3 see identical ARP and ICMP packets.

This illustrates the hub behaviour.

Figure 5: Hub Behaviour

Finally check the performance of a hub by running “*iperf*” in the mininet terminal. We see that hub has a bandwidth performance of 20.5Mbps uplink and 21.6Mbps downlink.

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['10.5 Mbits/sec', '11.9 Mbits/sec']
mininet>
```

Figure 6: Hub Performance

Self-Learning Switch Emulation:

After modifying the hub to a self-learning switch, we run the exact same ping test as in hub.

Note that the network configuration remains the same.

It can be observed that the switch floods the port only if it does not know the destination MAC address. Therefore, host h3 sees only the initial ARP request when h1 queries for MAC address of h2. When h2 sees this request, it sends the ARP response to h1 and not flood it.

The switch learns about the MAC address and port of h1 in its initial ping message. The MAC address and port number of h2 is understood by the switch in its ARP response message.

```

"Node: h3"
root@mininet-vm:~/pox# tcpdump -XX -n -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
18:28:42.280965 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
0x0000: ffff ffff ffff 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0000 0a00 0002 .....

"Node: h2"
root@mininet-vm:~/pox# tcpdump -XX -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
18:28:42.280969 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
0x0000: ffff ffff ffff 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0000 0a00 0002 .....
18:28:42.281072 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
0x0000: 0000 0000 0001 0000 0000 0002 0806 0001 .....
0x0010: 0800 0604 0002 0000 0000 0002 0a00 0002 .....
0x0020: 0000 0000 0001 0a00 0001 .....
18:28:42.286918 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 16302, seq 1, length 64
0x0000: 0000 0000 0002 0000 0000 0001 0800 4500 .....E.
0x0010: 0054 4971 4000 4001 dd35 0a00 0001 0a00 .Tlq@.5.....
0x0020: 0002 0800 8d38 3fae 0001 4a59 d15a 0000 .....8?...JY.Z..
0x0030: 0000 4d91 0300 0000 0000 1011 1213 1415 .....N.....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 .....!*"#$%&'()*+,-./012345
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 .....&'()*+,-./012345
0x0060: 3637 .....67
18:28:42.286951 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 16302, seq 1, length 64
0x0000: 0000 0000 0001 0000 0000 0002 0800 4500 .....E.
0x0010: 0054 0ca7 0000 4001 5a00 0a00 0002 0a00 .T....@.Z.....
0x0020: 0001 0000 9538 3fae 0001 4a59 d15a 0000 .....8?...JY.Z..
0x0030: 0000 4d91 0300 0000 0000 1011 1213 1415 .....N.....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 .....!*"#$%&'()*+,-./012345
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 .....&'()*+,-./012345
0x0060: 3637 .....67
18:28:47.293412 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
0x0000: 0000 0000 0001 0000 0000 0002 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0002 0a00 0002 .....
0x0020: 0000 0000 0000 0a00 0001 .....
18:28:47.299639 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
0x0000: 0000 0000 0002 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0002 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0002 0a00 0002 .....

"Node: h1"
root@mininet-vm:~/pox# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=54.5 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 54.598/54.598/54.598/0.000 ms
root@mininet-vm:~/pox#

```

Figure 7: Switch Behaviour

The performance of the switch is observed to be 18Mbps uplink and 19.7Mbps downlink. This performance increase can be attributed to the fact that not all the ports are flooded with the packet destined for a specific port.

```

mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['18.0 Mbits/sec', '19.7 Mbits/sec']
mininet>

```

Figure 8: Switch Performance without Flow Tables

Now we add the flow tables into the switch. We can see that the switch behaves the same as far as the forwarding of the packets are concerned (same as in Figure 6). However, the performance is drastically improved. This is due to the reason that the switch maintains a flow table and does not send its packets to the controller to decide on which port it should forward.

```

mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['20.9 Gbits/sec', '20.9 Gbits/sec']
mininet>

```

Figure 9: Switch Performance with Flow Enabled

Router Exercise:

In this section, we create a different topology. The custom topology is configured in *mytopo_part1.py* which is a copy of “*~/mininet/custom/topo-2sw-2host.py*”. The following commands are edited to generate the required topology.

```
host1 = self.addHost( 'h1', ip="10.0.1.100/24", defaultRoute = "via 10.0.1.1" )
```

This is repeated for hosts 2 and 3.

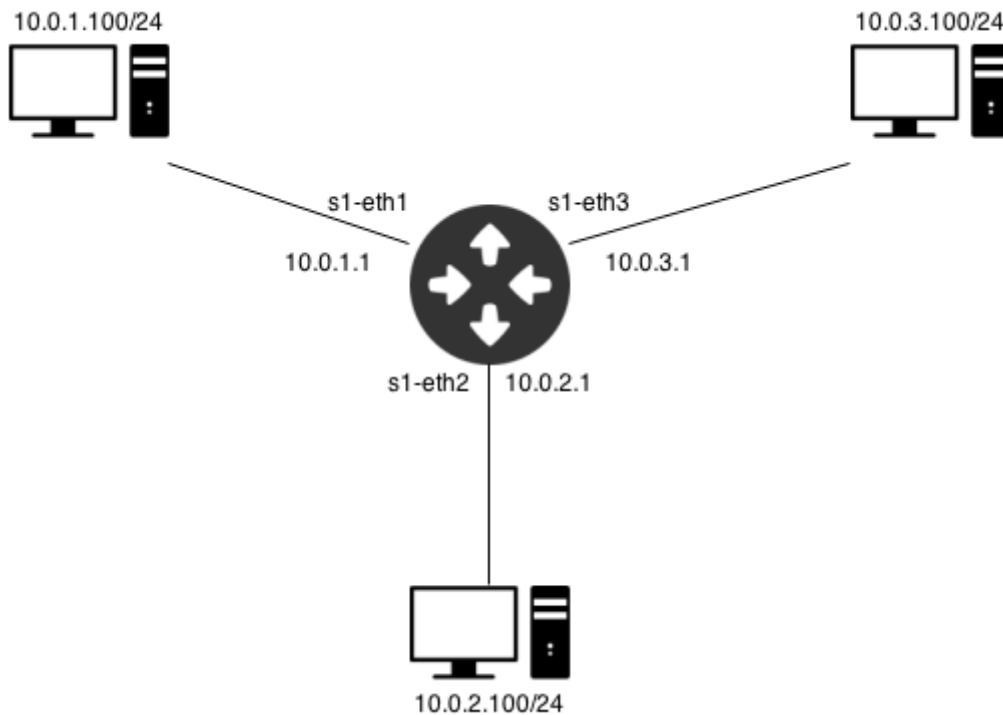


Figure 10: Custom Topology for Router Exercise

The code for the router is written in *router_part1.py* referring the code in the “*~/pox/pox/forwarding/l3_learning.py*”

We start up the custom topology:

```
$ sudo mn --custom mytopo_part1.py --topo mytopo --mac
```

Before starting the controller, we make sure that no other controller is running by executing:

```
$ sudo killall controller
```

Finally, we start the controller from “*~/pox*” directory by:

```
$ ./pox.py log.level --DEBUG misc.router_part1 misc.full_payload
```

Now we are ready to test the router.

First, we try to ping and unknown address range from any of the host. We observe that when we try to ping an address which does not belong to any of the router's subnet, it sends out a **Destination Net Unreachable** message.

Next, we ping an address which is in the same subnet as the source but a non-existing host. In this case too, the destination is unreachable, but we get a **Destination Host Unreachable** as the destination belongs to the same subnet. Note that the source IP address is 10.0.1.100.

Finally, we try to ping a genuine host which is connected to the router. This results in a successful ping.

```
mininet> xterm h1 h2 s1
mininet> h1 ping -c 1 10.99.0.1
PING 10.99.0.1 (10.99.0.1) 56(84) bytes of data.
From 10.99.0.1 icmp_seq=1 Destination Net Unreachable

--- 10.99.0.1 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet> h1 ping -c 1 10.0.1.10
PING 10.0.1.10 (10.0.1.10) 56(84) bytes of data.
From 10.0.1.100 icmp_seq=1 Destination Host Unreachable

--- 10.0.1.10 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet> h1 ping -c 1 10.0.2.100
PING 10.0.2.100 (10.0.2.100) 56(84) bytes of data.
64 bytes from 10.0.2.100: icmp_seq=1 ttl=64 time=111 ms

--- 10.0.2.100 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 111.085/111.085/111.085/0.000 ms
mininet>
```

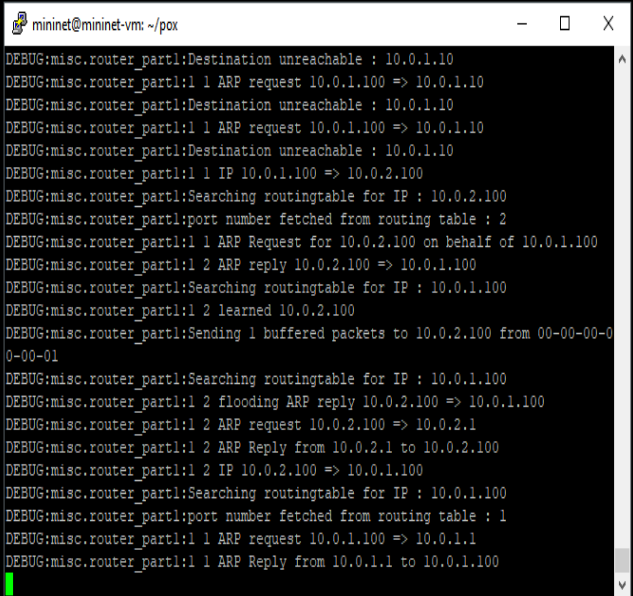


Figure 11: Ping Tests

A tcpdump on the h1-eth0, s1-eth1 and h2-eth0 shows validates the above behaviour.

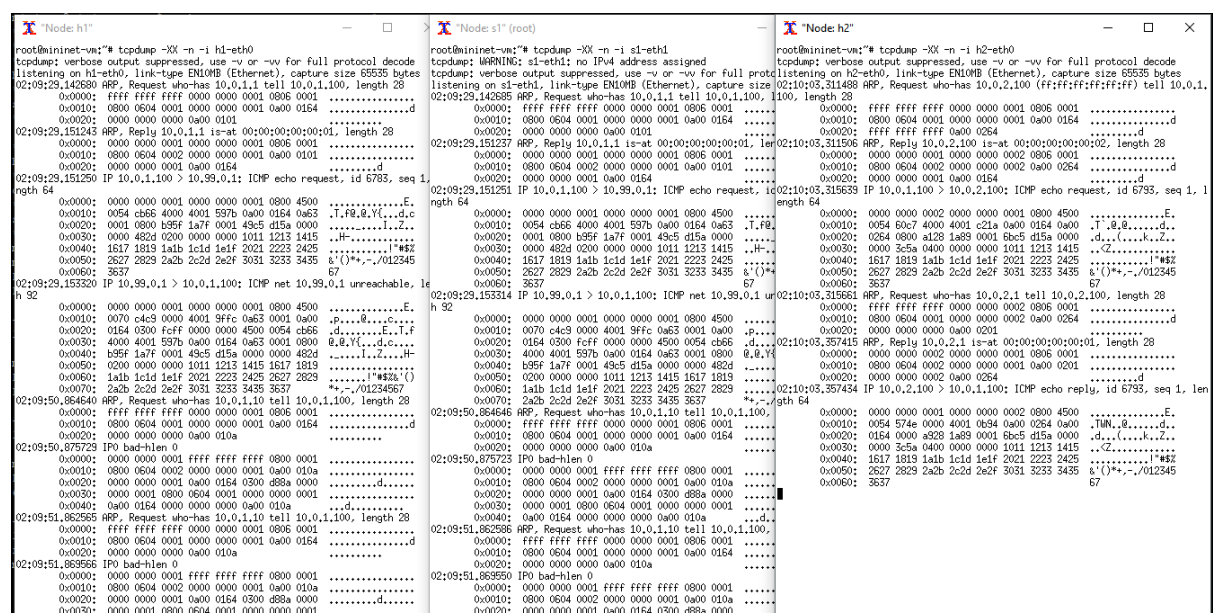


Figure 12: tcpdump for Ping Tests

Now, we test if the router is ping-able.

As expected, we can ping all the interfaces of the routers.

```
mininet> h1 ping -c1 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
64 bytes from 10.0.2.1: icmp_seq=1 ttl=64 time=58.7 ms

--- 10.0.2.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 58.757/58.757/58.757/0.000 ms
mininet> h2 ping -c1 10.0.3.1
PING 10.0.3.1 (10.0.3.1) 56(84) bytes of data.
64 bytes from 10.0.3.1: icmp_seq=1 ttl=64 time=57.2 ms

--- 10.0.3.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 57.235/57.235/57.235/0.000 ms
mininet> h2 ping -c1 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=22.4 ms

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 22.435/22.435/22.435/0.000 ms
mininet>
```

Figure 13: Test for Pinging the Router

```
Node: h1
root@mininet-vm:~# tcpdump -XX -n -i h1-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h1-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
02:24:52.762687 ARP, Request who-has 10.0.1.1 tell 10.0.1.100, length 28
0x0000: ffff ffff ffff 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0001 0a00 0164 .....d
0x0020: 0000 0000 0000 0a00 0101 .....
02:24:52.783435 ARP, Reply 10.0.1.1 is-at 00:00:00:00:00:01, length 28
0x0000: 0000 0000 0001 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0002 0000 0000 0001 0a00 0101 .....
0x0020: 0000 0000 0001 0a00 0164 .....d
02:24:52.783442 IP 10.0.1.100 > 10.0.2.1: ICMP echo request, id 7980, seq 1, len
gth 64
0x0000: 0000 0000 0001 0000 0000 0001 0800 4500 .....E.
0x0010: 0054 6b19 4000 4001 b82b 0a00 0164 0a00 .Tk..@...+...d..
0x0020: 0201 0800 2939 1f2c 0001 e4c8 d15a 0000 ...9.....Z..
0x0030: 0000 2fa3 0b00 0000 0000 1011 1213 1415 ..../.....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 .....!""#%&
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*+,-./012345
0x0060: 3637 67 .....
02:24:52.821428 IP 10.0.2.1 > 10.0.1.100: ICMP echo reply, id 7980, seq 1, lengt
h 64
0x0000: 0000 0000 0001 0000 0000 0001 0800 4500 .....E.
0x0010: 0054 c81d 0000 4001 9b27 0a00 0201 0a00 .T...@.....d..
0x0020: 0164 0000 3139 1f2c 0001 e4c8 d15a 0000 .d..19.....Z..
0x0030: 0000 2fa3 0b00 0000 0000 1011 1213 1415 ..../.....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 .....!""#%&
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*+,-./012345
0x0060: 3637 67 .....
02:26:57.933267 IP 10.0.2.100 > 10.0.1.1: ICMP echo request, id 8017, seq 1, len
gth 64
0x0000: 0000 0000 0001 0000 0000 0001 0800 4500 .....E.
0x0010: 0054 1a8b 4000 4001 08ba 0a00 0264 0a00 .T...@.....d..
0x0020: 0101 0800 e9ce 1f51 0001 61c9 d15a 0000 .....Q...a..Z..
0x0030: 0000 efe7 0d00 0000 0000 1011 1213 1415 .....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 .....!""#%&
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*+,-./012345
0x0060: 3637 67 .....

Node: s1 (root)
root@mininet-vm:~# tcpdump -XX -n -i s1-eth1
tcpdump: WARNING: s1-eth1: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s1-eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
02:24:52.762632 ARP, Request who-has 10.0.1.1 tell 10.0.1.100, length 28
0x0000: ffff ffff ffff 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0001 0a00 0164 .....d
0x0020: 0000 0000 0000 0a00 0101 .....
02:24:52.783429 ARP, Reply 10.0.1.1 is-at 00:00:00:00:00:01, length 28
0x0000: 0000 0000 0001 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0002 0000 0000 0001 0a00 0101 .....
0x0020: 0000 0000 0001 0a00 0164 .....d
02:24:52.783444 IP 10.0.1.100 > 10.0.2.1: ICMP echo request, id 7980, seq 1, len
gth 64
0x0000: 0000 0000 0001 0000 0000 0001 0800 4500 .....E.
0x0010: 0054 6b19 4000 4001 b82b 0a00 0164 0a00 .Tk..@...+...d..
0x0020: 0201 0800 2939 1f2c 0001 e4c8 d15a 0000 ...9.....Z..
0x0030: 0000 2fa3 0b00 0000 0000 1011 1213 1415 ..../.....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 .....!""#%&
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*+,-./012345
0x0060: 3637 67 .....
02:24:52.821422 IP 10.0.2.1 > 10.0.1.100: ICMP echo reply, id 7980, seq 1, lengt
h 64
0x0000: 0000 0000 0001 0000 0000 0001 0800 4500 .....E.
0x0010: 0054 c81d 0000 4001 9b27 0a00 0201 0a00 .T...@.....d..
0x0020: 0164 0000 3139 1f2c 0001 e4c8 d15a 0000 .d..19.....Z..
0x0030: 0000 2fa3 0b00 0000 0000 1011 1213 1415 ..../.....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 .....!""#%&
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*+,-./012345
0x0060: 3637 67 .....
02:26:57.933257 IP 10.0.2.100 > 10.0.1.1: ICMP echo request, id 8017, seq 1, len
gth 64
0x0000: 0000 0000 0001 0000 0000 0001 0800 4500 .....E.
0x0010: 0054 1a8b 4000 4001 08ba 0a00 0264 0a00 .T...@.....d..
0x0020: 0101 0800 e9ce 1f51 0001 61c9 d15a 0000 .....Q...a..Z..
0x0030: 0000 efe7 0d00 0000 0000 1011 1213 1415 .....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 .....!""#%&
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*+,-./012345
0x0060: 3637 67 .....
```

Figure 14: tpdump for the Router Ping Test

Now we proceed to the check the performance of the router. Note that the flow mods are already installed in the code.

We can observe that the bandwidth is in the range of 19Gbps uplink and 16.3Gbps downlink. This is attributed to the fact that the packets are not passed to the controller once the flow is created inside the router.

The screenshot shows a Mininet terminal window on the left and two iperf windows on the right. The Mininet terminal shows the creation of a network with three hosts (h1, h2, h3) and one switch (s1). The iperf windows show the results of a TCP test between h1 and h3. The h1 window shows a client connecting to h3 on port 5001 and a table of results. The h3 window shows a server listening on port 5001 and a table of results.

```

mininet@mininet-vm:~$ sudo mn --custom mytopo_part1.py --topo mytopo --mac
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet> xterm h1 h3
mininet>

```

Node: h1

```

root@mininet-vm:~# iperf -c 10.0.3.100
Client connecting to 10.0.3.100, TCP port 5001
TCP window size: 85.3 KByte (default)
[ 15] local 10.0.1.100 port 50825 connected with 10.0.3.100 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 15] 0.0-10.0 sec  19.0 GBytes 16.3 Gbits/sec
root@mininet-vm:~#

```

Node: h3

```

root@mininet-vm:~# iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
[ 16] local 10.0.3.100 port 5001 connected with 10.0.1.100 port 50825
[ ID] Interval      Transfer    Bandwidth
[ 16] 0.0-10.0 sec  19.0 GBytes 16.3 Gbits/sec

```

Figure 15: iperf Testing for TCP

The screenshot shows a Mininet terminal window on the left and two iperf windows on the right. The Mininet terminal shows the creation of a network with three hosts (h1, h2, h3) and one switch (s1). The iperf windows show the results of a UDP test between h1 and h3. The h1 window shows a client connecting to h3 on port 5001 and a table of results. The h3 window shows a server listening on port 5001 and a table of results.

```

mininet@mininet-vm:~$ sudo mn --custom mytopo_part1.py --topo mytopo --mac
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet> xterm h1 h3
mininet>

```

Node: h1

```

root@mininet-vm:~# iperf -c 10.0.3.100 -u
Client connecting to 10.0.3.100, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)
[ 15] local 10.0.1.100 port 48927 connected with 10.0.3.100 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 15] 0.0-10.0 sec  1.25 MBytes 1.05 Mbits/sec
[ 15] Sent 893 datagrams
[ 15] Server Report:
[ 15] 0.0-10.0 sec  1.25 MBytes 1.05 Mbits/sec  0.009 ms  0/ 893 (0%)
root@mininet-vm:~#

```

Node: h3

```

root@mininet-vm:~# iperf -s -u
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
[ 15] local 10.0.3.100 port 5001 connected with 10.0.1.100 port 48927
[ ID] Interval      Transfer    Bandwidth      Jitter   Lost/Total Datagrams
[ 15] 0.0-10.0 sec  1.25 MBytes 1.05 Mbits/sec  0.009 ms  0/ 893 (0%)

```

Figure 16: iperf Testing for UDP

PART 2

Advanced Topology:

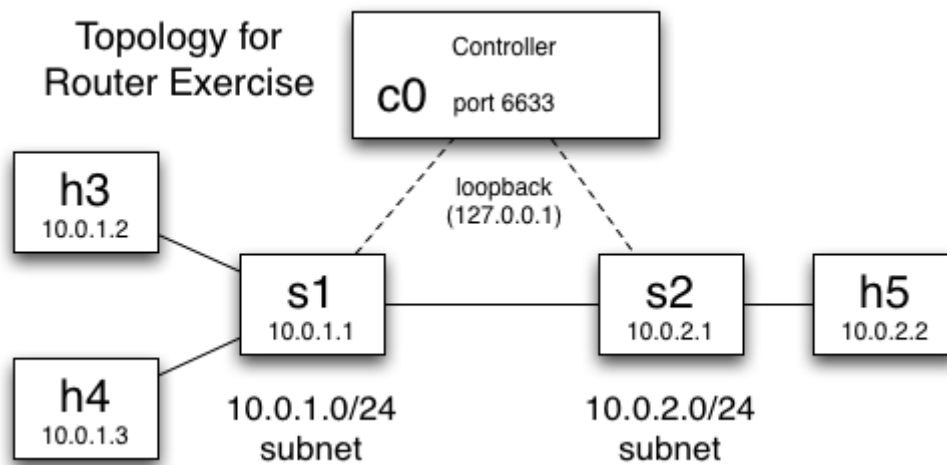


Figure 17: Advanced Topology

The above topology is configured in the *mytopo_part2.py*.

It is to be noted that the s1 – s2 interface is considered as another subnet and we assign the s1 interface an IP of

As far as the router code is concerned, we had to create one routing table each for switched s1 and s2. The code is available in *router_part2.py*.

First, start up the mininet virtual network:

```
$ sudo mn --custom mytopo_part2.py --topo mytopo2 --mac
```

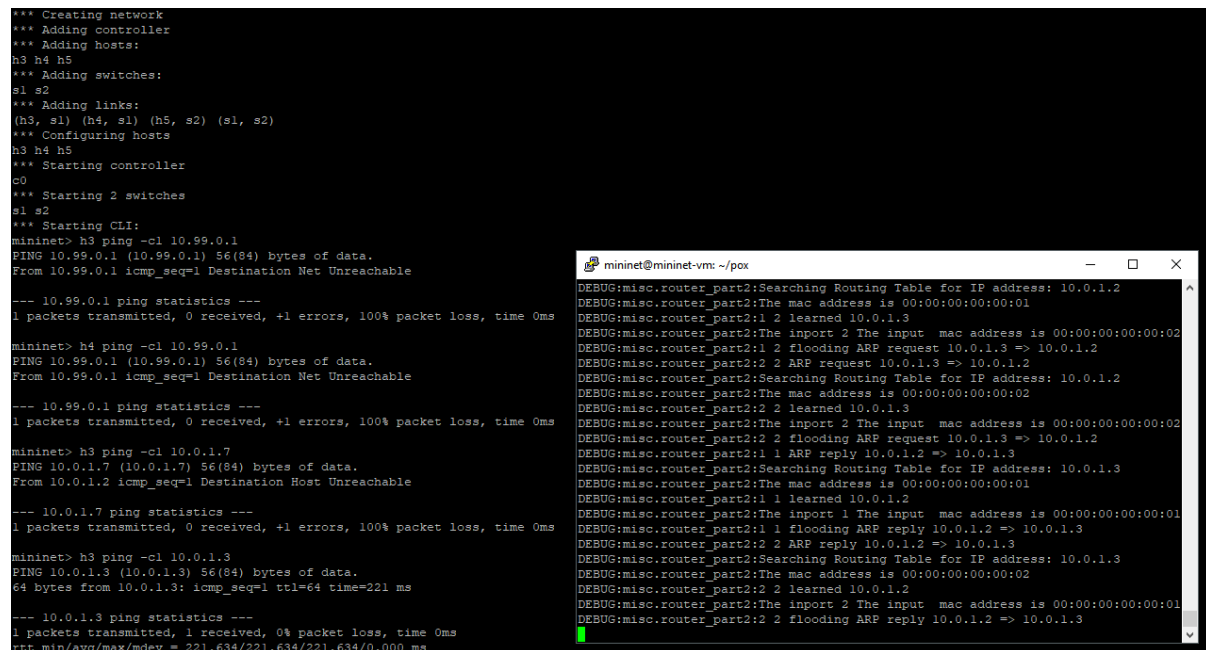
Then we start up the controller after killing all other controllers:

```
$ sudo killall controller
```

```
$ ./pox.py log.level --DEBUG misc.router_part2 misc.full_payload
```

We can carry out the tests similar to the Part 1 Router Exercise.

- A. First, we attempt to ping a host in an unknown address range.



```
*** Creating network
*** Adding controller
*** Adding hosts:
h3 h4 h5
*** Adding switches:
s1 s2
*** Adding links:
(h3, s1) (h4, s1) (h5, s2) (s1, s2)
*** Configuring hosts
h3 h4 h5
*** Starting controller
c0
*** Starting 2 switches
s1 s2
*** Starting CLI:
mininet> h3 ping -c1 10.99.0.1
PING 10.99.0.1 (10.99.0.1) 56(84) bytes of data.
From 10.99.0.1 icmp_seq=1 Destination Net Unreachable

--- 10.99.0.1 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet> h4 ping -c1 10.99.0.1
PING 10.99.0.1 (10.99.0.1) 56(84) bytes of data.
From 10.99.0.1 icmp_seq=1 Destination Net Unreachable

--- 10.99.0.1 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet> h3 ping -c1 10.0.1.7
PING 10.0.1.7 (10.0.1.7) 56(84) bytes of data.
From 10.0.1.2 icmp_seq=1 Destination Host Unreachable

--- 10.0.1.7 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet> h3 ping -c1 10.0.1.3
PING 10.0.1.3 (10.0.1.3) 56(84) bytes of data.
64 bytes from 10.0.1.3: icmp_seq=1 ttl=64 time=221 ms

--- 10.0.1.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 221.634/221.634/221.634/0.000 ms

DEBUG:misc.router_part2:Searching Routing Table for IP address: 10.0.1.2
DEBUG:misc.router_part2:The mac address is 00:00:00:00:00:01
DEBUG:misc.router_part2:1 2 learned 10.0.1.3
DEBUG:misc.router_part2:The input 2 The input mac address is 00:00:00:00:00:02
DEBUG:misc.router_part2:1 2 flooding ARP request 10.0.1.3 => 10.0.1.2
DEBUG:misc.router_part2:2 2 ARP request 10.0.1.3 => 10.0.1.2
DEBUG:misc.router_part2:Searching Routing Table for IP address: 10.0.1.2
DEBUG:misc.router_part2:The mac address is 00:00:00:00:00:02
DEBUG:misc.router_part2:2 2 learned 10.0.1.3
DEBUG:misc.router_part2:The input 2 The input mac address is 00:00:00:00:00:02
DEBUG:misc.router_part2:2 2 flooding ARP request 10.0.1.3 => 10.0.1.2
DEBUG:misc.router_part2:1 1 ARP reply 10.0.1.2 => 10.0.1.3
DEBUG:misc.router_part2:Searching Routing Table for IP address: 10.0.1.3
DEBUG:misc.router_part2:The mac address is 00:00:00:00:00:01
DEBUG:misc.router_part2:1 1 learned 10.0.1.2
DEBUG:misc.router_part2:The input 1 The input mac address is 00:00:00:00:00:01
DEBUG:misc.router_part2:1 1 flooding ARP reply 10.0.1.2 => 10.0.1.3
DEBUG:misc.router_part2:2 2 ARP reply 10.0.1.2 => 10.0.1.3
DEBUG:misc.router_part2:Searching Routing Table for IP address: 10.0.1.3
DEBUG:misc.router_part2:The mac address is 00:00:00:00:00:02
DEBUG:misc.router_part2:2 2 learned 10.0.1.2
DEBUG:misc.router_part2:The input 2 The input mac address is 00:00:00:00:00:01
DEBUG:misc.router_part2:2 2 flooding ARP reply 10.0.1.2 => 10.0.1.3
```

Figure 18: Pinging to unknown and known addresses

The above test gives the expected result. When we try to ping an unknown host, we get a **Destination Net Unreachable** or **Destination Host Unreachable** depending on the address we ping. It can also be seen that pinging a valid host is successful.

- B. Now we try to ping all the hosts in different network, ie., h3 ping h5, h4 ping h5, h5 ping h3 and h5 ping h4. With the successful completion of these tests, we can confirm that the packets sent to hosts on a known address range have their MAC dst field changed to that of the next-hop router.
- C. Next, we check if the router is ping-able.
We can see that the hosts can ping both of the two routers.
- D. Finally, we execute **pingall** in the mininet terminal to verify all the hosts are connected to each other.
pingall executes successfully.

```
mininet> h3 ping -c1 h5
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=1 ttl=64 time=238 ms

--- 10.0.2.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 238.895/238.895/238.895/0.000 ms
mininet> h4 ping -c1 h5
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=1 ttl=64 time=122 ms

--- 10.0.2.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 122.645/122.645/122.645/0.000 ms
mininet> h5 ping -c1 h3
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=104 ms

--- 10.0.1.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 104.859/104.859/104.859/0.000 ms
mininet> h5 ping -c1 h4
PING 10.0.1.3 (10.0.1.3) 56(84) bytes of data.
64 bytes from 10.0.1.3: icmp_seq=1 ttl=64 time=210 ms

--- 10.0.1.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 210.394/210.394/210.394/0.000 ms
mininet> 
```

```
mininet@mininet-vm: ~/pox
DEBUG:misc.router_part2:1 3 learned 10.0.2.2
DEBUG:misc.router_part2:The import 3 The input mac address is 00:00:00:00:00:03
DEBUG:misc.router_part2:Sending 1 buffered packets to 10.0.2.2 from 00-00-00-00-00-01
DEBUG:misc.router_part2:1 3 flooding ARP reply 10.0.2.2 => 10.0.1.3
DEBUG:misc.router_part2:2 2 IP 10.0.1.3 => 10.0.2.2
DEBUG:misc.router_part2:2 2 learned 10.0.1.3
DEBUG:misc.router_part2:port known 3
DEBUG:misc.router_part2:2 2 ARP Request for 10.0.2.2 on behalf of 10.0.1.3
DEBUG:misc.router_part2:2 1 ARP reply 10.0.2.2 => 10.0.1.3
DEBUG:misc.router_part2:Searching Routing Table for IP address: 10.0.1.3
DEBUG:misc.router_part2:The mac address is 00:00:00:00:00:02
DEBUG:misc.router_part2:2 1 learned 10.0.2.2
DEBUG:misc.router_part2:The import 1 The input mac address is 00:00:00:00:00:03
DEBUG:misc.router_part2:Sending 1 buffered packets to 10.0.2.2 from 00-00-00-00-00-02
DEBUG:misc.router_part2:2 1 flooding ARP reply 10.0.2.2 => 10.0.1.3
DEBUG:misc.router_part2:1 3 ARP reply 10.0.2.2 => 10.0.1.3
DEBUG:misc.router_part2:Searching Routing Table for IP address: 10.0.1.3
DEBUG:misc.router_part2:The mac address is 00:00:00:00:00:01
DEBUG:misc.router_part2:1 3 learned 10.0.2.2
DEBUG:misc.router_part2:The import 3 The input mac address is 00:00:00:00:00:03
DEBUG:misc.router_part2:1 3 flooding ARP reply 10.0.2.2 => 10.0.1.3
```

Figure 19: Pinging Known Hosts Across Switches

```
mininet> h3 ping -c1 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
64 bytes from 10.0.2.1: icmp_seq=1 ttl=64 time=7.78 ms

--- 10.0.2.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 7.788/7.788/7.788/0.000 ms
mininet> h3 ping -c1 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=53.8 ms

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 53.866/53.866/53.866/0.000 ms
mininet> h4 ping -c1 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=37.5 ms

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 37.556/37.556/37.556/0.000 ms
mininet> h4 ping -c1 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
64 bytes from 10.0.2.1: icmp_seq=1 ttl=64 time=15.6 ms

--- 10.0.2.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 15.621/15.621/15.621/0.000 ms
mininet> h5 ping -c1 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
64 bytes from 10.0.2.1: icmp_seq=1 ttl=64 time=49.5 ms

--- 10.0.2.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 49.573/49.573/49.573/0.000 ms
mininet> h5 ping -c1 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=41.7 ms

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 41.707/41.707/41.707/0.000 ms
mininet> 
```

Figure 20: Router Ping Test

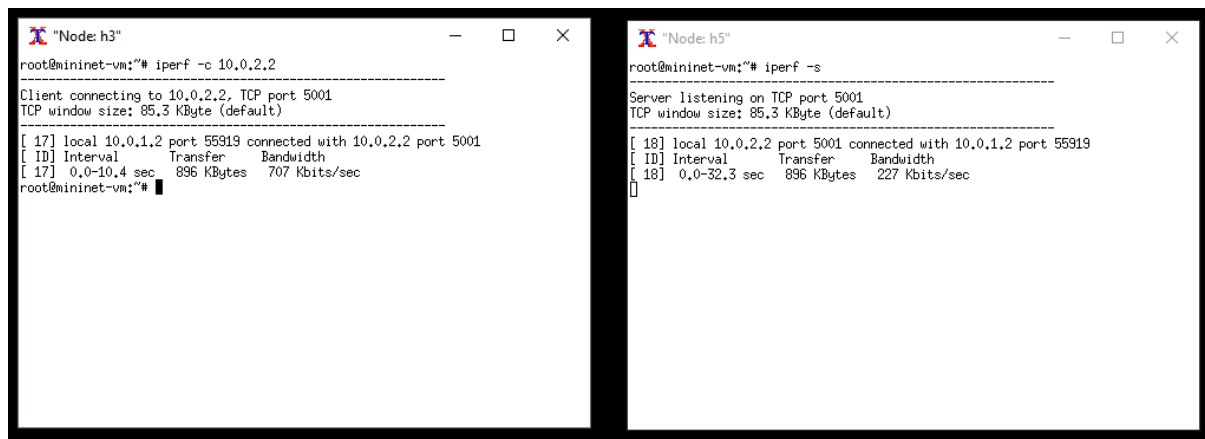
```

mininet> pingall
*** Ping: testing ping reachability
h3 -> h4 h5
h4 -> h3 h5
h5 -> h3 h4
*** Results: 0% dropped (6/6 received)
mininet>

```

Figure 21: pingall Verification Test

Completing all the tests successfully, we run the iperf tests for TCP and UDP.



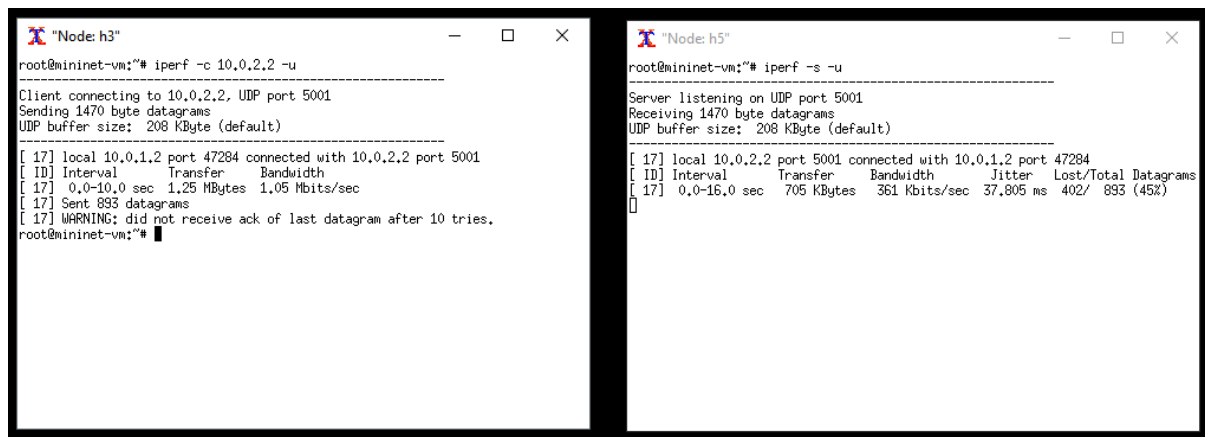
```

"Node: h3"
root@mininet-vm:~# iperf -c 10.0.2.2
-----
Client connecting to 10.0.2.2, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 17] local 10.0.1.2 port 55319 connected with 10.0.2.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 17] 0.0-10.4 sec   896 KBytes  707 Kbits/sec
root@mininet-vm:~#

"Node: h5"
root@mininet-vm:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 18] local 10.0.2.2 port 5001 connected with 10.0.1.2 port 55319
[ ID] Interval      Transfer    Bandwidth
[ 18] 0.0-32.3 sec   896 KBytes  227 Kbits/sec

```

Figure 22: iperf Bandwidth Check for TCP



```

"Node: h3"
root@mininet-vm:~# iperf -c 10.0.2.2 -u
-----
Client connecting to 10.0.2.2, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 17] local 10.0.1.2 port 47284 connected with 10.0.2.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 17] 0.0-10.0 sec   1.25 MBytes  1.05 Mbits/sec
[ 17] Sent 893 datagrams
[ 17] WARNING: did not receive ack of last datagram after 10 tries.
root@mininet-vm:~#

"Node: h5"
root@mininet-vm:~# iperf -s -u
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 17] local 10.0.2.2 port 5001 connected with 10.0.1.2 port 47284
[ ID] Interval      Transfer    Bandwidth      Jitter    Lost/Total Datagrams
[ 17] 0.0-16.0 sec   705 KBytes  361 Kbits/sec  37.805 ms  402/ 893 (45%)

```

Figure 23: iperf Bandwidth Check for UDP

The tests for Advanced Topology are completed successfully hereby.

BONUS

Create Firewall:

The following topology is used in this exercise.

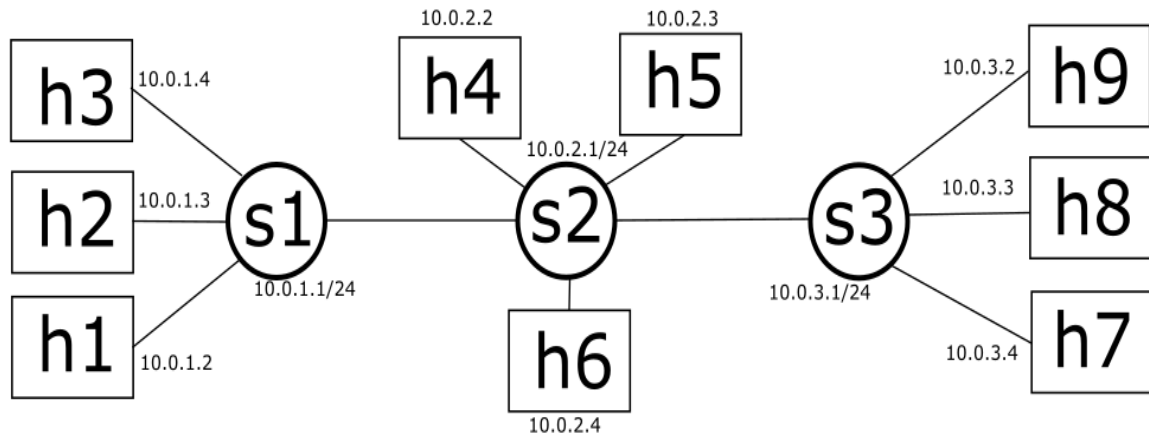


Figure 24: Firewall Exercise Topology

The above topology is configured in *bonus_topo.py* and the firewall implementation is done in *bonus_router.py*.

First, we fire up the network:

```
$ sudo mn --custom bonus_topo.py --topo bonustopo --mac
```

Then, the controller is turned on:

```
$ sudo killall controller
```

```
$ ./pox.py log.level --DEBUG misc.bonus_router misc.full_payload
```

Now we can start the test once the controller is connected.

We have configured a firewall for host h4, therefore, all the incoming and outgoing packets to and from h4 are blocked. We verify this using “pingall”.

```
mininet@mininet-vm:~$ sudo mn --custom bonus_topo.py --topo bonustopo --mac
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s2) (h5, s2) (h6, s2) (h7, s3) (h8, s3) (h9, s3) (s1, s2) (s2, s3)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3
*** Starting CLI:
mininet>
```

Figure 25: Topology for Firewall Exercise


```

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 X h5 h6 h7 h8 h9
h2 -> h1 h3 X h5 h6 h7 h8 h9
h3 -> h1 h2 X h5 h6 h7 h8 h9
h4 -> X X X X X X X X
h5 -> h1 h2 h3 X h6 h7 h8 h9
h6 -> h1 h2 h3 X h5 h7 h8 h9
h7 -> h1 h2 h3 X h5 h6 h8 h9
h8 -> h1 h2 h3 X h5 h6 h7 h9
h9 -> h1 h2 h3 X h5 h6 h7 h8
*** Results: 22% dropped (56/72 received)
mininet>

```

Figure 26: pingall for Firewall Test

```

"Node: h3"
root@mininet-vm:~# iperf -c 10.0.2.2
connect failed; Connection timed out
root@mininet-vm:~#

"Node: h4"
root@mininet-vm:~# iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)

mininet@mininet-vm: ~/pox
DEBUG:misc.bonus_router:ALERT!!! BLOCKED PACKETS FROM 10.0.2.2
DEBUG:misc.bonus_router:ALERT!!! BLOCKED PACKETS FROM 10.0.2.2
DEBUG:misc.bonus_router:ALERT!!! BLOCKED PACKETS FROM 10.0.2.2
DEBUG:misc.bonus_router:ALERT!!! BLOCKED PACKETS FROM 10.0.2.2
DEBUG:misc.bonus_router:ALERT!!! BLOCKED PACKETS FROM 10.0.2.2
DEBUG:misc.bonus_router:ALERT!!! BLOCKED PACKETS FROM 10.0.2.2
DEBUG:misc.bonus_router:ALERT!!! BLOCKED PACKETS FROM 10.0.2.2
DEBUG:misc.bonus_router:ALERT!!! BLOCKED PACKETS FROM 10.0.2.2
DEBUG:misc.bonus_router:ALERT!!! BLOCKED PACKETS FROM 10.0.2.2
DEBUG:misc.bonus_router:ALERT!!! BLOCKED PACKETS FROM 10.0.2.2
DEBUG:misc.bonus_router:ALERT!!! BLOCKED PACKETS FROM 10.0.2.2
DEBUG:misc.bonus_router:ALERT!!! BLOCKED PACKETS FROM 10.0.2.2
DEBUG:misc.bonus_router:ALERT!!! BLOCKED PACKETS FROM 10.0.2.2
DEBUG:misc.bonus_router:ALERT!!! BLOCKED PACKETS FROM 10.0.2.2
DEBUG:misc.bonus_router:ALERT!!! BLOCKED PACKETS FROM 10.0.2.2
DEBUG:misc.bonus_router:DPID: 1 INPORT: 3 ARP request 10.0.1.4 => 10.0.1.1
DEBUG:misc.bonus_router:      The mac address is 00:00:00:00:00:01
DEBUG:misc.bonus_router:      The mac address is 00:00:00:00:00:01
DEBUG:misc.bonus_router:      The mac address is 00:00:00:00:00:01
DEBUG:misc.bonus_router:DPID: 1 INPORT: 3 learned 10.0.1.4
DEBUG:misc.bonus_router:INPORT 3 Input MAC: 00:00:00:00:00:03
DEBUG:misc.bonus_router:      ARP PACKET, known path.
DEBUG:misc.bonus_router:DPID: 1 INPORT: 3 ARP Reply from 10.0.1.1 to 10.0.1.4

```

Figure 27: ping Fail Due to Firewall Setup

This completes the firewall exercise.

CHALLENGES FACED

- We faced some difficulty in setting up and running the host-only interface in the VirtualBox. A detailed explanation of getting this step done was given as a link in the reference git page.
- Understanding the POX library was difficult, especially with little experience in python. The reference code related to `l3_learning.py` played a crucial role in understanding the library.
- Since the editing of python files in through the terminal was less flexible, we used **pscp** in a Windows command window to transfer the files to edit in standard GUI editor. Though minor changes in the final code were done using the vim editor.

CONCLUSION

This project helped us to get a hands-on experience on the OpenFlow protocol which is widely used in today's SDN world. We were able to appreciate the concept of separating the controller from the switch which drastically can change the performance of the network bandwidth.

The order of exercises laid out helped us build up our understanding step by step. In the hub exercise, we could observe the results and understand the code quickly. The Wireshark tool made it even easier for us to verify our understanding. Moving on to the self-learning switch, we ended up knowing how to use the Python dictionary. Then we moved on to create the flow. The performance change after installing flow mods was tremendous. We were able to realize this performance change by bypassing the controller for known source and destination communication.

As we moved on to the router exercise, we made a decision to modularize the code for different functionalities. Now we had to deal with different subnets and implement ARP and ICMP messages. This exercise was completed successfully thanks to the fact that the router is statically configured.

Finally, we also tried to implement a firewall in which connection to specific ports are rejected. This is highly configurable, i.e., we can block based on IP address or application port number or any other parameters. Thus, this exercise shows how OpenFlow can do basic layer 4 functionality.

REFERENCES

- <https://github.com/mininet/openflow-tutorial/wiki>
- <http://mininet.org/>
- <https://openflow.stanford.edu/display/ONL/POX+Wiki/>