EE555 Fall 2019 Major Project

Design of OpenFlow controller using Python POX Library

Huayue Hua huayuehu@usc.edu
Mengdi Yuan mengdiyu@usc.edu

Abstract

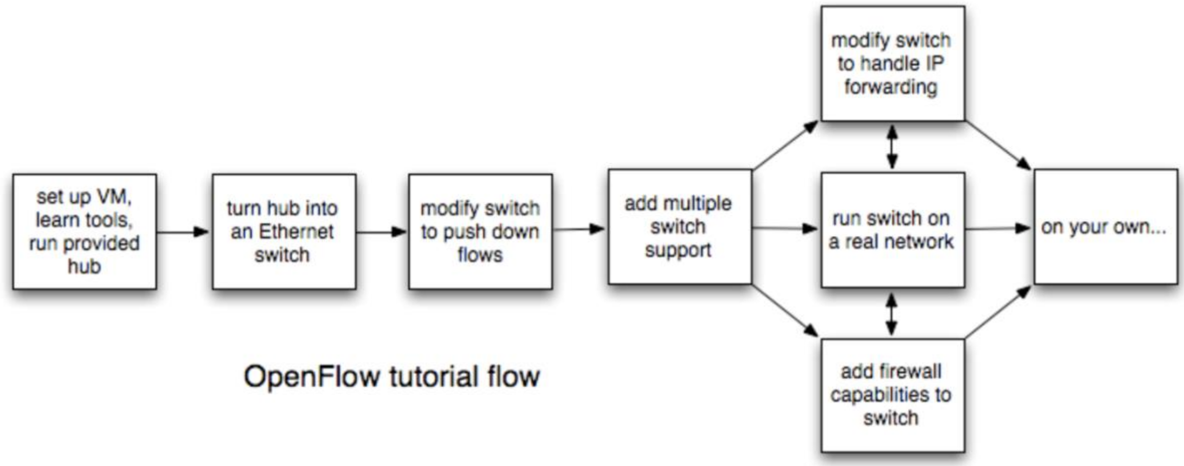


Figure 1 OpenFlow tutorial flow

OpenFlow is an open interface for remotely controlling the forwarding tables in network switches, routers, and access points. Upon this low-level primitive, researchers can build networks with new high-level properties. For example, OpenFlow enables more secure default-off networks, wireless networks with smooth handoffs, scalable data center networks, host mobility, more energy-efficient networks and new wide-area networks – to name a few.

We use POX in this project, POX is a Python-based SDN controller platform geared towards research and education.

In our project, there are five scenarios:
Scenario 1: Create a Learning Switch
Scenario 2: Router Exercise
Scenario 3: Advanced Topology
Scenario 4: Loop Topology
Bonus Scenario: Firewall

Scenario 1: Create a Learning Switch

In this scenario, we design the controller so that the packet forwarding switch acts like a Layer 2 switch.

I. Topology

Topology file is not needed, the topology is like this:

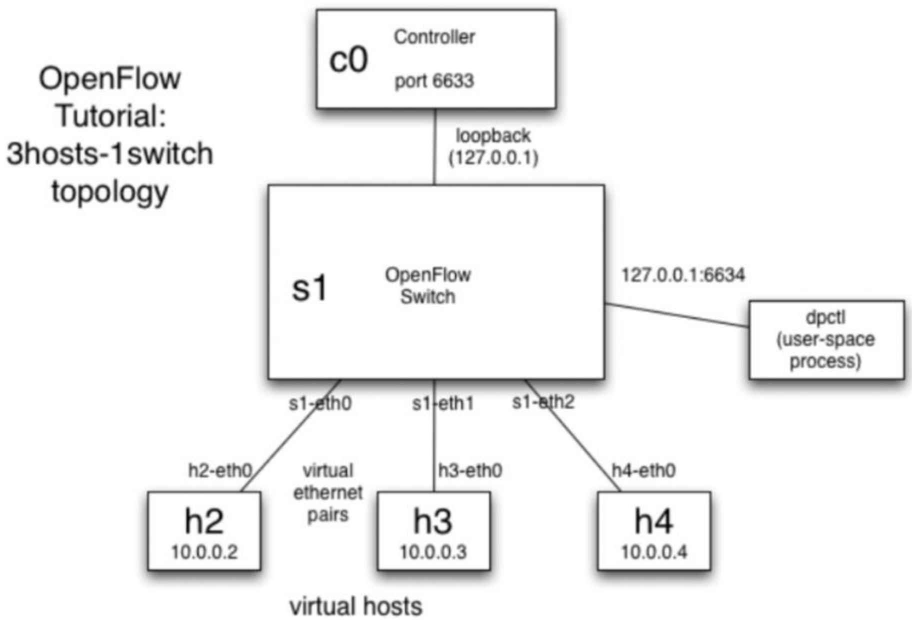


Figure 2 Scenario 1 topology

To create this network in the VM, open an SSH terminal and enter command below:

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

This tells Mininet to start up a 3-host, single switch topology, set the MAC address of each host equal to its IP, and point to a remote controller which defaults to the local host. These 3 hosts are connected to the switch and they are in the same LAN. The command sudo mn -c is to make sure everything is "clean" in Mininet

II. Controller

In this scenario, we need to modify the code so that the hub will act as an L2 learning switch.

1. Command to run the controller file

The controller file we use is ***of_tutorial.py*** file. It needs to be stored in the location "~/pox/pox/misc/". To run the file, open another SSH terminal and enter commands below:

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

This tells POX to enable verbose logging and to start the of_tutorial component. The switches may take a little bit of time to connect. When an OpenFlow switch loses its connection to a controller, it will generally increase the period between which it attempts to contact the controller, up to a maximum of 15 seconds. Since the OpenFlow switch has not connected yet, this delay may be anything between 0 and 15 seconds. If this is too long to wait, the switch can be configured to wait no more than N seconds using the --max-backoff parameter. Alternately, you exit Mininet to remove the switch, start the controller, and then start Mininet to immediately connect.

Wait until the application indicates that the OpenFlow switch has connected. When the switch connects, POX will print something like this:

```
INFO:openflow.of_01:[Con 1/1] Connected to 00-00-00-00-00-01
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 1]
```

2. Design of the controller file

The switch will examine each packet and learn the source-port mapping. Thereafter, the source MAC address will be associated with the port. If the destination of the packet is already associated with some port, the packet will be sent to the given port, else it will be flooded on all ports of the switch.

When implementing to the topology of this scenario, the process is as follows： if h1 is sending packets to hosts that not in the network, h2 and h3 will get broadcast ARP request three times. If it is the first time that h1 sends packets to h2, h2 will send ARP reply and begin to communicate with h1. h3 will only receive an ARP request and drop it. If it is the first time that h1 sends packets to h3, h3 will respond and h2 will drop the packet. When the switch learnt above, next time when h1 sends packets to h2, it will only forward to h2 so that h3 will not hear it.

III. Process

First, set up the network topology and controller connections as shown above. Then, verify reachability. Mininet should be running in the first SSH terminal, along with the POX switch in a second SSH terminal. In the first SSH terminal, run **pingall** command:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

Enter command in the first SSH terminal to open terminals for node h1, h2, and h3, in the node host terminals to test ping command:

```
mininet> xterm h1 h2 h3
```

Below shows the result of **h2 ping h3**:

"Node: h2"

```
root@mininet-vm:~# ping -c1 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=40.0 ms

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 40.028/40.028/40.028/0.000 ms
root@mininet-vm:~#
```

"Node: h3"

```
root@mininet-vm:~# 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 262144 bytes
17:20:54.194870 IP 10.0.0.2 > 10.0.0.3: ICMP echo request, id 10772, seq 1, leng
th 64
        0x0000:  0000 0000 0003 0000 0000 0002 0800 4500  ..............E.
        0x0010:  0054 9284 4000 4001 9420 0a00 0002 0a00  .T..@.@.........
        0x0020:  0003 0800 6820 2a14 0001 f620 df5d 0000  ....h.*......]..
        0x0030:  0000 cf78 0200 0000 0000 1011 1213 1415  ...x............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
17:20:54.194896 IP 10.0.0.3 > 10.0.0.2: ICMP echo reply, id 10772, seq 1, length
 64
        0x0000:  0000 0000 0002 0000 0000 0003 0800 4500  ..............E.
        0x0010:  0054 854d 0000 4001 e157 0a00 0003 0a00  .T.M..@..W......
        0x0020:  0002 0000 7020 2a14 0001 f620 df5d 0000  ....p.*......]..
        0x0030:  0000 cf78 0200 0000 0000 1011 1213 1415  ...x............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
17:20:59.195704 ARP, Request who-has 10.0.0.2 tell 10.0.0.3, length 28
        0x0000:  0000 0000 0002 0000 0000 0003 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0003 0a00 0003  ................
```

Now, in the first SSH terminal, to test our controller-based Ethernet switch, we verify the behavior by running iperf:

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['42.9 Gbits/sec', '43.0 Gbits/sec']
mininet>
```

Now test **iperf TCP and UDP between h2 and h3**:

"Node: h2"

```
root@mininet-vm:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 16] local 10.0.0.2 port 5001 connected with 10.0.0.3 port 40506
[ ID] Interval       Transfer     Bandwidth
[ 16]  0.0-13.5 sec  10.2 MBytes  6.37 Mbits/sec
^Croot@mininet-vm:~# iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.0.2 port 5001 connected with 10.0.0.3 port 58619
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[ 15]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   2.036 ms    0/  893 (0%)
```

"Node: h3"

```
root@mininet-vm:~# iperf -c 10.0.0.2
------------------------------------------------------------
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.0.3 port 40506 connected with 10.0.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 15]  0.0-10.8 sec  10.2 MBytes  7.97 Mbits/sec
root@mininet-vm:~# iperf -c 10.0.0.2 -u
------------------------------------------------------------
Client connecting to 10.0.0.2, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.0.3 port 58619 connected with 10.0.0.2 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   2.035 ms    0/  893 (0%)
root@mininet-vm:~#
```

Scenario 2: Router Exercise

In this exercise, we make a static Layer-3 switch. It's not exactly a router, because it won't decrement the IP TTL and recompute the checksum at each hop. However, it will match on masked IP prefix ranges, just like a real router.

I. Topology

Topology file for this scenario is ***topology2.py*** file. It needs to be stored in the location "~/mininet/custom/". The topology is like this:

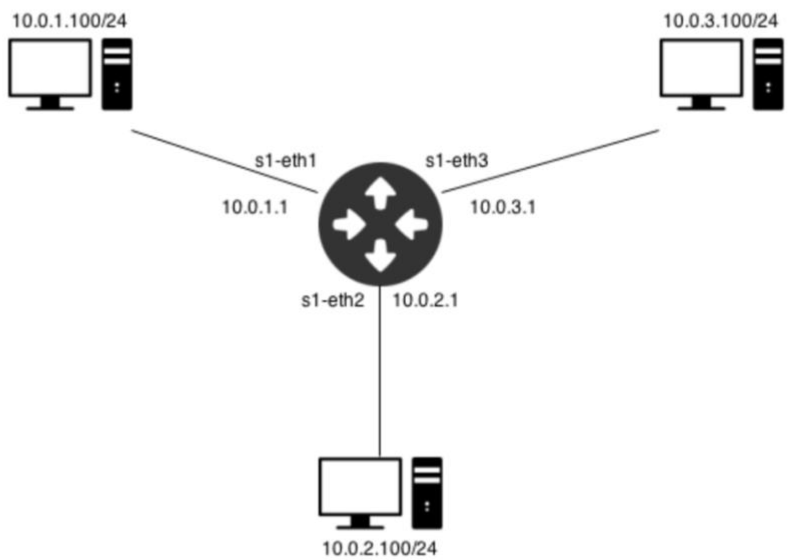


Figure 3 Scenario 2 topology

According to the above topology, we set up IP configuration on each host to force each one to send to the gateway for destination IPs that are outside of their configured subnet in *topology2.py*. Also, we configure each host with a subnet, IP, gateway, and subnet mask.

To start this network in the VM, open an SSH terminal and enter command below to run the topology file:

```
$ sudo mn -c
$ sudo mn --custom topology2.py --topo topology2 --mac --
controller=remote,ip=127.0.0.1,port=6633
```

II. Controller

In this scenario, we need to modify the code so that the hub will act as an L3 learning switch.

1. Command to run the controller file

The controller file we use is ***controller2.py*** file. It needs to be stored in the location "~/pox/pox/misc/". To run the file, open another SSH terminal and enter commands below:

```
$ cd pox
$ sudo ./pox.py log.level --DEBUG misc.controller2 misc.full_payload
```

2. Design of the controller file

Each network node will have a configured subnet. If a packet is destined for a host within that subnet, the node acts as a switch and forwards the packet with no changes, to a known port or broadcast, just like in the previous exercise. If a packet is destined for some IP address for which the router knows the next hop, it should modify the Layer-2 destination and forward the packet to the correct port.

Different from former scenario, this scenario we control the switch to act like a router. A router generally has to respond to ARP requests. Ethernet broadcasts which will be forwarded to the controller. Controller construct ARP replies and forward them out the appropriate ports. Additionally, controller may receive ICMP echo (ping) requests for the router, which it should respond to. Packets for unreachable subnets should be responded to with ICMP network unreachable messages.

III. Process

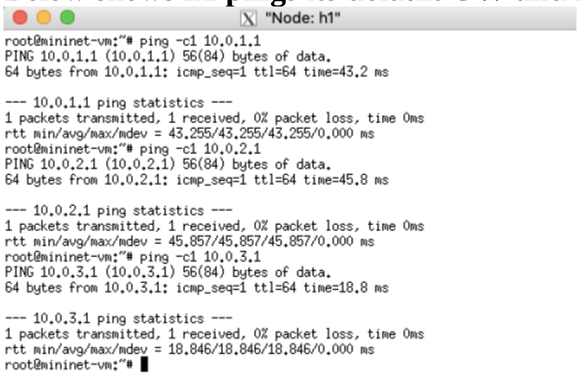
First, set up the network topology and controller connections as shown above. Then, verify reachability. Mininet should be running in the first SSH terminal, along with the POX switch in a second SSH terminal. In the first SSH terminal, run **pingall** command:

```
[mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

Enter commands in the first SSH terminal to open terminals for node h1, h2, and h3, in the node hosts terminal to test ping command:

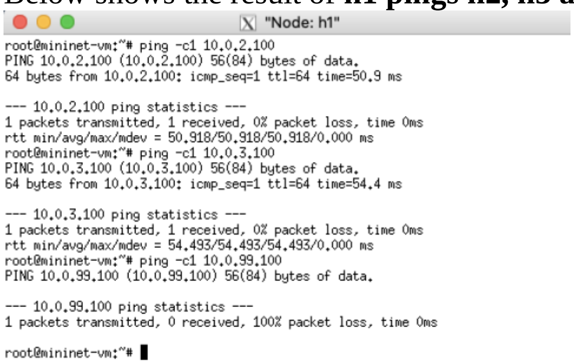
```
mininet> xterm h1 h2 h3
```

Below shows **h1 pings its default GW and router IP2 and router IP3**:



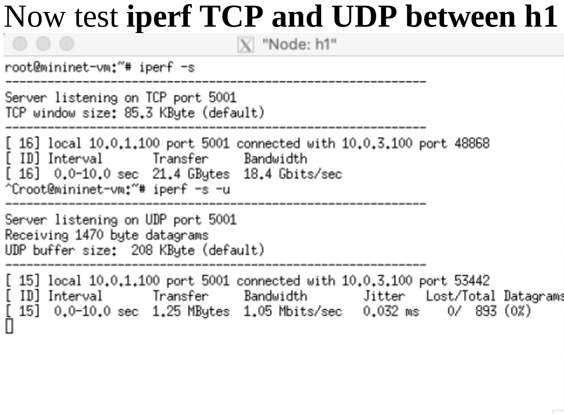
```
"Node: h1"
root@mininet-vm:~# ping -c1 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=43.2 ms

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 43.255/43.255/43.255/0.000 ms
root@mininet-vm:~# 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=45.8 ms

--- 10.0.2.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 45.857/45.857/45.857/0.000 ms
root@mininet-vm:~# 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=18.8 ms

--- 10.0.3.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 18.846/18.846/18.846/0.000 ms
root@mininet-vm:~#
```

Below shows the result of **h1 pings h2, h3 and unknown host**:

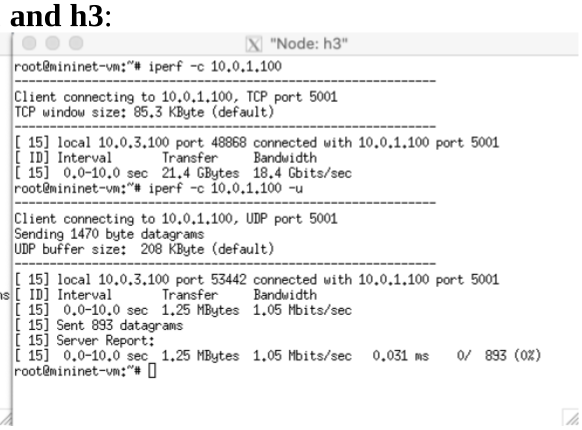


```
"Node: h1"
root@mininet-vm:~# ping -c1 10.0.2.100
PING 10.0.2.100 (10.0.2.100) 56(84) bytes of data.
64 bytes from 10.0.2.100: icmp_seq=1 ttl=64 time=50.9 ms

--- 10.0.2.100 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 50.918/50.918/50.918/0.000 ms
root@mininet-vm:~# ping -c1 10.0.3.100
PING 10.0.3.100 (10.0.3.100) 56(84) bytes of data.
64 bytes from 10.0.3.100: icmp_seq=1 ttl=64 time=54.4 ms

--- 10.0.3.100 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 54.493/54.493/54.493/0.000 ms
root@mininet-vm:~# ping -c1 10.0.99.100
PING 10.0.99.100 (10.0.99.100) 56(84) bytes of data.

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

root@mininet-vm:~#
```

Now test **iperf TCP and UDP between h1 and h3**:



```
"Node: h1"
root@mininet-vm:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 16] local 10.0.1.100 port 5001 connected with 10.0.3.100 port 48868
[ ID] Interval       Transfer     Bandwidth
[ 16]  0.0-10.0 sec  21.4 GBytes  18.4 Gbits/sec
^Croot@mininet-vm:~# iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.1.100 port 5001 connected with 10.0.3.100 port 53442
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[ 15]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.032 ms    0/  893 (0%)
```



```
"Node: h3"
root@mininet-vm:~# iperf -c 10.0.1.100
------------------------------------------------------------
Client connecting to 10.0.1.100, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.3.100 port 48868 connected with 10.0.1.100 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 15]  0.0-10.0 sec  21.4 GBytes  18.4 Gbits/sec
root@mininet-vm:~# iperf -c 10.0.1.100 -u
------------------------------------------------------------
Client connecting to 10.0.1.100, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.3.100 port 53442 connected with 10.0.1.100 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   0.031 ms    0/  893 (0%)
root@mininet-vm:~#
```

Scenario 3: Advanced Topology

Process for this scenario are the same for the scenario as the previous one. The only difference is the topology, which includes two routers controlling different subnets.

I. Topology

Topology file for this scenario is ***topology3.py*** file. It needs to be stored in the location "~/mininet/custom/". The topology is like this:

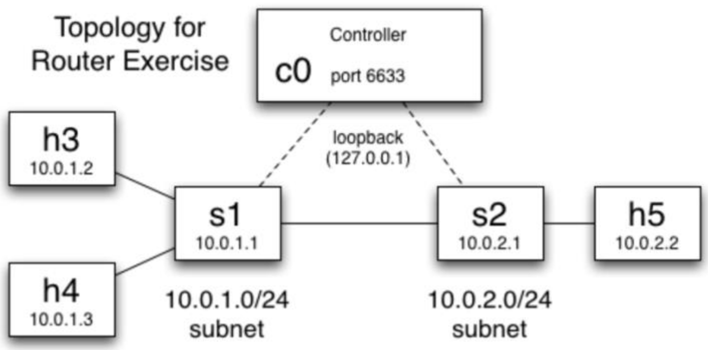


Figure 4 Scenario 3 topology

According to the above topology, we set up IP configuration on each host to force each one to send to the gateway for destination IPs that are outside of their configured subnet in *topology3.py*. Also, we configure each host with a subnet, IP, gateway, and subnet mask.

To start this network in the VM, open an SSH terminal and enter command below to run the topology file:

```
$ sudo mn -c
$ sudo mn --custom topology3.py --topo topology3 --mac --
controller=remote,ip=127.0.0.1,port=6633
```

II. Controller

In this scenario, we need to modify the code so that the hub will act as an L3 learning switch.

1. Command to run the controller file

The controller file we use is ***controller3.py*** file. It needs to be stored in the location "~/pox/pox/misc/". To run the file, open another SSH terminal and enter commands below:

```
$ cd pox
$ sudo ./pox.py log.level --DEBUG misc.controller3 misc.full_payload
```

2. Design of the controller file

Each network node will have a configured subnet. If a packet is destined for a host within that subnet, the node acts as a switch and forwards the packet with no changes, to a known port or broadcast, just like in the previous exercise. If a packet is destined for some IP address for which the router knows the next hop, it should modify the Layer-2 destination and forward the packet to the correct port.

Same as last scenario, we control the switch to act like a router. A router generally has to respond to ARP requests. Ethernet broadcasts which will be forwarded to the controller. Controller construct ARP replies and forward them out the appropriate ports. Additionally, controller may receive ICMP echo (ping) requests for the router, which it should respond to. Packets for unreachable subnets should be responded to with ICMP network unreachable messages.

III. Process

First, set up the network topology and controller connections as shown above. Then, verify reachability. Mininet should be running in the first SSH terminal, along with the POX switch in a second SSH terminal. In the first SSH terminal, run **pingall** command:

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

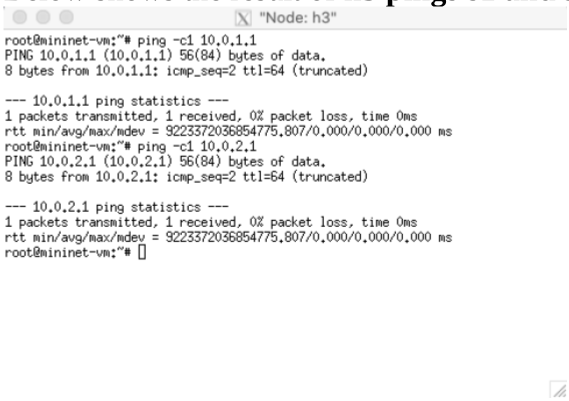
Enter commands in the first SSH terminal to open terminals for node h3, h4, and h5, in the node host terminals to test ping command:

```
mininet> xterm h3 h4 h5
```

Below shows **h3 pings h4 and h5**:

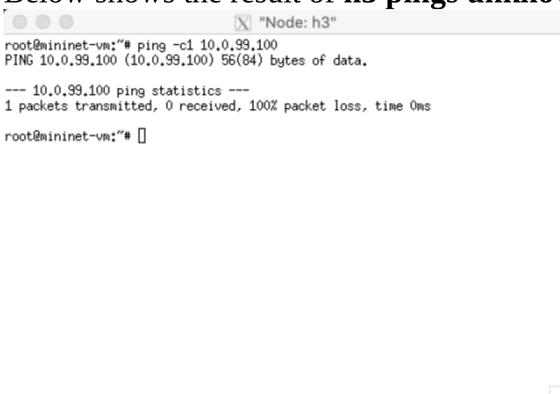
```
"Node: h3"
root@mininet-vm:~# ping -c1 10.0.1.3
PING 10.0.1.3 (10.0.1.3) 56(84) bytes of data.
64 bytes from 10.0.1.3: icmp_seq=1 ttl=64 time=66.8 ms

--- 10.0.1.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 66.846/66.846/66.846/0.000 ms
root@mininet-vm:~# ping -c1 10.0.2.2
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=52.6 ms

--- 10.0.2.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 52.604/52.604/52.604/0.000 ms
root@mininet-vm:~# 
```

Below shows the result of **h3 pings s1 and s2**:



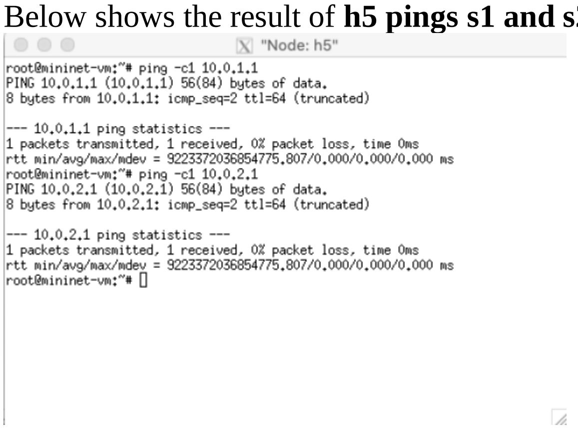
```
root@mininet-vm:~# ping -c1 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
8 bytes from 10.0.1.1: icmp_seq=2 ttl=64 (truncated)

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
root@mininet-vm:~# ping -c1 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
8 bytes from 10.0.2.1: icmp_seq=2 ttl=64 (truncated)

--- 10.0.2.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
root@mininet-vm:~# 
```

Below shows the result of **h3 pings unknown host**:



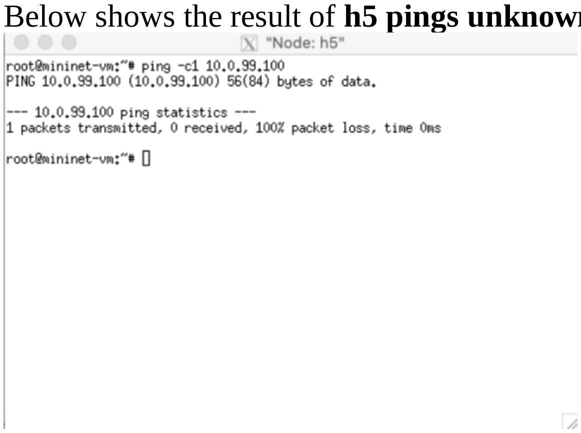
```
root@mininet-vm:~# ping -c1 10.0.99.100
PING 10.0.99.100 (10.0.99.100) 56(84) bytes of data.

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

root@mininet-vm:~# 
```

Below shows the result of **h5 pings s1 and s2**:



```
root@mininet-vm:~# ping -c1 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
8 bytes from 10.0.1.1: icmp_seq=2 ttl=64 (truncated)

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
root@mininet-vm:~# ping -c1 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
8 bytes from 10.0.2.1: icmp_seq=2 ttl=64 (truncated)

--- 10.0.2.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
root@mininet-vm:~# 
```

Below shows the result of **h5 pings unknown host**:



```
root@mininet-vm:~# ping -c1 10.0.99.100
PING 10.0.99.100 (10.0.99.100) 56(84) bytes of data.

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

root@mininet-vm:~# 
```

Now test **iperf TCP and UDP between h3 and h4**:

"Node: h3"

```
root@mininet-vm:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 18] local 10.0.1.2 port 5001 connected with 10.0.1.3 port 34812
[ ID] Interval       Transfer     Bandwidth
[ 18]  0.0-10.0 sec  21.3 GBytes  18.3 Gbits/sec
^Croot@mininet-vm:~# iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 17] local 10.0.1.2 port 5001 connected with 10.0.1.3 port 40981
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[ 17]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.022 ms    0/  893 (0%)
```

"Node: h4"

```
root@mininet-vm:~# iperf -c 10.0.1.2
------------------------------------------------------------
Client connecting to 10.0.1.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 17] local 10.0.1.3 port 34812 connected with 10.0.1.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 17]  0.0-10.0 sec  21.3 GBytes  18.3 Gbits/sec
root@mininet-vm:~# iperf -c 10.0.1.2 -u
------------------------------------------------------------
Client connecting to 10.0.1.2, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 17] local 10.0.1.3 port 40981 connected with 10.0.1.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 17]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 17] Sent 893 datagrams
[ 17] Server Report:
[ 17]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.021 ms    0/  893 (0%)
root@mininet-vm:~#
```

Test **iperf TCP and UDP between h3 and h5**:

"Node: h3"

```
root@mininet-vm:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 18] local 10.0.1.2 port 5001 connected with 10.0.2.2 port 52664
[ ID] Interval       Transfer     Bandwidth
[ 18]  0.0-10.0 sec  20.9 GBytes  17.9 Gbits/sec
^Croot@mininet-vm:~# iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 17] local 10.0.1.2 port 5001 connected with 10.0.2.2 port 53691
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[ 17]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.018 ms    0/  893 (0%)
```

"Node: h5"

```
root@mininet-vm:~# iperf -c 10.0.1.2
------------------------------------------------------------
Client connecting to 10.0.1.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 17] local 10.0.2.2 port 52664 connected with 10.0.1.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 17]  0.0-10.0 sec  20.9 GBytes  17.9 Gbits/sec
root@mininet-vm:~# iperf -c 10.0.1.2 -u
------------------------------------------------------------
Client connecting to 10.0.1.2, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 17] local 10.0.2.2 port 53691 connected with 10.0.1.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 17]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 17] Sent 893 datagrams
[ 17] Server Report:
[ 17]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.018 ms    0/  893 (0%)
root@mininet-vm:~#
```

Scenario 4: Loop Topology

This is an extension to the scenario 3 and there exists a loop in the bellowing topology.

I. Topology

Topology file for this scenario is ***topology4.py*** file. It needs to be stored in the location "~/mininet/custom/". The topology is like this:

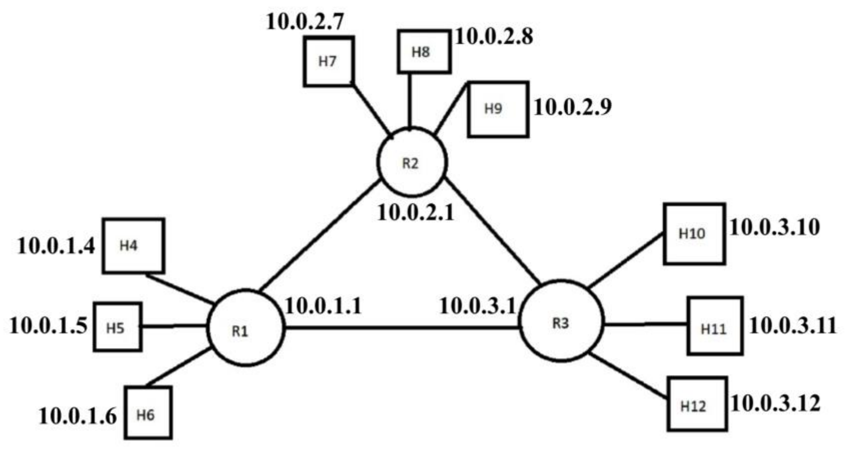


Figure 5 Scenario 4 topology

II. Controller

1. Command to run the controller file

The controller file we use is ***controller4.py*** file. It needs to be stored in the location "~/pox/pox/misc/". To run the file, open another SSH terminal and enter commands below:

```
$ cd pox
$ sudo ./pox.py log.level --DEBUG misc.controller4 misc.full_payload
```

2. Design of the controller file

Each network node will have a configured subnet. If the destination host address belongs to the same LAN, the node acts like a switch and forward packet internally. If the destination does not belong to the same LAN, the packet should be moved to its nearest router.

III. Process

First, set up the network topology and controller connections as shown above. Then, verify reachability. Mininet should be running in the first SSH terminal, along with the POX switch in a second SSH terminal. In the first SSH terminal, run **pingall** command:

```
*** Adding hosts:
h4 h5 h6 h7 h8 h9 h10 h11 h12
*** Adding switches:
s1 s2 s3
*** Adding links:
(h4, s1) (h5, s1) (h6, s1) (h7, s2) (h8, s2) (h9, s2) (h10, s3) (h11, s3) (h
12, s3) (s1, s2) (s1, s3) (s2, s3)
*** Configuring hosts
h4 h5 h6 h7 h8 h9 h10 h11 h12
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h4 -> h5 h6 h7 h8 h9 h10 h11 h12
h5 -> h4 h6 h7 h8 h9 h10 h11 h12
h6 -> h4 h5 h7 h8 h9 h10 h11 h12
h7 -> h4 h5 h6 h8 h9 h10 h11 h12
h8 -> h4 h5 h6 h7 h9 h10 h11 h12
h9 -> h4 h5 h6 h7 h8 h10 h11 h12
h10 -> h4 h5 h6 h7 h8 h9 h11 h12
h11 -> h4 h5 h6 h7 h8 h9 h10 h12
h12 -> h4 h5 h6 h7 h8 h9 h10 h11
*** Results: 0% dropped (72/72 received)
```

Below shows the result of **h4 ping h5**

"Node: h4"

```
root@mininet-vm:~/mininet/custom# ping -c3 10.0.1.5
PING 10.0.1.5 (10.0.1.5) 56(84) bytes of data.
64 bytes from 10.0.1.5: icmp_seq=1 ttl=64 time=57.4 ms
64 bytes from 10.0.1.5: icmp_seq=2 ttl=64 time=46.3 ms
64 bytes from 10.0.1.5: icmp_seq=3 ttl=64 time=27.4 ms

--- 10.0.1.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 27.415/43.738/57.444/12.399 ms
root@mininet-vm:~/mininet/custom#
```

Below shows the result of **h4 ping h7**

"Node: h4"

```
root@mininet-vm:~/mininet/custom# ping -c3 10.0.2.7
PING 10.0.2.7 (10.0.2.7) 56(84) bytes of data.
64 bytes from 10.0.2.7: icmp_seq=1 ttl=64 time=32.1 ms
64 bytes from 10.0.2.7: icmp_seq=2 ttl=64 time=41.0 ms
64 bytes from 10.0.2.7: icmp_seq=3 ttl=64 time=28.6 ms

--- 10.0.2.7 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 28.609/33.940/41.067/5.244 ms
root@mininet-vm:~/mininet/custom#
```

Below shows the result of **h4 ping h10**

"Node: h4"

```
root@mininet-vm:~/mininet/custom# ping -c3 10.0.3.10
PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data.
64 bytes from 10.0.3.10: icmp_seq=1 ttl=64 time=37.1 ms
64 bytes from 10.0.3.10: icmp_seq=2 ttl=64 time=39.1 ms
64 bytes from 10.0.3.10: icmp_seq=3 ttl=64 time=30.4 ms

--- 10.0.3.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 30.471/35.599/39.178/3.722 ms
root@mininet-vm:~/mininet/custom#
```

Below shows the result of **h4 ping R1**

"Node: h4"

```
root@mininet-vm:~/mininet/custom# ping -c3 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
8 bytes from 10.0.1.1: icmp_seq=2 ttl=64 (truncated)
8 bytes from 10.0.1.1: icmp_seq=3 ttl=64 (truncated)
8 bytes from 10.0.1.1: icmp_seq=4 ttl=64 (truncated)

--- 10.0.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
root@mininet-vm:~/mininet/custom#
```

Below shows the result of **h4 ping R2**

"Node: h4"

```
root@mininet-vm:~/mininet/custom# ping -c3 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
8 bytes from 10.0.2.1: icmp_seq=2 ttl=64 (truncated)
8 bytes from 10.0.2.1: icmp_seq=3 ttl=64 (truncated)
8 bytes from 10.0.2.1: icmp_seq=4 ttl=64 (truncated)

--- 10.0.2.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
root@mininet-vm:~/mininet/custom#
```

Below shows the result of **h4 ping R3**

"Node: h4"

```
root@mininet-vm:~/mininet/custom# ping -c3 10.0.3.1
PING 10.0.3.1 (10.0.3.1) 56(84) bytes of data.
8 bytes from 10.0.3.1: icmp_seq=2 ttl=64 (truncated)
8 bytes from 10.0.3.1: icmp_seq=3 ttl=64 (truncated)
8 bytes from 10.0.3.1: icmp_seq=4 ttl=64 (truncated)

--- 10.0.3.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
root@mininet-vm:~/mininet/custom#
```

Now test **iperf TCP between h4 and h5**

"Node: h4"

```
root@mininet-vm:~/mininet/custom# iperf -c 10.0.1.5
------------------------------------------------------------
Client connecting to 10.0.1.5, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 31] local 10.0.1.4 port 45596 connected with 10.0.1.5 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 31]  0.0-10.0 sec  20.3 GBytes  17.5 Gbits/sec
root@mininet-vm:~/mininet/custom# 
```

"Node: h5"

```
root@mininet-vm:~/mininet/custom# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 32] local 10.0.1.5 port 5001 connected with 10.0.1.4 port 45596
[ ID] Interval       Transfer     Bandwidth
[ 32]  0.0-10.0 sec  20.3 GBytes  17.4 Gbits/sec
```

Test **iperf UDP between h4 and h5**

"Node: h4"

```
root@mininet-vm:~/mininet/custom# iperf -c 10.0.1.5 -u
------------------------------------------------------------
Client connecting to 10.0.1.5, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 31] local 10.0.1.4 port 34955 connected with 10.0.1.5 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 31]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 31] Sent 893 datagrams
[ 31] Server Report:
[ 31]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.032 ms    0/  893 (0%)
root@mininet-vm:~/mininet/custom# 
```

"Node: h5"

```
root@mininet-vm:~/mininet/custom# iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 31] local 10.0.1.5 port 5001 connected with 10.0.1.4 port 34955
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[ 31]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.032 ms    0/  893 (0%)
```

Test **iperf TCP between h4 and h7**

"Node: h4"

```
root@mininet-vm:~/mininet/custom# iperf -c 10.0.2.7
------------------------------------------------------------
Client connecting to 10.0.2.7, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 31] local 10.0.1.4 port 42186 connected with 10.0.2.7 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 31]  0.0-10.3 sec  3.12 MBytes  2.54 Mbits/sec
root@mininet-vm:~/mininet/custom# 
```

"Node: h7"

```
root@mininet-vm:~/mininet/custom# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 32] local 10.0.2.7 port 5001 connected with 10.0.1.4 port 42186
[ ID] Interval       Transfer     Bandwidth
[ 32]  0.0-18.5 sec  3.12 MBytes  1.41 Mbits/sec
```

Test **iperf UDP between h4 and h7**

"Node: h4"

```
root@mininet-vm:~/mininet/custom# iperf -c 10.0.2.7 -u
------------------------------------------------------------
Client connecting to 10.0.2.7, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 31] local 10.0.1.4 port 41118 connected with 10.0.2.7 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 31]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 31] Sent 893 datagrams
[ 31] Server Report:
[ 31]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   1.950 ms    0/  893 (0%)
root@mininet-vm:~/mininet/custom# 
```

"Node: h7"

```
root@mininet-vm:~/mininet/custom# iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 31] local 10.0.2.7 port 5001 connected with 10.0.1.4 port 41118
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[ 31]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   1.950 ms    0/  893 (0%)
```

h4 ping an unknown host

"Node: h4"

```
root@mininet-vm:~/mininet/custom# ping -c3 10.0.99.1
PING 10.0.99.1 (10.0.99.1) 56(84) bytes of data.
From 10.0.99.1 icmp_seq=1 Destination Net Unreachable
From 10.0.99.1 icmp_seq=2 Destination Net Unreachable
From 10.0.99.1 icmp_seq=3 Destination Net Unreachable

--- 10.0.99.1 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2003ms

root@mininet-vm:~/mininet/custom# 
```

Bonus Scenario: Firewall

I. Topology

Topology file for this scenario is ***topology5.py*** file. It needs to be stored in the location "~/mininet/custom/". The topology is like this:

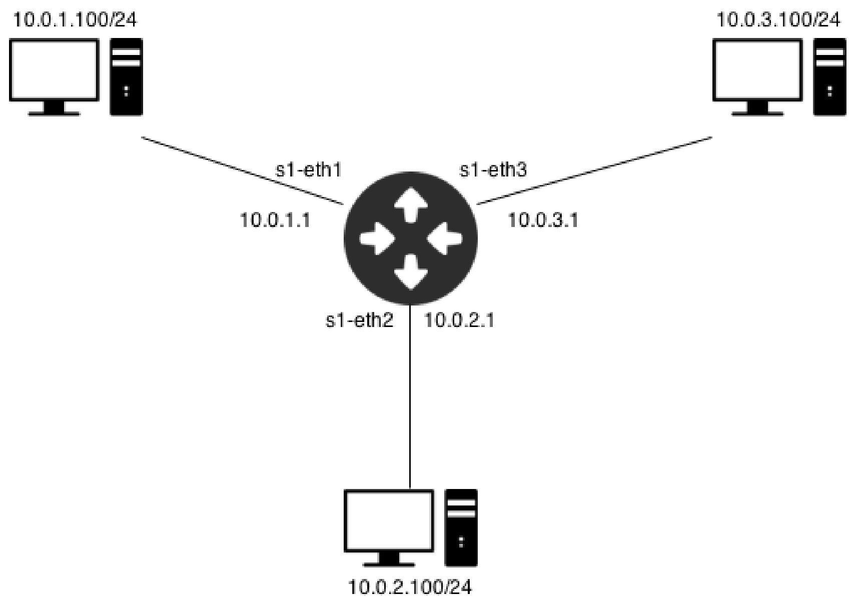


Figure 6 Scenario 5 topology

II. Controller

1. The controller file we use is ***controller5.py*** file. It needs to be stored in the location "~/pox/pox/misc/". To run the file, open another SSH terminal and enter commands below:

```
$ cd pox
$ sudo ./pox.py log.level --DEBUG misc.controller5 misc.full_payload
```

2. Design of the controller file

We are still trying. Functions have not been implemented yet.