CO515: Advances in Computer Networks: Selected Topics - Mininet Lab 01

E/19/446: Wijerathna I.M.K.D.I.

23/03/2024

Lab Environment Setup

I used a lightweight ubuntu os setup which runs over vagrant+virtual box. The following is the lab environment structure I used for this implementation.

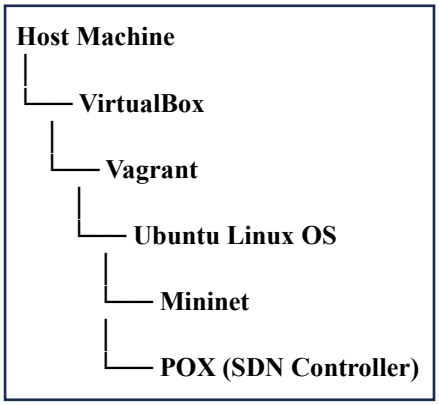

Figure 1: Lab Environment Setup

The reason for using this virtual environment is to avoid mininet networks conflicting with my local private network and other virtual networks already running on my pc. I chose POX as the remote SDN controller over alternatives like Ryu or Pyretic because it is lightweight, easily customizable, and ease of integrable with mininet.

Initial Mininet Test

```
vagrant@sdn-box:~/labtasks$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
```

Figure 2: Initial mininet testing

Running the default mininet network to check if installation was successful.

- *Tried out a few basic mininet commands (pingall, nodes, net)*
- *pingall was successful between all the available hosts.*

Setting Up My Custom Topology

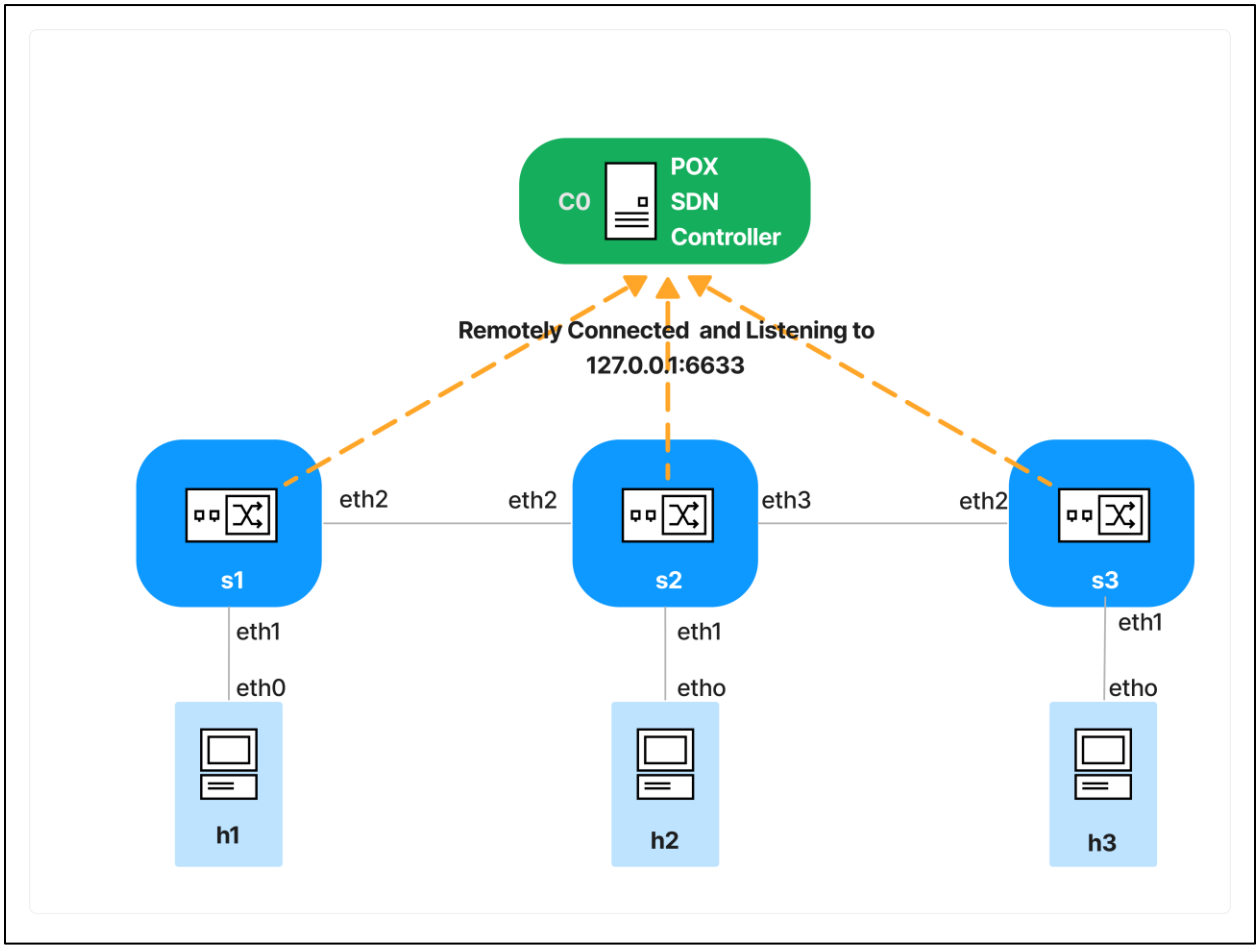

Figure 3: The Custom Multi Switch Topology I used for the Lab

The topology was defined using the following python script:

```
# mytopology.py

from mininet.net import Mininet
from mininet.node import Controller, RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info

def multiSwitch():
    # Create Mininet object
    net = Mininet(controller=RemoteController)

    # Add the pox remote controller to the network
    info('*** Adding controller\n')
    net.addController('c0', controller=RemoteController, ip="127.0.0.1", port=6633)

    # Add hosts to the network
    info('*** Adding hosts\n')
    h1 = net.addHost('h1', ip='10.0.0.1')
```

```
    h2 = net.addHost('h2', ip='10.0.0.2')
    h3 = net.addHost('h3', ip='10.0.0.3')

    # Add switches to the network
    info('*** Adding switch\n')
    s1 = net.addSwitch('s1')
    s2 = net.addSwitch('s2')
    s3 = net.addSwitch('s3')

    # Create links between hosts and switches
    info('*** Creating links\n')
    net.addLink(h1, s1)
    net.addLink(h2, s2)
    net.addLink(h3, s3)

    # Create links between switches
    net.addLink(s1, s2)
    net.addLink(s2, s3)

    # Start the network
    info('*** Starting network\n')
    net.start()

    # Run Mininet CLI
    info('*** Running CLI\n')
    CLI(net)

    # Stop the network
    info('*** Stopping network')
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    multiSwitch()
```

```
vagrant@sdn-box:~/labtasks$ sudo python mytopology.py
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts
*** Adding switch
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Running CLI
*** Starting CLI:
mininet> nodes
available nodes are:
c0 h1 h2 h3 s1 s2 s3
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth2
s2 lo:  s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2
s3 lo:  s3-eth1:h3-eth0 s3-eth2:s2-eth3
c0
```

Running the mini net topology without starting the POX SDN controller

Figure 4: Starting the mininet Topology

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
mininet>
```

Figure 5: Initial pingall

Since, the POX SDN controller hasn't started up yet, there are no openflow rules to handle, permit, drop any packets for the openflow switches s1, s2, s3. ∴ All pings failed as expected.

Connecting the Pox SDN Controller

Running the Pox SDN controller with layer-2 learning openflow rules. These rules allow s1, s2, s3 to learn and forward traffic between all the hosts.

```
vagrant@sdn-box:~/pox$ sudo ./pox.py forwarding.l2_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
INFO:openflow.of_01:[00-00-00-00-00-03 1] connected
INFO:openflow.of_01:[00-00-00-00-00-02 2] connected
```

Figure 6: Initializing Pox SDN Controller

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

Figure 7: pingall after connecting with the Pox SDN Controller

Since, the POX SDN controller has startedt, there are suitable openflow rules that permit traffic between all the hosts in the topology. ∴ All pings succeeded as expected.

Writing Simple OpenFlow Rules to Control Packet Flow through the Network

1. *OpenFlow rule to drop packets from h1(10.0.0.1) to h2(10.0.0.2) on s1*

```
mininet> sh ovs-ofctl add-flow s1 priority=65535,dl_type=0x800,nw_src=10.0.0.1,nw_dst=10.0.0.2,actions=drop
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3
h2 -> X h3
h3 -> h1 h2
*** Results: 33% dropped (4/6 received)
mininet>
```

Since a ping to be successful both echo request and echo reply should not fail. Therefore, both (h1 to h2) and (h2 to h1) pings have to be failed. All the other pings should succeed. The expected behavior is achieved by this OpenFlow rule successfully.

deleting this OpenFlow rule and going back to original setup.

```
mininet> sh ovs-ofctl del-flows s1 "dl_type=0x800,nw_src=10.0.0.1,nw_dst=10.0.0.2"
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

2. *OpenFlow rule on s2 switch to: "if a packet arrives from port-1(eth1) it is always forwarded to port2 (eth2)*

```
mininet>  sh ovs-ofctl add-flow s2 in_port=1,actions=output:2
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 X
h3 -> h1 X
*** Results: 33% dropped (4/6 received)
mininet>
```

Since the rule always directs packet from s2-eth1 to s2-eth2, h2 will not be able to ping h3. Also h3 cannot ping h2 as well. All the other ping should succeed. The expected behavior is achieved by this OpenFlow rule successfully.

Deleting this rule and going back to the original config,

```
mininet> sh ovs-ofctl del-flows s2
mininet>
```

3. *OpenFlow rule to drop all the packet arriving at switch-3*

```
mininet> sh ovs-ofctl add-flow s3 actions=drop
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 X
h3 -> X X
*** Results: 66% dropped (2/6 received)
mininet>
```

Since s3 is directly connected to h3, upon applying this rule, h3 should not be able to ping anyone else. Also, all the other should not be able to ping h3. Rest of the ping should succeed. This expected behavior was achieved successfully using the above OpenFlow rule.

The packet dropping messages was displayed in the Pox SDN Controller's log as well.

```
vagrant@sdn-box:~/pox$ sudo ./pox.py forwarding.l2_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
INFO:openflow.of_01:[00-00-00-00-00-03 1] connected
INFO:openflow.of_01:[00-00-00-00-00-02 2] connected
WARNING:forwarding.l2_learning:Same port for packet from da:61:cd:ab:52:a3 -> b6:1e:3d:94:7f:48 on 00-00-00-00-00-01.2.  Drop.
WARNING:forwarding.l2_learning:Same port for packet from da:61:cd:ab:52:a3 -> b6:1e:3d:94:7f:48 on 00-00-00-00-00-01.2.  Drop.
WARNING:forwarding.l2_learning:Same port for packet from da:61:cd:ab:52:a3 -> b6:1e:3d:94:7f:48 on 00-00-00-00-00-01.2.  Drop.
```

The above rule was deleted as well in order to go back to the original config,

4. *OpenFlow rule on s2 to: "If a packet arrives from h1(10.0.0.1) to h3(10.0.0.3) always forward that to h2(s2-eth2)"*

```
mininet> sh ovs-ofctl add-flow s2 dl_type=0x800,nw_src=10.0.0.1,nw_dst=10.0.0.3,actions=output:2
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 h3
h3 -> X h2
*** Results: 33% dropped (4/6 received)
mininet>
```

As Expected all the other pings except for between h1,h3 have been failed. Moreover, if wireshark has been used in promiscuous mode to capture, then those packets(from h1 to h3) could be received by h2.

Experimenting with Different Link Types (bandwidths and latencies)

In place of the default links, 5 different link types has been used as follows by using TCLink module.

```python
# modified mytopology.py

from mininet.net import Mininet
from mininet.node import Controller, RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink

def multiSwitch():

    net = Mininet(controller=RemoteController)

    info('*** Adding controller\n')
    net.addController('c0', controller=RemoteController, ip="127.0.0.1", port=6633)

    info('*** Adding hosts\n')
    h1 = net.addHost('h1', ip='10.0.0.1')
    h2 = net.addHost('h2', ip='10.0.0.2')
    h3 = net.addHost('h3', ip='10.0.0.3')

    info('*** Adding switch\n')
    s1 = net.addSwitch('s1')
    s2 = net.addSwitch('s2')
    s3 = net.addSwitch('s3')

    info('*** Creating links\n')
    net.addLink(h1, s1,cls=TCLink, bw=10, delay='5ms') #bandwidth of 10 Mbps and latency of 5ms
    net.addLink(h2, s2,cls=TCLink, bw=1000, delay='5ms') #bandwidth of 1 Gbps and latency of 1ms
    net.addLink(h3, s3,cls=TCLink, bw=10, delay='1000ms') #bandwidth of 10 Mbps and latency of 1s
    net.addLink(s1, s2,cls=TCLink, bw=1000, delay='1000ms') #bandwidth of 1 Gbps and latency of 1s
    net.addLink(s2, s3,cls=TCLink, bw=10000, delay='1ms') #bandwidth of 10 Gbps and latency of 1ms

    info('*** Starting network\n')
    net.start()

    info('*** Running CLI\n')
    CLI(net)

    info('*** Stopping network')
```

```
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    multiSwitch()
```

Cleaning up the old topology

```
vagrant@sdn-box:~/labtasks$ sudo mn -c
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
vagrant@sdn-box:~/labtasks$
```

Figure 8: Cleaning the old mininet topology

Starting the new Topology

```
vagrant@sdn-box:~/labtasks$ sudo python mytopology.py
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts
*** Adding switch
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Running CLI
*** Starting CLI:
mininet> nodes
available nodes are:
c0 h1 h2 h3 s1 s2 s3
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s2-eth1 (OK OK)
h3-eth0<->s3-eth1 (OK OK)
s1-eth2<->s2-eth2 (OK OK)
s2-eth3<->s3-eth2 (OK OK)
```

Figure 9: Starting new topology, links are at good conditions

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 h3
h3 -> h1 h2
*** Results: 16% dropped (5/6 received)
mininet> h1 ping h2
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=2138 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=2021 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=2020 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=2020 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=2020 ms
^C
--- 10.0.0.2 ping statistics ---
7 packets transmitted, 5 received, 28% packet loss, time 6022ms
rtt min/avg/max/mdev = 2020.671/2044.507/2138.622/47.075 ms, pipe 3
mininet> h1 ping h3
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=4095 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=4017 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=4014 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=4015 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=4013 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=4015 ms
^C
--- 10.0.0.3 ping statistics ---
9 packets transmitted, 6 received, 33% packet loss, time 8057ms
rtt min/avg/max/mdev = 4013.935/4028.521/4095.087/29.829 ms, pipe 5
mininet>
```

Figure 10: ping between the hosts after implementing the variety of link types

*It could be clearly observed that the delay it takes a packet to complete the icmp round trip is varying a lot. Due the heavy delay in the links between h1 to h3 has caused some packets to drop during the pingall test. The configured max delay link between h1-h2 is 1s; therefore, those roundtrips have taken approximately 2*1s ≈2s dealys as shown in the above Figure-10. The configuration has 2 links with 1s delay between h1-h3. Therefore, it should take twice the time(≈ 4s) it took between h1-h2. This is also resulted as expected.*

Testing Network Performance(using iperf, tcp dump, wireshark)

1. *Testing "tcp dump" with pingall*

```
vagrant@sdn-box:~$ sudo tcpdump -i s1-eth2 -nn -e
tcpdump: WARNING: s1-eth2: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s1-eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
10:19:06.335447 92:05:85:f2:51:0a > da:61:cd:ab:52:a3, ethertype IPv4 (0x0800), length 98: 10.0.0.1 > 10.0.0.2: ICMP echo request, id 8744,
seq 1, length 64
10:19:06.338024 da:61:cd:ab:52:a3 > 92:05:85:f2:51:0a, ethertype IPv4 (0x0800), length 98: 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 8744, se
q 1, length 64
10:19:06.341679 92:05:85:f2:51:0a > b6:1e:3d:94:7f:48, ethertype IPv4 (0x0800), length 98: 10.0.0.1 > 10.0.0.3: ICMP echo request, id 8745,
seq 1, length 64
10:19:06.343159 b6:1e:3d:94:7f:48 > 92:05:85:f2:51:0a, ethertype IPv4 (0x0800), length 98: 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 8745, se
q 1, length 64
10:19:06.345198 da:61:cd:ab:52:a3 > 92:05:85:f2:51:0a, ethertype IPv4 (0x0800), length 98: 10.0.0.2 > 10.0.0.1: ICMP echo request, id 8746,
seq 1, length 64
10:19:06.345872 92:05:85:f2:51:0a > da:61:cd:ab:52:a3, ethertype IPv4 (0x0800), length 98: 10.0.0.1 > 10.0.0.2: ICMP echo reply, id 8746, se
q 1, length 64
10:19:06.349567 b6:1e:3d:94:7f:48 > 92:05:85:f2:51:0a, ethertype IPv4 (0x0800), length 98: 10.0.0.3 > 10.0.0.1: ICMP echo request, id 8748,
seq 1, length 64
10:19:06.350385 92:05:85:f2:51:0a > b6:1e:3d:94:7f:48, ethertype IPv4 (0x0800), length 98: 10.0.0.1 > 10.0.0.3: ICMP echo reply, id 8748, se
q 1, length 64
```

Figure 11: TCP Dump with pingall

It was observed that during the pingall command inside mininet CLI, the above echo request-responses were captured using tcp dump at s1-eth2 interface.

2. *Testing with "iperf"*

```
mininet> h1 iperf -s &
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
mininet> h3 iperf -c 10.0.0.1 -t 10
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.3 port 40648 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-24.1 sec   256 KBytes  87.0 Kbits/sec
mininet> h2 iperf -c 10.0.0.1 -t 10
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.2 port 35080 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-12.2 sec  1.25 MBytes   862 Kbits/sec
mininet>
```

Figure 12: Testing with iperf

Initially, configured h1 host as the iperf server. Then h2 and h3 was connected as iperf clients to the h1-iperf-server. The proctol has calculated the bottleneck-bandwidths and the delays of the links from h2-h1 and h3-h1.

3. *Testing with wireshark*
   Used the Wireshark – CLI (tshark)

   Case-1: ping from h1 to h2

```
mininet> h1 ping h2
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=2111 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=2026 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=2020 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=2020 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=2027 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=2022 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=2020 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=2022 ms
^C
--- 10.0.0.2 ping statistics ---
10 packets transmitted, 8 received, 20% packet loss, time 9025ms
rtt min/avg/max/mdev = 2020.566/2034.002/2111.392/29.379 ms, pipe 3
mininet>
```

Figure 13: ping from h1 to h2 while capturing from wireshark

```
vagrant@sdn-box:~$ sudo tshark -i s1-eth1
Running as user "root" and group "root". This could be dangerous.
tshark: Lua: Error during loading:
 /usr/share/wireshark/init.lua:32: dofile has been disabled due to running Wireshark as superuser. See https://wiki.wireshark.org/CaptureSet
up/CapturePrivileges for help in running Wireshark as an unprivileged user.
Capturing on 's1-eth1'
    1 0.000000000     10.0.0.1 → 10.0.0.2     ICMP 98 Echo (ping) request  id=0x34cf, seq=1/256, ttl=64
    2 1.004339570     10.0.0.1 → 10.0.0.2     ICMP 98 Echo (ping) request  id=0x34cf, seq=2/512, ttl=64
    3 2.014444822     10.0.0.1 → 10.0.0.2     ICMP 98 Echo (ping) request  id=0x34cf, seq=3/768, ttl=64
    4 2.106256608     10.0.0.2 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34cf, seq=1/256, ttl=64 (request in 1)
    5 3.015667521     10.0.0.1 → 10.0.0.2     ICMP 98 Echo (ping) request  id=0x34cf, seq=4/1024, ttl=64
    6 3.025712334     10.0.0.2 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34cf, seq=2/512, ttl=64 (request in 2)
    7 4.016982806     10.0.0.1 → 10.0.0.2     ICMP 98 Echo (ping) request  id=0x34cf, seq=5/1280, ttl=64
    8 4.030107085     10.0.0.2 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34cf, seq=3/768, ttl=64 (request in 3)
    9 5.017206661     10.0.0.1 → 10.0.0.2     ICMP 98 Echo (ping) request  id=0x34cf, seq=6/1536, ttl=64
   10 5.031136976     10.0.0.2 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34cf, seq=4/1024, ttl=64 (request in 5)
   11 6.019467712     10.0.0.1 → 10.0.0.2     ICMP 98 Echo (ping) request  id=0x34cf, seq=7/1792, ttl=64
   12 6.039096573     10.0.0.2 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34cf, seq=5/1280, ttl=64 (request in 7)
   13 7.023032623     10.0.0.1 → 10.0.0.2     ICMP 98 Echo (ping) request  id=0x34cf, seq=8/2048, ttl=64
   14 7.034240448     10.0.0.2 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34cf, seq=6/1536, ttl=64 (request in 9)
   15 7.166699777 ea:26:e7:8a:6a:91 → ee:d8:10:fa:4f:84 ARP 42 Who has 10.0.0.1? Tell 10.0.0.2
   16 7.171790298 ee:d8:10:fa:4f:84 → ea:26:e7:8a:6a:91 ARP 42 10.0.0.1 is at ee:d8:10:fa:4f:84
   17 8.024186396     10.0.0.1 → 10.0.0.2     ICMP 98 Echo (ping) request  id=0x34cf, seq=9/2304, ttl=64
   18 8.034862572     10.0.0.2 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34cf, seq=7/1792, ttl=64 (request in 11)
   19 8.080385407 ea:26:e7:8a:6a:91 → ee:d8:10:fa:4f:84 ARP 42 Who has 10.0.0.1? Tell 10.0.0.2
   20 8.085521414 ee:d8:10:fa:4f:84 → ea:26:e7:8a:6a:91 ARP 42 10.0.0.1 is at ee:d8:10:fa:4f:84
   21 9.025329904     10.0.0.1 → 10.0.0.2     ICMP 98 Echo (ping) request  id=0x34cf, seq=10/2560, ttl=64
   22 9.040420990     10.0.0.2 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34cf, seq=8/2048, ttl=64 (request in 13)
   23 9.080329744 ea:26:e7:8a:6a:91 → ee:d8:10:fa:4f:84 ARP 42 Who has 10.0.0.1? Tell 10.0.0.2
   24 9.085452852 ee:d8:10:fa:4f:84 → ea:26:e7:8a:6a:91 ARP 42 10.0.0.1 is at ee:d8:10:fa:4f:84
   25 10.040557691     10.0.0.2 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34cf, seq=9/2304, ttl=64 (request in 17)
   26 11.041981874     10.0.0.2 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34cf, seq=10/2560, ttl=64 (request in 21)
```

Figure 14: wireshark terminal output captured at s1-eth1 interface while pinging from h1 to h2

The echo request and response cycles and ARP messages can be clearly seen.

<u>Case-2: ping from h1 to h3</u>

```
mininet> h1 ping h3
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=4089 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=4014 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=4019 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=4016 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=4014 ms
^C
--- 10.0.0.3 ping statistics ---
9 packets transmitted, 5 received, 44% packet loss, time 8045ms
rtt min/avg/max/mdev = 4014.505/4030.843/4089.191/29.243 ms, pipe 5
mininet>
```

Figure 15: ping from h1 to h3 while capturing from wireshark

```
vagrant@sdn-box:~$ sudo tshark -i s2-eth2
Running as user "root" and group "root". This could be dangerous.
tshark: Lua: Error during loading:
 /usr/share/wireshark/init.lua:32: dofile has been disabled due to running Wireshark as superuser. See https://wiki.wireshark.org/CaptureSet
up/CapturePrivileges for help in running Wireshark as an unprivileged user.
Capturing on 's2-eth2'
    1 0.000000000     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x34e8, seq=1/256, ttl=64
    2 0.972004960     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x34e8, seq=2/512, ttl=64
    3 1.982592397     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x34e8, seq=3/768, ttl=64
    4 2.988635188     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x34e8, seq=4/1024, ttl=64
    5 3.020376567     10.0.0.3 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34e8, seq=1/256, ttl=64 (request in 1)
    6 3.976182693     10.0.0.3 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34e8, seq=2/512, ttl=64 (request in 2)
    7 3.995781145     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x34e8, seq=5/1280, ttl=64
    8 4.968770330 ee:d8:10:fa:4f:84 → 62:c8:e2:78:ce:76 ARP 42 Who has 10.0.0.3? Tell 10.0.0.1
    9 4.985798865     10.0.0.3 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34e8, seq=3/768, ttl=64 (request in 3)
   10 4.997843350     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x34e8, seq=6/1536, ttl=64
   11 5.965904962 ee:d8:10:fa:4f:84 → 62:c8:e2:78:ce:76 ARP 42 Who has 10.0.0.3? Tell 10.0.0.1
   12 5.993654427     10.0.0.3 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34e8, seq=4/1024, ttl=64 (request in 4)
   13 6.000103259     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x34e8, seq=7/1792, ttl=64
   14 6.966500243 ee:d8:10:fa:4f:84 → 62:c8:e2:78:ce:76 ARP 42 Who has 10.0.0.3? Tell 10.0.0.1
   15 6.999915987     10.0.0.3 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34e8, seq=5/1280, ttl=64 (request in 7)
   16 7.005360956     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x34e8, seq=8/2048, ttl=64
   17 7.999143269 62:c8:e2:78:ce:76 → ee:d8:10:fa:4f:84 ARP 42 10.0.0.3 is at 62:c8:e2:78:ce:76
   18 8.001028115     10.0.0.3 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34e8, seq=6/1536, ttl=64 (request in 10)
   19 8.015469764 ee:d8:10:fa:4f:84 → Broadcast    ARP 42 Who has 10.0.0.3? Tell 10.0.0.1
   20 8.968966256 62:c8:e2:78:ce:76 → ee:d8:10:fa:4f:84 ARP 42 10.0.0.3 is at 62:c8:e2:78:ce:76
   21 9.003533160     10.0.0.3 → 10.0.0.1     ICMP 98 Echo (ping) reply    id=0x34e8, seq=7/1792, ttl=64 (request in 13)
   22 9.041719450 ee:d8:10:fa:4f:84 → Broadcast    ARP 42 Who has 10.0.0.3? Tell 10.0.0.1
   23 9.050995118     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x34e8, seq=9/2304, ttl=64
```

Figure 16: wireshark terminal output at s2-eth2 while pinging from h1 to h3

Case-2: ping from h1 to h3 after configuring a drop action OpenFlow rule

```
mininet> sh ovs-ofctl add-flow s2 priority=65535,dl_type=0x800,nw_src=10.0.0.1,nw_dst=10.0.0.3,actions=drop
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
^C
--- 10.0.0.3 ping statistics ---
44 packets transmitted, 0 received, 100% packet loss, time 43142ms

mininet>
```

Figure 17: ping from h1 to h3 with drop action rule

```
vagrant@sdn-box:~$ sudo tshark -i s2-eth2
Running as user "root" and group "root". This could be dangerous.
tshark: Lua: Error during loading:
 /usr/share/wireshark/init.lua:32: dofile has been disabled due to running Wireshark as superuser. See https://wiki.wireshark.org/CaptureSet
up/CapturePrivileges for help in running Wireshark as an unprivileged user.
Capturing on 's2-eth2'
    1 0.000000000     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x3502, seq=1/256, ttl=64
    2 0.984058117     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x3502, seq=2/512, ttl=64
    3 1.996086751     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x3502, seq=3/768, ttl=64
    4 3.000055086     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x3502, seq=4/1024, ttl=64
    5 4.001212993     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x3502, seq=5/1280, ttl=64
    6 4.990114147 ee:d8:10:fa:4f:84 → 62:c8:e2:78:ce:76 ARP 42 Who has 10.0.0.3? Tell 10.0.0.1
    7 5.012459507     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x3502, seq=6/1536, ttl=64
    8 5.985847208 ee:d8:10:fa:4f:84 → 62:c8:e2:78:ce:76 ARP 42 Who has 10.0.0.3? Tell 10.0.0.1
    9 6.018753878     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x3502, seq=7/1792, ttl=64
   10 6.986474034 ee:d8:10:fa:4f:84 → 62:c8:e2:78:ce:76 ARP 42 Who has 10.0.0.3? Tell 10.0.0.1
   11 7.017562300     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x3502, seq=8/2048, ttl=64
   12 8.021948160 ee:d8:10:fa:4f:84 → Broadcast    ARP 42 Who has 10.0.0.3? Tell 10.0.0.1
   13 8.026202387 62:c8:e2:78:ce:76 → ee:d8:10:fa:4f:84 ARP 42 10.0.0.3 is at 62:c8:e2:78:ce:76
   14 8.988885325 62:c8:e2:78:ce:76 → ee:d8:10:fa:4f:84 ARP 42 10.0.0.3 is at 62:c8:e2:78:ce:76
   15 9.049146770 ee:d8:10:fa:4f:84 → Broadcast    ARP 42 Who has 10.0.0.3? Tell 10.0.0.1
   16 9.060142467     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x3502, seq=9/2304, ttl=64
   17 9.060144167     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x3502, seq=10/2560, ttl=64
   18 9.988836878 62:c8:e2:78:ce:76 → ee:d8:10:fa:4f:84 ARP 42 10.0.0.3 is at 62:c8:e2:78:ce:76
   19 10.017721669     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x3502, seq=11/2816, ttl=64
   20 11.017528169     10.0.0.1 → 10.0.0.3     ICMP 98 Echo (ping) request  id=0x3502, seq=12/3072, ttl=64
   21 11.055152191 62:c8:e2:78:ce:76 → ee:d8:10:fa:4f:84 ARP 42 10.0.0.3 is at 62:c8:e2:78:ce:76
^C21 packets captured
vagrant@sdn-box:~$
```

Figure 18: Wireshark terminal output while capturing at s2-eth2 with the openflow rule

Since the rule defined to drop packets arriving from h1 to h3 at s2 switch; the s2-eth2 should be able to capture echo request icmp packets. However, since the openflow rule s2 drops these icmp packet, which avoids h3 from sending echo reply packets to h1. ∴ Echo- response icmp packets are not visible in this output (Figure-18). Therefore, the wireshark terminal has captured the packets exactly as expected.

Cleaning-up the Mininet Network Setup

Closing the Pox SDN controller

```
INFO:openflow.of_01:[00-00-00-00-00-03 1] closed
INFO:openflow.of_01:[00-00-00-00-00-02 2] closed
INFO:openflow.of_01:[00-00-00-00-00-01 3] closed
^CINFO:core:Going down...
INFO:core:Down.
vagrant@sdn-box:~/pox$
```

Cleaning the mininet network

```
mininet> exit
*** Stopping network*** Stopping 1 controllers
c0
*** Stopping 5 links
.....
*** Stopping 3 switches
s1 s2 s3
*** Stopping 3 hosts
h1 h2 h3
*** Done
vagrant@sdn-box:~/labtasks$ sudo mn -c
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
vagrant@sdn-box:~/labtasks$
```