

## Final Project Lab Report

First, let's start with TCP/ipv4. As you can see, h1, h2, h3, and h5 can communicate just fine due to the rules installed in the flow table.

```
mininet@mininet-vm:~/lab/CE150/lab4$ sudo python final_skel.py
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['19.9 Gbits/sec', '19.9 Gbits/sec']
mininet> iperf h5 h3
*** Iperf: testing TCP bandwidth between h5 and h3
*** Results: ['30.9 Gbits/sec', '30.9 Gbits/sec']
mininet> iperf h4 h3
*** Iperf: testing TCP bandwidth between h4 and h3
```

However, what happens when h4, an outsider, tries to connect? The connection completely freezes up, and you have to tell it to give up. Why? I designed the switches 'learn' that h4 is not a part of trusted hosted by ofpmod, as seen by dpctl dump-flows here: (Notice the actions=drop)

```
mininet> iperf h3 h4
*** Iperf: testing TCP bandwidth between h3 and h4
^C
Interrupt
mininet> dpctl dump-flows
*** s1 -----
NXST_FLOW reply (xid=0x4):
*** s2 -----
NXST_FLOW reply (xid=0x4):
*** s3 -----
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=103.615s, table=0, n_packets=0, n_bytes=0, idle_age=103, priority=3,tcp,nw_src=123.45.67.89,nw_dst=10.3.3.30 actions=drop
  cookie=0x0, duration=103.614s, table=0, n_packets=6, n_bytes=444, idle_age=1, priority=3,tcp,nw_src=10.3.3.30,nw_dst=123.45.67.89 actions=drop
*** s4 -----
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=103.656s, table=0, n_packets=6, n_bytes=444, idle_age=72, priority=3,tcp,nw_src=123.45.67.89,nw_dst=10.3.3.30 actions=drop
  cookie=0x0, duration=103.619s, table=0, n_packets=0, n_bytes=0, idle_age=103, priority=3,tcp,nw_src=10.3.3.30,nw_dst=123.45.67.89 actions=drop
*** s5 -----
NXST_FLOW reply (xid=0x4):
```

Similar behavior for UDP, another ipv4 protocol. h3 -> h5 udp connection by xterm is successful below.

```

"Node: h3"
root@mininet-vm:~/lab/CE150/lab4# iperf -u -c 10.5.5.50
Client connecting to 10.5.5.50, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 160 KByte (default)

[ 27] local 10.3.3.30 port 45416 connected with 10.5.5.50 port 5001
[ 10] Interval      Transfer      Bandwidth
[ 27] 0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 27] Sent 893 datagrams
[ 27] Server Report:
[ 27] 0.0- 9.9 sec  1.24 MBytes  1.05 Mbits/sec   0.013 ms  11/ 893 (1.2%)
root@mininet-vm:~/lab/CE150/lab4#

"Node: h5"
root@mininet-vm:~/lab/CE150/lab4# iperf -u -s
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 160 KByte (default)

[ 27] local 10.5.5.50 port 5001 connected with 10.3.3.30 port 45416
[ 10] Interval      Transfer      Bandwidth      Jitter    Lost/Total Datagrams
[ 27] 0.0- 0.0 sec  0.00 Bytes  0.00 bits/sec   0.000 ms   0/ 0 (-nan%)
[ 28] local 10.5.5.50 port 5001 connected with 10.3.3.30 port 45416
[ 28] 0.0- 0.0 sec  2.87 KBytes  1.16 Mbits/sec   0.333 ms   7/ 10 (70%)
[ 28] 0.0- 0.0 sec  1 datagrams received out-of-order
[ 27] local 10.5.5.50 port 5001 connected with 10.3.3.30 port 45416
[ 27] 0.0- 9.9 sec  1.24 MBytes  1.05 Mbits/sec   0.013 ms  11/ 893 (1.2%)
```

Now let's try UDP on h4 -> h5. H5 Refuses to acknowledge h4 at all, like with TCP!

```
"Node: h4"
mininet-vn:/lab/CE150/lab4# iperf -u -c 10.5.5.50
-----
Not connecting to 10.5.5.50, UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 160 KByte (default)
-----
[  0] local 123.45.67.89 port 53716 connected with 10.5.5.50 port 5001
[  0] Interval      Transfer    Bandwidth
[  0] 0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  0] Sent 893 datagrams
[  0] WARNING: did not receive ack of last datagram after 10 tries.
mininet-vn:/lab/CE150/lab4#

"Node: h5"
root@mininet-vn:/lab/CE150/lab4# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 160 KByte (default)
-----
[]
```

```
"Node: h5"
root@mininet-vn:/lab/CE150/lab4# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 160 KByte (default)
-----
[]
```

Now the attacker tries to use pingall. What happens?

```
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> X X X X
h5 -> h1 h2 h3 X
```

As you can see, h4 can still attempt to send out ARP packets. I have all non-ipv4 packets to of.OFPF\_FLOOD. h1 is still pinging h4! However, moment that h4 even attempts to try ICMP packets through ping, it is blocked out from the network. Check out all the actions= 'drop' rules installed.

```

*** s1 -----
NXST FLOW reply (xid=0x4):
cookie=0x0, duration=55.384s, table=0, n_packets=0, n_bytes=0, idle_age=55, priority=3,icmp,nw_src=10.2.2.20,nw_dst=123.45.67.89 actions=drop
cookie=0x0, duration=45.412s, table=0, n_packets=0, n_bytes=0, idle_age=45, priority=3,icmp,nw_src=10.3.3.30,nw_dst=123.45.67.89 actions=drop
cookie=0x0, duration=65.36s, table=0, n_packets=1, n_bytes=98, idle_age=65, priority=3,icmp,nw_src=10.1.1.10,nw_dst=123.45.67.89 actions=drop
cookie=0x0, duration=35.322s, table=0, n_packets=0, n_bytes=0, idle_age=35, priority=3,icmp,nw_src=10.5.5.50,nw_dst=123.45.67.89 actions=drop
*** s2 -----
NXST FLOW reply (xid=0x4):
cookie=0x0, duration=55.385s, table=0, n_packets=1, n_bytes=98, idle_age=55, priority=3,icmp,nw_src=10.2.2.20,nw_dst=123.45.67.89 actions=drop
cookie=0x0, duration=45.413s, table=0, n_packets=0, n_bytes=0, idle_age=45, priority=3,icmp,nw_src=10.3.3.30,nw_dst=123.45.67.89 actions=drop
cookie=0x0, duration=65.361s, table=0, n_packets=0, n_bytes=0, idle_age=65, priority=3,icmp,nw_src=10.1.1.10,nw_dst=123.45.67.89 actions=drop
cookie=0x0, duration=35.323s, table=0, n_packets=0, n_bytes=0, idle_age=35, priority=3,icmp,nw_src=10.5.5.50,nw_dst=123.45.67.89 actions=drop
*** s3 -----
NXST FLOW reply (xid=0x4):
cookie=0x0, duration=55.39s, table=0, n_packets=0, n_bytes=0, idle_age=55, priority=3,icmp,nw_src=10.2.2.20,nw_dst=123.45.67.89 actions=drop
cookie=0x0, duration=45.418s, table=0, n_packets=1, n_bytes=98, idle_age=45, priority=3,icmp,nw_src=10.3.3.30,nw_dst=123.45.67.89 actions=drop
cookie=0x0, duration=65.366s, table=0, n_packets=0, n_bytes=0, idle_age=65, priority=3,icmp,nw_src=10.1.1.10,nw_dst=123.45.67.89 actions=drop
cookie=0x0, duration=35.328s, table=0, n_packets=0, n_bytes=0, idle_age=35, priority=3,icmp,nw_src=10.5.5.50,nw_dst=123.45.67.89 actions=drop
*** s4 -----
NXST FLOW reply (xid=0x4):
cookie=0x0, duration=55.397s, table=0, n_packets=0, n_bytes=0, idle_age=55, priority=3,icmp,nw_src=10.2.2.20,nw_dst=123.45.67.89 actions=drop
cookie=0x0, duration=45.425s, table=0, n_packets=0, n_bytes=0, idle_age=45, priority=3,icmp,nw_src=10.3.3.30,nw_dst=123.45.67.89 actions=drop
cookie=0x0, duration=65.44s, table=0, n_packets=0, n_bytes=0, idle_age=65, priority=3,icmp,nw_src=10.1.1.10,nw_dst=123.45.67.89 actions=drop
cookie=0x0, duration=35.4s, table=0, n_packets=0, n_bytes=0, idle_age=35, priority=3,icmp,nw_src=10.5.5.50,nw_dst=123.45.67.89 actions=drop
*** s5 -----
NXST FLOW reply (xid=0x4):
cookie=0x0, duration=55.398s, table=0, n_packets=0, n_bytes=0, idle_age=55, priority=3,icmp,nw_src=10.2.2.20,nw_dst=123.45.67.89 actions=drop
cookie=0x0, duration=45.425s, table=0, n_packets=0, n_bytes=0, idle_age=45, priority=3,icmp,nw_src=10.3.3.30,nw_dst=123.45.67.89 actions=drop
cookie=0x0, duration=65.374s, table=0, n_packets=0, n_bytes=0, idle_age=65, priority=3,icmp,nw_src=10.1.1.10,nw_dst=123.45.67.89 actions=drop
cookie=0x0, duration=35.336s, table=0, n_packets=2, n_bytes=196, idle_age=25, priority=3,icmp,nw_src=10.5.5.50,nw_dst=123.45.67.89 actions=drop
mininet>

```

Overall, I learned a lot about how packet handling works in openflow. You absolutely have to broadcast rules to the appropriate switches when forwarding ports. If I were to expand upon this assignment, I would try to see how to handle a DDoS attack or Trojan attack, as ARP leaves a hole open in our network.