Christy Jose

Final lab report

<u>Mininet Topology:</u>

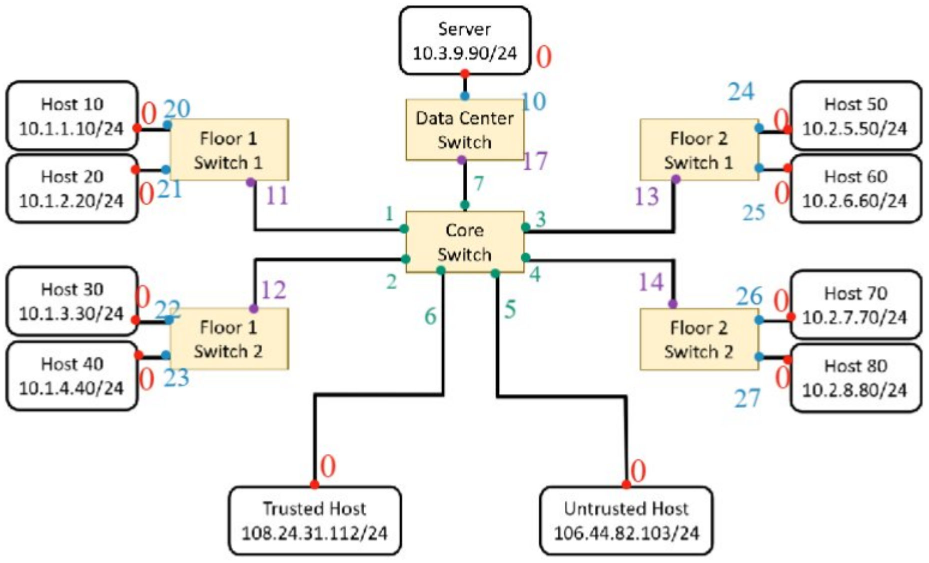


Figure 1.1

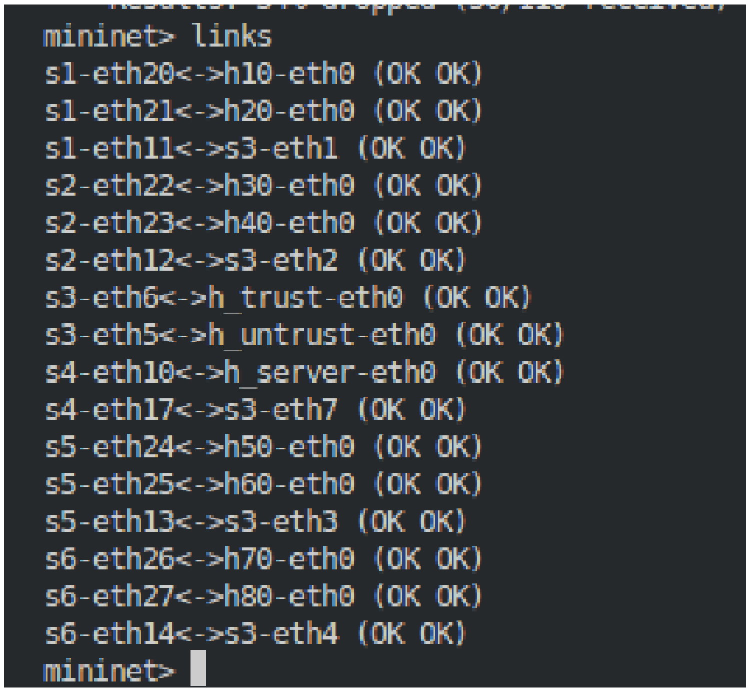
The image above (Figure 1.1) shows the topology for this lab. It includes the port numbers for each switch. Here are the abbreviations that I used in my code:

- Floor 1 switch 1 = s1
- Floor 1 switch 2 = s2
- Core switch = s3
- Data Center switch = s4
- Floor 2 switch 1 = s5
- Floor 2 switch 2 = s6
- Host 10 - Host 80 = h10 - h80
- Trusted Host = h_trust
- Untrusted Host = h_untrust
- Server = h_server

In order to find the links in the topology, I used the ***links*** command on the mininet terminal. The image below (Figure 1.2) shows the links made in this topology. Figure 1.2 shows the links in the following format:

(switch name)-eth(switch's port number)<->(host name)-eth(host's port number)

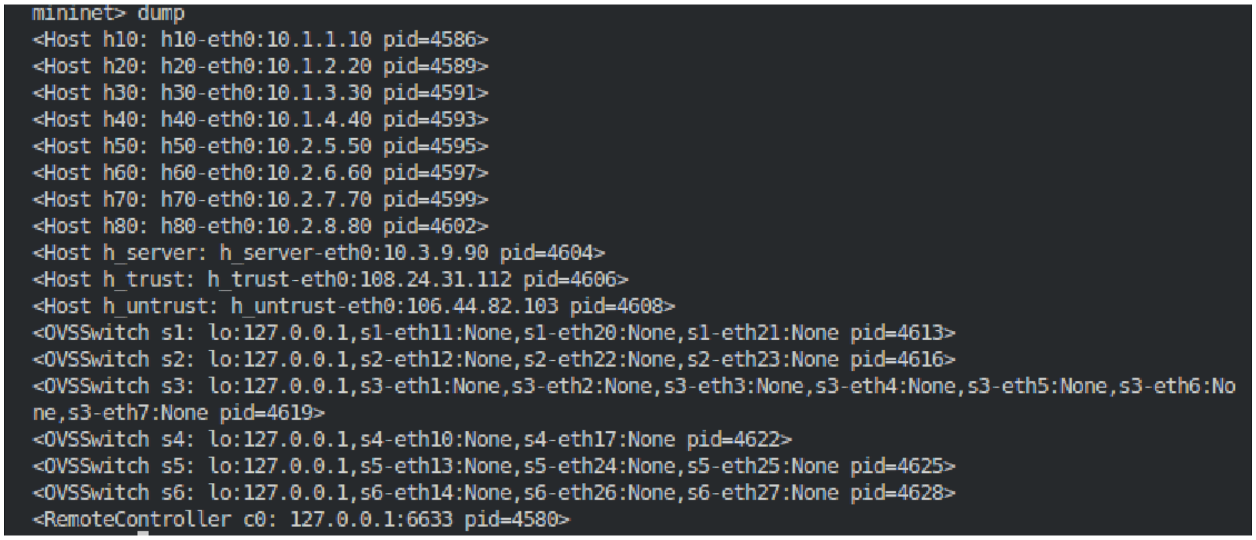
Lets take an example from the first line from Figure 1.2. The switch s1 is connected to host h10 from s1's port 20 to h10's port 0. This can be confirmed by looking at the topology diagram. The same can be inferred for the rest of the links. There are 16 links in Figure 1.2. This is because there are 16 connections in the topology diagram from Figure 1.1. All of the links from Figure 1.2 are consistent with the diagram from Figure 1.1



```
mininet> links
s1-eth20<->h10-eth0 (OK OK)
s1-eth21<->h20-eth0 (OK OK)
s1-eth11<->s3-eth1 (OK OK)
s2-eth22<->h30-eth0 (OK OK)
s2-eth23<->h40-eth0 (OK OK)
s2-eth12<->s3-eth2 (OK OK)
s3-eth6<->h_trust-eth0 (OK OK)
s3-eth5<->h_untrust-eth0 (OK OK)
s4-eth10<->h_server-eth0 (OK OK)
s4-eth17<->s3-eth7 (OK OK)
s5-eth24<->h50-eth0 (OK OK)
s5-eth25<->h60-eth0 (OK OK)
s5-eth13<->s3-eth3 (OK OK)
s6-eth26<->h70-eth0 (OK OK)
s6-eth27<->h80-eth0 (OK OK)
s6-eth14<->s3-eth4 (OK OK)
mininet>
```

Figure 1.2

In order to find the ip addresses of the hosts, I used the ***dump*** command on the mininet terminal. In Figure 1.3 below are all of the ip addresses of the hosts.



```
mininet> dump
<Host h10: h10-eth0:10.1.1.10 pid=4586>
<Host h20: h20-eth0:10.1.2.20 pid=4589>
<Host h30: h30-eth0:10.1.3.30 pid=4591>
<Host h40: h40-eth0:10.1.4.40 pid=4593>
<Host h50: h50-eth0:10.2.5.50 pid=4595>
<Host h60: h60-eth0:10.2.6.60 pid=4597>
<Host h70: h70-eth0:10.2.7.70 pid=4599>
<Host h80: h80-eth0:10.2.8.80 pid=4602>
<Host h_server: h_server-eth0:10.3.9.90 pid=4604>
<Host h_trust: h_trust-eth0:108.24.31.112 pid=4606>
<Host h_untrust: h_untrust-eth0:106.44.82.103 pid=4608>
<OVSSwitch s1: lo:127.0.0.1,s1-eth11:None,s1-eth20:None,s1-eth21:None pid=4613>
<OVSSwitch s2: lo:127.0.0.1,s2-eth12:None,s2-eth22:None,s2-eth23:None pid=4616>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None,s3-eth4:None,s3-eth5:None,s3-eth6:No
ne,s3-eth7:None pid=4619>
<OVSSwitch s4: lo:127.0.0.1,s4-eth10:None,s4-eth17:None pid=4622>
<OVSSwitch s5: lo:127.0.0.1,s5-eth13:None,s5-eth24:None,s5-eth25:None pid=4625>
<OVSSwitch s6: lo:127.0.0.1,s6-eth14:None,s6-eth26:None,s6-eth27:None pid=4628>
<RemoteController c0: 127.0.0.1:6633 pid=4580>
```

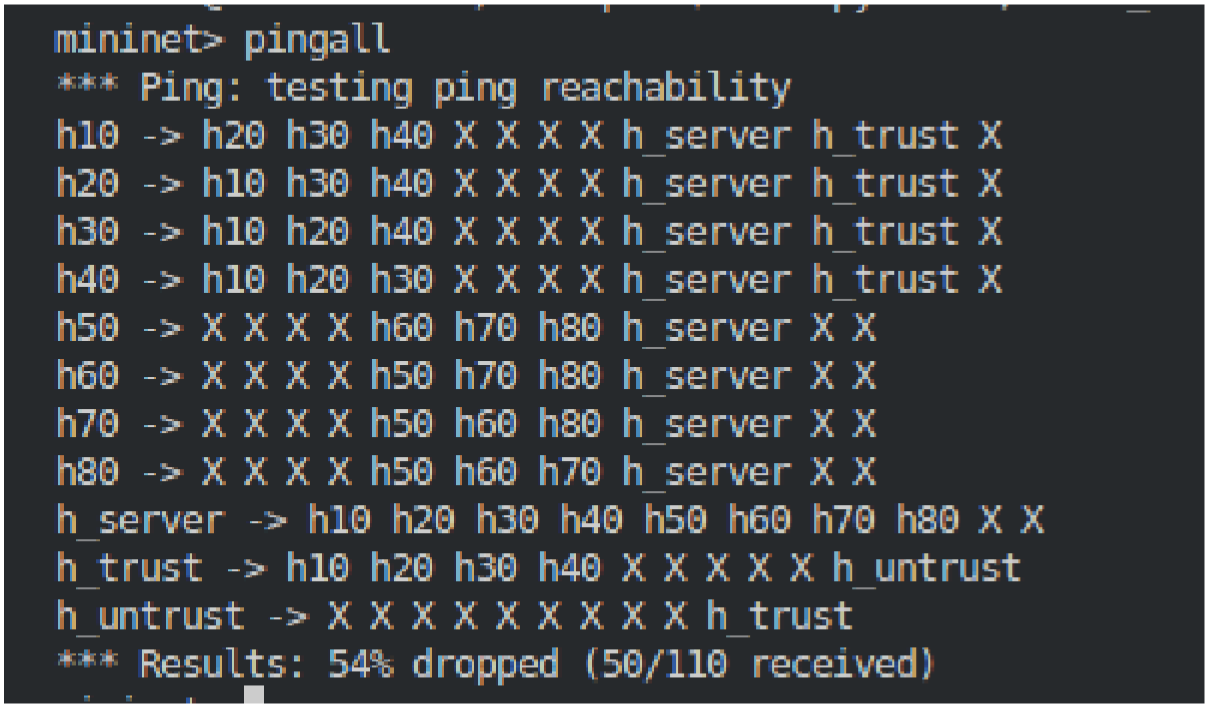
Figure 1.3

The figure shows the information in the following format:

<Host (host name): (host name)-eth0:(host’s ip address) pid=(pid number)>

The host's ip address is consistent with the requirements. For example, Host 10's ip address is 10.1.1.10, and so on.

Pox Controller:



```
mininet> pingall
*** Ping: testing ping reachability
h10 -> h20 h30 h40 X X X X h_server h_trust X
h20 -> h10 h30 h40 X X X X h_server h_trust X
h30 -> h10 h20 h40 X X X X h_server h_trust X
h40 -> h10 h20 h30 X X X X h_server h_trust X
h50 -> X X X X h60 h70 h80 h_server X X
h60 -> X X X X h50 h70 h80 h_server X X
h70 -> X X X X h50 h60 h80 h_server X X
h80 -> X X X X h50 h60 h70 h_server X X
h_server -> h10 h20 h30 h40 h50 h60 h70 h80 X X
h_trust -> h10 h20 h30 h40 X X X X X h_untrust
h_untrust -> X X X X X X X X X h_trust
*** Results: 54% dropped (50/110 received)
```

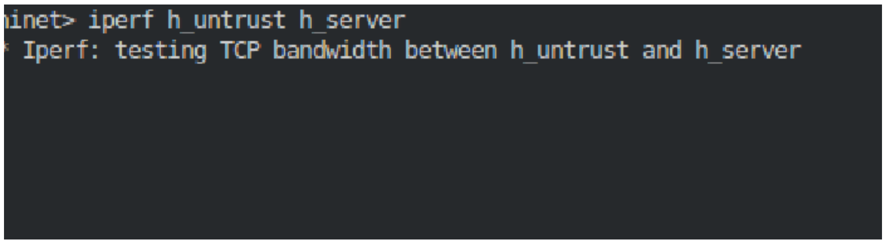
Figure 2.1

The pingall command on mininet sends an icmp packet from every host to every host. This shows how each host communicates with the other host given the icmp packet. This also shows the implementation of the rules in the firewall of each host. Here were the following rules:

1. Untrusted host cannot send icmp traffic to Host 10 to 80, or the server:

If you look at the h_untrust row in Figure 2.1, you will notice that there are nine Xs in the front. The first eight Xs represents hosts 10 to 80, and the ninth X represents the server host. An X indicates that the icmp packet was dropped, which means that the icmp packet was not received on the other end. However, you will notice that Untrusted host can send icmp traffic to Trusted host.

2. Untrusted host cannot send any ip traffic to the server.

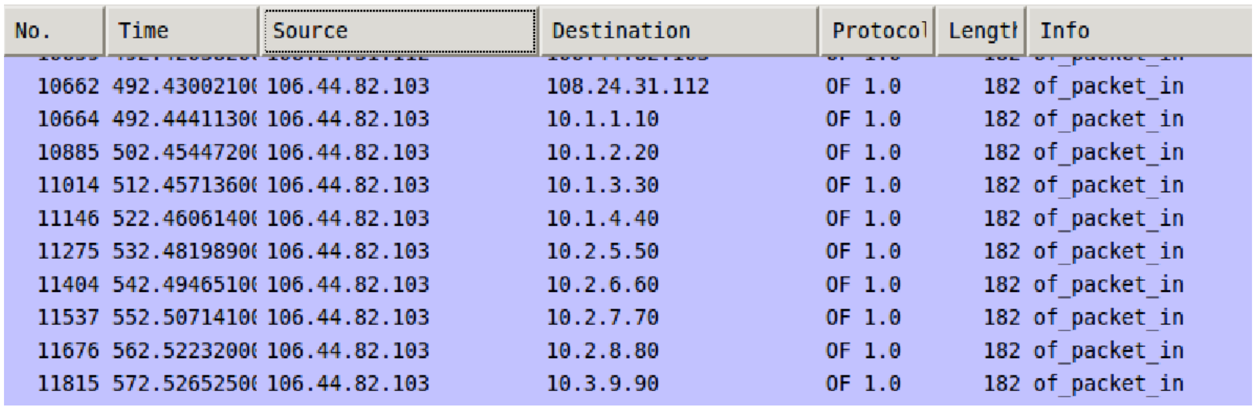


```
inet> iperf h_untrust h_server
 Iperf: testing TCP bandwidth between h_untrust and h_server
```

Figure 2.2

The iperf command is used to test the TCP bandwidth between two hosts. This command was used between the untrusted host and the server host. A bandwidth can only be calculated if an IP packet has been received. This includes icmp and non icmp ip traffic. Since the packet had not been received in a long time, the computer is left waiting. This means that no ip traffic has been received by the server host from the untrusted host.

2.1. The Untrusted host can send non icmp traffic to hosts.



No.	Time	Source	Destination	Protocol	Length	Info
10662	492.4300210(	106.44.82.103	108.24.31.112	OF 1.0	182	of_packet_in
10664	492.4441130(	106.44.82.103	10.1.1.10	OF 1.0	182	of_packet_in
10885	502.4544720(	106.44.82.103	10.1.2.20	OF 1.0	182	of_packet_in
11014	512.4571360(	106.44.82.103	10.1.3.30	OF 1.0	182	of_packet_in
11146	522.4606140(	106.44.82.103	10.1.4.40	OF 1.0	182	of_packet_in
11275	532.4819890(	106.44.82.103	10.2.5.50	OF 1.0	182	of_packet_in
11404	542.4946510(	106.44.82.103	10.2.6.60	OF 1.0	182	of_packet_in
11537	552.5071410(	106.44.82.103	10.2.7.70	OF 1.0	182	of_packet_in
11676	562.5223200(	106.44.82.103	10.2.8.80	OF 1.0	182	of_packet_in
11815	572.5265250(	106.44.82.103	10.3.9.90	OF 1.0	182	of_packet_in

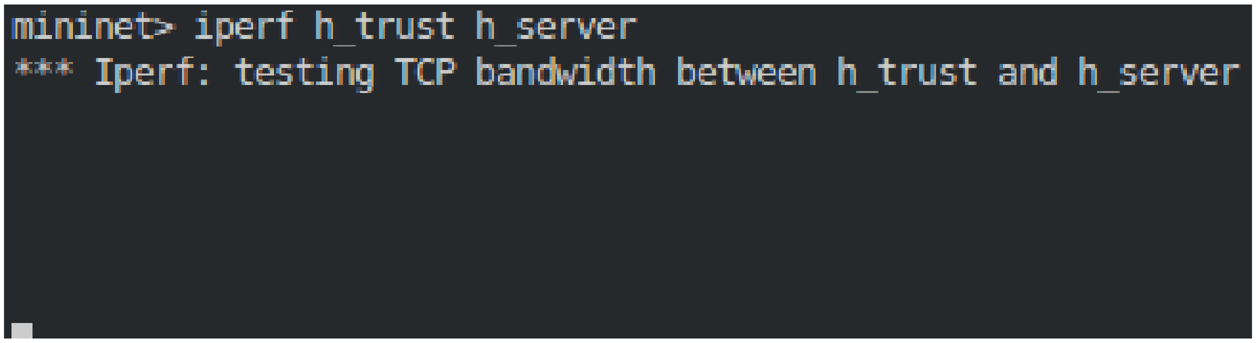
Figure 2.2.1

This figure shows the Untrusted host (106.44.82.103) sending Openflow (of 1.0) packets, which are not icmp, to hosts 10 to 80, and server, and trusted host.

3. Trusted host cannot send icmp traffic to Host 50 to 80, or the server.

If you look at the h_trust row in Figure 2.1, you will notice hosts 10 to 40, and has five Xs right after. The first four Xs are hosts 50 to 80, and the fifth X is the server. I was able to tell this because the hosts in each row are ordered in the same way it is from going top to bottom, excluding itself. An X indicates that the icmp packet was dropped, which means that the icmp packet was not received by hosts 50 to 80 and the server host. If there is a host's name instead of an X, then it means that the icmp packet was received, which means that hosts 10 to 40 can receive icmp traffic.

4. Trusted host cannot send any ip traffic to the server



```
mininet> iperf h_trust h_server
*** Iperf: testing TCP bandwidth between h_trust and h_server
```

Figure 2.3

The iperf command is used to test the TCP bandwidth between two hosts. This command was used between the trusted host and the server host. As mentioned previously, a bandwidth can only be calculated if an IP packet has been received. This includes icmp and non icmp ip traffic. Since the packet had not been received in a long time, the computer is left waiting. This means that no ip traffic has been received by the server host from the untrusted host.

5. Hosts 10 to 40 cannot send any icmp traffic to Host 50 to 80 and vice versa.

If you look at the host 10 to 40 rows in Figure 2.1, you will notice that there are a block of hosts 10 to 40 followed by a block of four Xs. This block of four Xs represents hosts 50 to 80. An X indicates that the icmp packet was dropped, which means that the icmp packet was not received by hosts 50 to 80, and was received by hosts 10 to 40.

If you look at the host 50 to 80 rows, you will notice a block of four Xs in the beginning followed by the hosts 50 to 80. This block of four Xs represents hosts 10 to 40, which means that these hosts cannot receive icmp traffic from hosts 50 to 80. The following block of hosts 50 to 80 show that these hosts can receive icmp traffic.

<u>How the project was implemented:</u>

Topology:
I refered to the final project skeleton code to add my hosts with the mac addresses. I made sure to match the default route of the host to the hostname and port number.

Controller:
I used my drop function and my flood function from my previous lab. I modified the flood function to take in a port number, and used that number as the port number instead of using of.OFPP_FLOOD.

Before i checked if the packets were ip or not, I accepted the arp packet so that it would remember the MAC address and the associated ip address. To do this, I used the old flood function with the of.OFPP_FLOOD.

Then I checked if the packet was ip. Once I did that, I checked if it was an icmp packet. Then I went through each switch, checked the destination ip address and forwarded the packed using the flood function to the respective port in the switch. I did this for all of the switches except for the core. I implemented the firewall in the core switch. I used the port_on_switch to determine where the packet was coming from. I checked the destination ip address and decided whether or not to flood it to the respective port on the core or to drop it.

I followed the same process under if it was an ip packet but not icmp, with their respective rules.