

CSE 150 Final Project Report

Introduction:

The purpose of this lab is to construct a network for a small company with a 2-floor building. The network consists of multiple switches, subnets, and servers, interconnected by a core switch. The goal is to implement various requirements to ensure network security and proper communication between different devices.

The network setup includes two floors, each with its own switches and subnets. Floor 1 hosts computers from Department A, while Floor 2 hosts computers from Department B. The IP addresses assigned to the devices on Floor 1 are 10.1.1.10/24, 10.1.2.20/24, 10.1.3.30/24, and 10.1.4.40/24, while the devices on Floor 2 have IP addresses of 10.2.5.50/24, 10.2.6.60/24, 10.2.7.70/24, and 10.2.8.80/24.

Additionally, there are two external hosts: a trusted host (h_trust) with the IP address 108.24.31.112/24, owned by a certified employee from Department A, and an untrusted host (h_untrust) with the IP address 106.44.82.103/24, treated as a potential hacker.

There is also a server (h_server) with the IP address 10.3.9.90/24, used by internal or trusted hosts for various purposes.

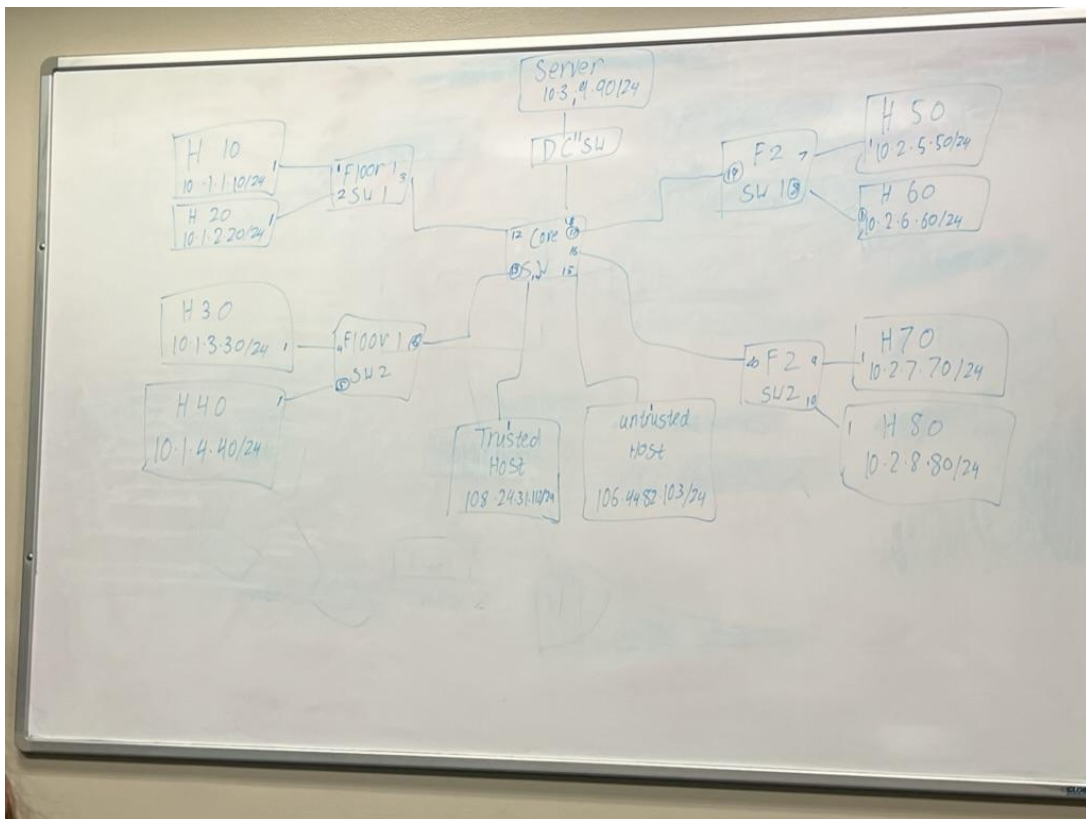
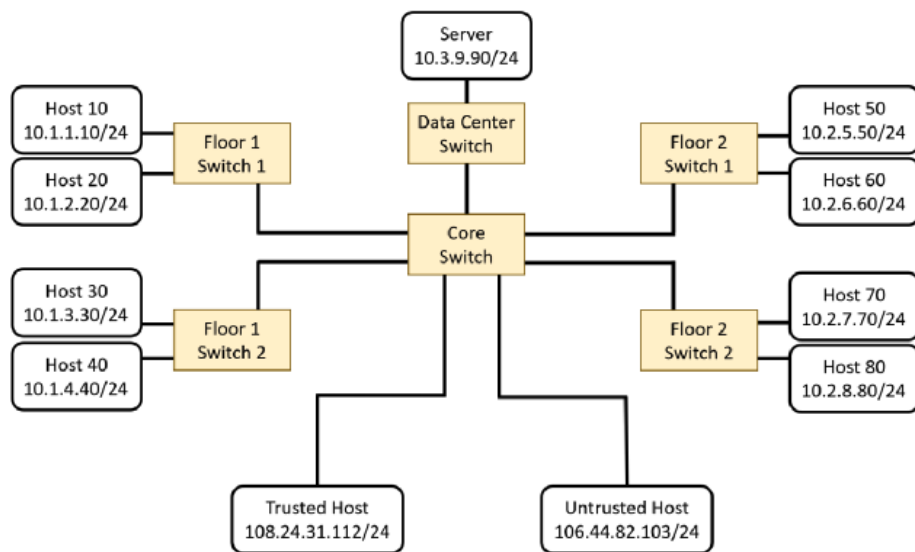
The main objective of this assignment is to allow or block traffic between different hosts and servers based on specific requirements. The non-IP traffic can be flooded using the same method as in Lab 3, while specific ports need to be specified for IP traffic. The destination port for IP

traffic can be determined based on the source and destination IP addresses and the source port on the originating switch.

The requirements for this assignment are as follows:

1. Block all IP traffic from the untrusted host (h_untrust) to the server (h_server) to protect the servers from the untrusted Internet.
2. Block all ICMP traffic from the untrusted host (h_untrust) to any internal destination, including hosts 10-80 and the server, to prevent the discovery of internal IP addresses.
3. Allow the trusted host (h_trust) to send any traffic to the hosts in Department A (hosts 10, 20, 30, 40). However, the trusted host should not send any ICMP or IP traffic to the server (h_server) and should not send ICMP traffic to the hosts in Department B (hosts 50, 60, 70, 80).
4. Block all ICMP traffic from hosts in Department A (hosts 10, 20, 30, 40) to hosts in Department B (hosts 50, 60, 70, 80), and vice versa.

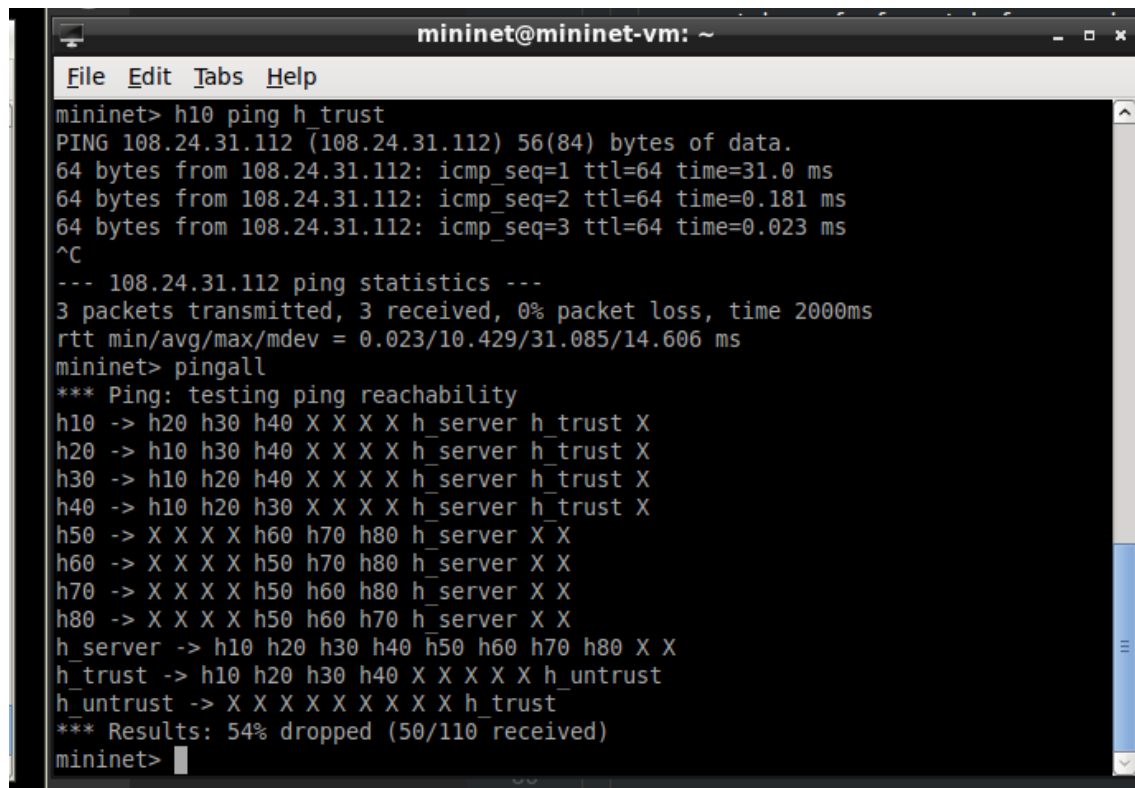
In this report, we will explain how these requirements were implemented and demonstrate their functionality using screenshots.



This is an image of my topology with the port numbers that I selected. Originally I had all the hosts lie on port 1, however, after viewing the skeleton topology script, I changed all the host ports to port 0. On top of that, you can see that the host port numbers are the only numbers that stay the

same, all the other ports for the switches are each a unique number in order to not enter a loop while traversing packets.

pingall Commands



```
mininet@mininet-vm: ~  
File Edit Tabs Help  
mininet> h10 ping h_trust  
PING 108.24.31.112 (108.24.31.112) 56(84) bytes of data.  
64 bytes from 108.24.31.112: icmp_seq=1 ttl=64 time=31.0 ms  
64 bytes from 108.24.31.112: icmp_seq=2 ttl=64 time=0.181 ms  
64 bytes from 108.24.31.112: icmp_seq=3 ttl=64 time=0.023 ms  
^C  
--- 108.24.31.112 ping statistics ---  
3 packets transmitted, 3 received, 0% packet loss, time 2000ms  
rtt min/avg/max/mdev = 0.023/10.429/31.085/14.606 ms  
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 h_untrust  
h_untrust -> X X X X X X X X h_trust  
*** Results: 54% dropped (50/110 received)  
mininet>
```

The first command I ran was a simple ‘pingall’ command. Running *pingall* sends a ICMP echo request to every host on the network in order to test for connectivity. Since some of our restrictions involved having to block ICMP traffic from various hosts, this command was a great way to show that.

From the screenshot, we can see that ICMP messages are blocked from Department A hosts to Department B hosts as well as from the Untrusted Host to anywhere internally and from the Trusted Host to Department B hosts and the Server. This showed that our Router was effectively blocking ICMP packets as per the project requirements.

ICMP is one of the main IP protocols and since we are only flooding non-IP packets, ICMP was not going to be automatically flooded. On top of that, our router was only configured to block ICMP packets from the Trusted Host to the Department B hosts and not vice versa but the reason it shows

that Department B hosts cannot ping the Trusted Host is because the Trusted Host fails to send a ICMP response message even though it may receive a request message.

Overall, the "pingall" results validate that the implemented network restrictions are functioning as intended. The restrictions successfully control the flow of traffic between different hosts, safeguarding the servers and preventing unauthorized access while maintaining the necessary communication channels between trusted hosts and internal departments.

Dpctl dump-flow Commands

```
mininet@mininet-vm: ~  
File Edit Tabs Help  
#trust -> h10 h20 h30 h40 X X X X h untrust  
h untrust -> X X X X X X X X h trust  
*** Results: 54% dropped (50/110 received)  
mininet> dpctl dump-flows  
*** s5 -----  
NXST_FLOW reply (xid=0x4):  
 cookie=0x0, duration=26.4575, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:09,d1_d  
st=00:00:00:00:10,arp_spa=108.24.31.112,arp_tpa=106.44.82.103,arp_op=2 actio  
ns=FLOOD  
 cookie=0x0, duration=26.4625, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:10,d1  
st=00:00:00:00:09,arp_spa=106.44.82.103,arp_tpa=108.24.31.112,arp_op=1 actio  
ns=FLOOD  
 cookie=0x0, duration=26.4615, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:09,d1  
st=00:00:00:00:10,arp_spa=108.24.31.112,arp_tpa=106.44.82.103,arp_op=1 actio  
ns=FLOOD  
 cookie=0x0, duration=26.4185, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:10,d1  
st=00:00:00:00:09,arp_spa=106.44.82.103,arp_tpa=108.24.31.112,arp_op=2 actio  
ns=FLOOD  
*** s6 -----  
NXST_FLOW reply (xid=0x4):  
 cookie=0x0, duration=26.4545, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:09,d1  
st=00:00:00:00:10,arp_spa=108.24.31.112,arp_tpa=106.44.82.103,arp_op=2 actio  
ns=FLOOD  
 cookie=0x0, duration=26.4615, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:10,d1  
st=00:00:00:00:09,arp_spa=106.44.82.103,arp_tpa=108.24.31.112,arp_op=1 actio  
ns=FLOOD  
 cookie=0x0, duration=26.4615, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:10,d1  
st=00:00:00:00:10,arp_spa=108.24.31.112,arp_tpa=106.44.82.103,arp_op=1 actio  
ns=FLOOD  
 cookie=0x0, duration=26.3895, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:10,d1  
st=00:00:00:00:09,arp_spa=106.44.82.103,arp_tpa=108.24.31.112,arp_op=2 actio  
ns=FLOOD  
*** s11 -----  
NXST_FLOW reply (xid=0x4):  
 cookie=0x0, duration=26.4595, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:09,d1  
st=00:00:00:00:10,arp_spa=108.24.31.112,arp_tpa=106.44.82.103,arp_op=2 actio  
ns=FLOOD  
 cookie=0x0, duration=26.4685, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:10,d1  
st=00:00:00:00:09,arp_spa=106.44.82.103,arp_tpa=108.24.31.112,arp_op=1 actio  
ns=FLOOD  
 cookie=0x0, duration=26.4675, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:09,d1
```

```
mininet@mininet-vm: ~  
File Edit Tabs Help  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:09,d1  
st=00:00:00:00:10,arp_spa=108.24.31.112,arp_tpa=106.44.82.103,arp_op=2 actio  
ns=FLOOD  
 cookie=0x0, duration=26.4635, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:10,d1  
st=00:00:00:00:09,arp_spa=106.44.82.103,arp_tpa=108.24.31.112,arp_op=1 actio  
ns=FLOOD  
 cookie=0x0, duration=26.4635, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:09,d1  
st=00:00:00:00:10,arp_spa=108.24.31.112,arp_tpa=106.44.82.103,arp_op=1 actio  
ns=FLOOD  
 cookie=0x0, duration=26.3945, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:10,d1  
st=00:00:00:00:09,arp_spa=106.44.82.103,arp_tpa=108.24.31.112,arp_op=2 actio  
ns=FLOOD  
*** s21 -----  
NXST_FLOW reply (xid=0x4):  
 cookie=0x0, duration=26.475, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:09,d1  
st=00:00:00:00:10,arp_spa=108.24.31.112,arp_tpa=106.44.82.103,arp_op=2 actio  
ns=FLOOD  
 cookie=0x0, duration=26.4715, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:10,d1  
st=00:00:00:00:09,arp_spa=106.44.82.103,arp_tpa=108.24.31.112,arp_op=1 actio  
ns=FLOOD  
 cookie=0x0, duration=26.475, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:09,d1  
st=00:00:00:00:10,arp_spa=108.24.31.112,arp_tpa=106.44.82.103,arp_op=1 actio  
ns=FLOOD  
 cookie=0x0, duration=26.45, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:10,d1  
st=00:00:00:00:09,arp_spa=106.44.82.103,arp_tpa=108.24.31.112,arp_op=2 actio  
ns=FLOOD  
*** s22 -----  
NXST_FLOW reply (xid=0x4):  
 cookie=0x0, duration=26.4695, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:09,d1  
st=00:00:00:00:10,arp_spa=108.24.31.112,arp_tpa=106.44.82.103,arp_op=2 actio  
ns=FLOOD  
 cookie=0x0, duration=26.4775, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:10,d1  
st=00:00:00:00:09,arp_spa=106.44.82.103,arp_tpa=108.24.31.112,arp_op=1 actio  
ns=FLOOD  
 cookie=0x0, duration=26.4765, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:09,d1  
st=00:00:00:00:10,arp_spa=108.24.31.112,arp_tpa=106.44.82.103,arp_op=1 actio  
ns=FLOOD  
 cookie=0x0, duration=26.4835, table=0, n_packets=1, n_bytes=42, idle_timeout=30  
 , hard_timeout=30, idle_age=26, arp_vlan_tci=0x0000,d1_src=00:00:00:00:10,d1  
st=00:00:00:00:09,arp_spa=106.44.82.103,arp_tpa=108.24.31.112,arp_op=2 actio  
ns=FLOOD  
mininet>
```

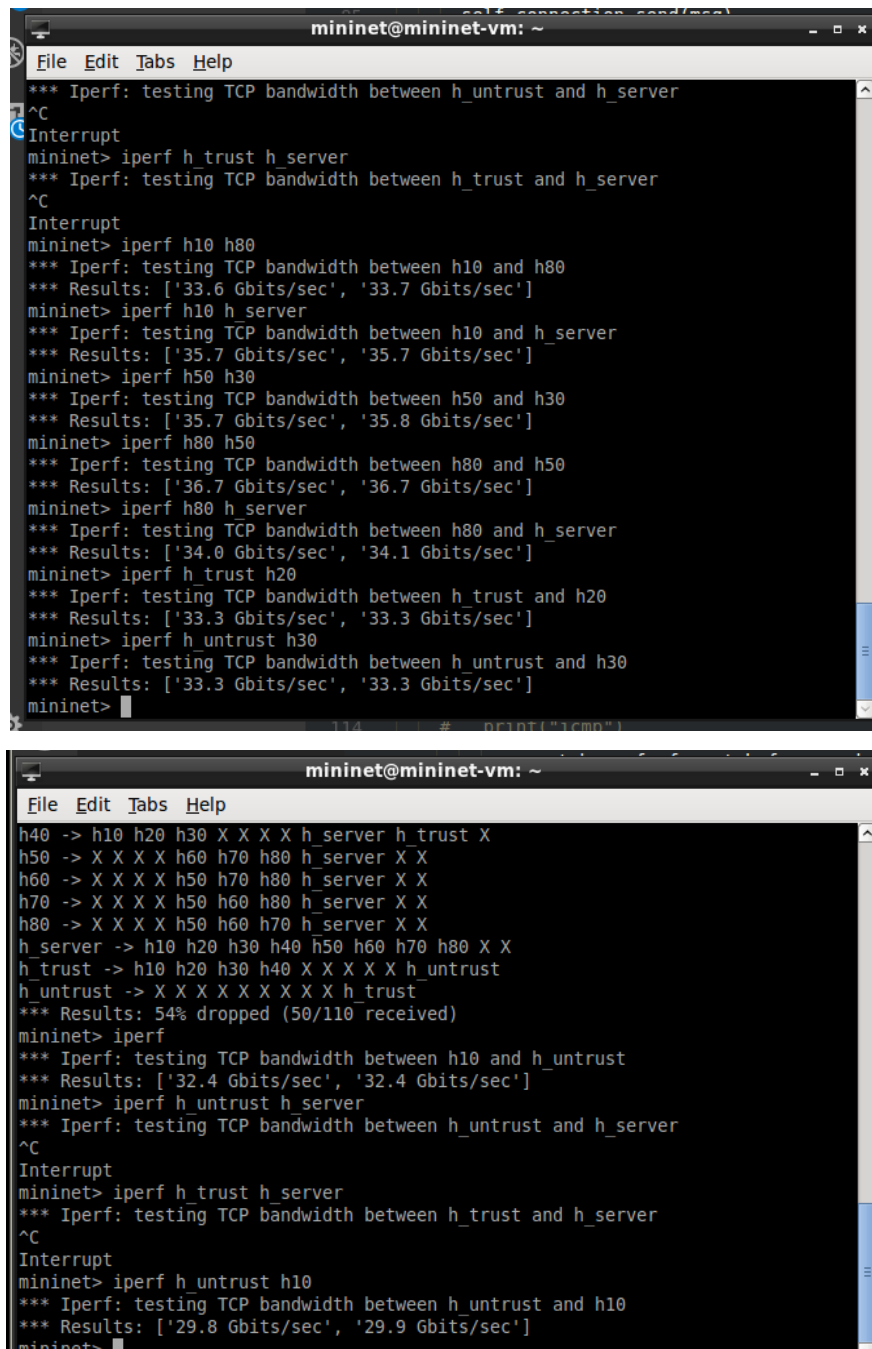
After running the *pingall* command, I immediately ran the *dpctl dump-flows* command to examine the flow table entries within the OpenFlow switch. The purpose of this command was to examine how the network was handling different types of traffic.

The output of the *dpctl dump-flows* command revealed that all ARP packets were flooded, indicating that the Router was functioning properly. ARP (Address Resolution Protocol) is a non-IP packet that is used to resolve IP addresses to MAC addresses within a network. Given that ARP packets are non-IP packets, it was expected that they would be flooded by the Router.

Flooding ARP packets is considered an appropriate behavior as these packets contain no substantial data and are primarily used for discovering and mapping MAC addresses of devices within the network. By flooding ARP packets, the network ensures that all devices are able to communicate and resolve IP-to-MAC address mappings effectively. The fact that all ARP packets were successfully flooded demonstrated that our network was meeting the requirements set out in the lab assignment.

In conclusion, the output of the *dpctl dump-flows* command confirmed that the network was appropriately handling non-IP packets by flooding them as required. This further validated the effectiveness of our network in meeting the specified lab requirements.

Iperf Commands



```
mininet@mininet-vm: ~  
File Edit Tabs Help  
*** Iperf: testing TCP bandwidth between h_untrust and h_server  
^C  
Interrupt  
mininet> iperf h_trust h_server  
*** Iperf: testing TCP bandwidth between h_trust and h_server  
^C  
Interrupt  
mininet> iperf h10 h80  
*** Iperf: testing TCP bandwidth between h10 and h80  
*** Results: ['33.6 Gbits/sec', '33.7 Gbits/sec']  
mininet> iperf h10 h_server  
*** Iperf: testing TCP bandwidth between h10 and h_server  
*** Results: ['35.7 Gbits/sec', '35.7 Gbits/sec']  
mininet> iperf h50 h30  
*** Iperf: testing TCP bandwidth between h50 and h30  
*** Results: ['35.7 Gbits/sec', '35.8 Gbits/sec']  
mininet> iperf h80 h50  
*** Iperf: testing TCP bandwidth between h80 and h50  
*** Results: ['36.7 Gbits/sec', '36.7 Gbits/sec']  
mininet> iperf h80 h_server  
*** Iperf: testing TCP bandwidth between h80 and h_server  
*** Results: ['34.0 Gbits/sec', '34.1 Gbits/sec']  
mininet> iperf h_trust h20  
*** Iperf: testing TCP bandwidth between h_trust and h20  
*** Results: ['33.3 Gbits/sec', '33.3 Gbits/sec']  
mininet> iperf h_untrust h30  
*** Iperf: testing TCP bandwidth between h_untrust and h30  
*** Results: ['33.3 Gbits/sec', '33.3 Gbits/sec']  
mininet>  
113 # print("cmd")  
  
mininet@mininet-vm: ~  
File Edit Tabs Help  
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 h_untrust  
h_untrust -> X X X X X X X X h_trust  
*** Results: 54% dropped (50/110 received)  
mininet> iperf  
*** Iperf: testing TCP bandwidth between h10 and h_untrust  
*** Results: ['32.4 Gbits/sec', '32.4 Gbits/sec']  
mininet> iperf h_untrust h_server  
*** Iperf: testing TCP bandwidth between h_untrust and h_server  
^C  
Interrupt  
mininet> iperf h_trust h_server  
*** Iperf: testing TCP bandwidth between h_trust and h_server  
^C  
Interrupt  
mininet> iperf h_untrust h10  
*** Iperf: testing TCP bandwidth between h_untrust and h10  
*** Results: ['29.8 Gbits/sec', '29.9 Gbits/sec']  
mininet>
```

Running the *pingall* command helped to verify ICMP restrictions, however, in order to verify IP restrictions, I conducted *iperf* tests between specific hosts. *Iperf*, a network performance measurement tool, utilizes TCP as its underlying protocol, which directly interacts with IP packets. Therefore,

the success of TCP connections serves as an indicator of the proper transmission of IP packets.

Based on the project instructions, restrictions were set to prevent the Trusted and Untrusted Hosts from sending IP packets to the Server. Analyzing the image above, it is evident that when I ran *iperf* tests between these hosts, the output became stalled without producing any result. This behavior signifies that the transmitted packet was immediately dropped and failed to traverse through the network. The restrictions effectively stopped IP packets from the Trusted and Untrusted Hosts towards the Server.

Furthermore, to ensure that the network was not immediately dropping all IP packets regardless of their destination, I conducted *iperf* tests between hosts that were expected to be able to exchange IP packets. This additional verification aimed to confirm that the network was selectively blocking IP packets as per the assigned restrictions. By running *iperf* tests between these hosts, I was able to confirm that the network was indeed allowing the desired communication of IP packets between the designated hosts.

In summary, the *iperf* tests and their corresponding outcomes provide compelling evidence of the proper implementation of IP packet restrictions within the network.

Dump Commands

[illegible]

I ran a simple *dump* command to ensure that all the hosts in my network were properly configured based on the specified topology. The purpose of this command was to validate that each host had the correct IP addresses assigned to them, which plays a crucial role in accurately defining and implementing our network rules.

Upon analyzing the provided screenshot, we can affirm that all the hosts indeed possess the correct IP addresses as per the defined topology. This information is crucial in our rule-writing process, as it ensures that our destination addresses align with the intended destinations within the network. Having accurate IP addresses assigned to each host allows for precise rule configuration, enabling effective traffic management and control.

By verifying the correctness of host IP addresses through the *dump* command, we can proceed with confidence in designing and implementing network rules, knowing that our rule definitions will be accurately targeted and applied to the appropriate hosts. This validation step ensures the smooth functioning of the network, minimizing the potential for misconfigured rules and facilitating seamless communication between hosts.

```
mininet@mininet-vm: ~  
File Edit Tabs Help  
*** Results: ['36.7 Gbits/sec', '36.7 Gbits/sec']  
mininet> iperf h80 h_server  
*** Iperf: testing TCP bandwidth between h80 and h_server  
*** Results: ['34.0 Gbits/sec', '34.1 Gbits/sec']  
mininet> iperf h_trust h20  
*** Iperf: testing TCP bandwidth between h_trust and h20  
*** Results: ['33.3 Gbits/sec', '33.3 Gbits/sec']  
mininet> iperf h_untrust h30  
*** Iperf: testing TCP bandwidth between h_untrust and h30  
*** Results: ['33.3 Gbits/sec', '33.3 Gbits/sec']  
mininet> h_untrust ping -c 1 h10  
PING 10.1.1.10 (10.1.1.10) 56(84) bytes of data.  
  
--- 10.1.1.10 ping statistics ---  
1 packets transmitted, 0 received, 100% packet loss, time 0ms  
  
mininet> h_untrust ping -c 10 h10  
PING 10.1.1.10 (10.1.1.10) 56(84) bytes of data.  
  
--- 10.1.1.10 ping statistics ---  
10 packets transmitted, 0 received, 100% packet loss, time 9070ms  
  
mininet> h_trust ping -c 5 h50  
PING 10.2.5.50 (10.2.5.50) 56(84) bytes of data.  
  
--- 10.2.5.50 ping statistics ---  
5 packets transmitted, 0 received, 100% packet loss, time 4031ms  
  
mininet>   
114 # print("icmp")
```

Here you can see a few more successful *iperf* commands that I ran as well as a few *ping* commands as a sanity check. Since Untrusted cannot send ICMP packets anywhere, I ran a ping between Untrusted and Host 10 and saw that all 10 of the ping packets were dropped. Furthermore, we know the Department A (host 10, 20, 30, 40) cannot send any ICMP pings to Department B (host 50, 60, 70, 80). This restriction was validated through the above screenshot as well since we see that all packets going from host 10 to host 50 were dropped.