Final Report

My controller first gets the protocols TCP, ARP, and ICMP through the usage of packet.find(). After getting the protocols, my controller then continues to set up the rules. Firstly, if the packet that was received is of ARP protocol, then what my controller does is to first match the packet to a variable, then send flood that rule to all of the switches so the rest of the switches will know what to do if they ever come across the same packet. The second thing that it does is to flood the packet as ARP packets are non-IP traffic. Secondly if the packet is of TCP or ICMP protocol, it will do the following: match the current switch id to either the switches connected to host1, host2, host3, or server, or the core switch itself. If the switch id is the switches connected either to host1, host2, host3, or the server then it will check the port number that the packet was received on. If the port number connects with the port number of the hosts or the server, then first match the packet and send it to the switch's flow table, and secondly send the packet through the port number connecting to the port number of the core switch. Else that means the packet is coming from the core switch so the controller will match the packet and send it through the current switch's flow table, then send the packet through the port connecting to its respected host or server. Else if the switch id matches the id of the core switch then it will do the following. First the controller will get the ip of the packet. If the source ip is the untrusted host and the current protocol is ICMP, then the controller will match the packet and add it to the flow table, then drop the packet. Else if the destination ip is of host1, host2, host3, or untrusted host then the controller will match the packet, add it to the flow table, then pass the packet through the port number that connects to the port number of the switch that is respectfully connected to the destination ip (host ip), or to the untrusted host if the destination ip is the untrusted host ip. However, if the destination ip is the server ip, then it will do the following. If the source ip is the untrusted host ip, then the controller will match the packet and save it into its flow table, then drop the packet. Else if the source ip is not the untrusted host ip, the controller will match the packet then save it to the flow table, and then pass the packet through the port number that is connected with the switch that is connected to the server.

For proof of correctness, when running iperf commands which gives TCP packets, my controller is able to pass the TCP packets from hosts to untrusted host or server, and block TCP packets from untrusted host to server. As displayed by the following screenshot:

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['30.4 Gbits/sec', '30.4 Gbits/sec']
mininet>
```

Host 1 and host 2 are able to communicate to each other through iperf and as a result, the corresponding switches get their flow tables filled in after they have received the packet.

```
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=20.819s, table=0, n_packets=480249, n_bytes=19030802426, idle_age=
15, tcp,vlan_tci=0x0000,dl_src=fe:76:1b:6e:fd:6d,dl_dst=12:45:b7:8b:17:8e,nw_src=10.0.1.
101,nw_dst=10.0.2.102,nw_tos=0,tp_src=56834,tp_dst=5001 actions=output:11
 cookie=0x0, duration=20.824s, table=0, n_packets=2, n_bytes=132, idle_age=20, tcp,vlan_
tci=0x0000,dl_src=12:45:b7:8b:17:8e,dl_dst=fe:76:1b:6e:fd:6d,nw_src=10.0.2.102,nw_dst=10
.0.1.101,nw_tos=0,tp_src=5001,tp_dst=56833 actions=output:1
 cookie=0x0, duration=20.683s, table=0, n_packets=130498, n_bytes=8618920, idle_age=15,
tcp,vlan_tci=0x0000,dl_src=12:45:b7:8b:17:8e,dl_dst=fe:76:1b:6e:fd:6d,nw_src=10.0.2.102,
nw_dst=10.0.1.101,nw_tos=0,tp_src=5001,tp_dst=56834 actions=output:1
 cookie=0x0, duration=20.965s, table=0, n_packets=3, n_bytes=198, idle_age=20, tcp,vlan_
tci=0x0000,dl_src=fe:76:1b:6e:fd:6d,dl_dst=12:45:b7:8b:17:8e,nw_src=10.0.1.101,nw_dst=10
.0.2.102,nw_tos=16,tp_src=56833,tp_dst=5001 actions=output:11
 cookie=0x0, duration=21.042s, table=0, n_packets=0, n_bytes=0, idle_age=21, arp,vlan_tc
i=0x0000,dl_src=fe:76:1b:6e:fd:6d,dl_dst=ff:ff:ff:ff:ff:ff,arp_spa=10.0.1.101,arp_tpa=10
.0.2.102,arp_op=1 actions=FLOOD
 cookie=0x0, duration=20.967s, table=0, n_packets=0, n_bytes=0, idle_age=20, arp,vlan_tc
i=0x0000,dl_src=12:45:b7:8b:17:8e,dl_dst=fe:76:1b:6e:fd:6d,arp_spa=10.0.2.102,arp_tpa=10
.0.1.101,arp_op=2 actions=FLOOD
```

```
*** s2 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=20.744s, table=0, n_packets=480249, n_bytes=19030802426, idle_age=
15, tcp,vlan_tci=0x0000,dl_src=fe:76:1b:6e:fd:6d,dl_dst=12:45:b7:8b:17:8e,nw_src=10.0.1.
101,nw_dst=10.0.2.102,nw_tos=0,tp_src=56834,tp_dst=5001 actions=output:2
 cookie=0x0, duration=20.894s, table=0, n_packets=2, n_bytes=132, idle_age=20, tcp,vlan_
tci=0x0000,dl_src=12:45:b7:8b:17:8e,dl_dst=fe:76:1b:6e:fd:6d,nw_src=10.0.2.102,nw_dst=10
.0.1.101,nw_tos=0,tp_src=5001,tp_dst=56833 actions=output:21
 cookie=0x0, duration=20.74s, table=0, n_packets=130498, n_bytes=8618920, idle_age=15, t
cp,vlan_tci=0x0000,dl_src=12:45:b7:8b:17:8e,dl_dst=fe:76:1b:6e:fd:6d,nw_src=10.0.2.102,n
w_dst=10.0.1.101,nw_tos=0,tp_src=5001,tp_dst=56834 actions=output:21
 cookie=0x0, duration=20.896s, table=0, n_packets=3, n_bytes=198, idle_age=20, tcp,vlan_
tci=0x0000,dl_src=fe:76:1b:6e:fd:6d,dl_dst=12:45:b7:8b:17:8e,nw_src=10.0.1.101,nw_dst=10
.0.2.102,nw_tos=16,tp_src=56833,tp_dst=5001 actions=output:2
 cookie=0x0, duration=21.04s, table=0, n_packets=0, n_bytes=0, idle_age=21, arp,vlan_tci
=0x0000,dl_src=fe:76:1b:6e:fd:6d,dl_dst=ff:ff:ff:ff:ff:ff,arp_spa=10.0.1.101,arp_tpa=10.
0.2.102,arp_op=1 actions=FLOOD
 cookie=0x0, duration=21.038s, table=0, n_packets=0, n_bytes=0, idle_age=21, arp,vlan_tc
i=0x0000,dl_src=12:45:b7:8b:17:8e,dl_dst=fe:76:1b:6e:fd:6d,arp_spa=10.0.2.102,arp_tpa=10
.0.1.101,arp_op=2 actions=FLOOD
```

```
*** s4 ----------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=20.755s, table=0, n_packets=480249, n_bytes=19030802426, idle_age=
15, tcp,vlan_tci=0x0000,dl_src=fe:76:1b:6e:fd:6d,dl_dst=12:45:b7:8b:17:8e,nw_src=10.0.1.
101,nw_dst=10.0.2.102,nw_tos=0,tp_src=56834,tp_dst=5001 actions=output:42
 cookie=0x0, duration=20.835s, table=0, n_packets=2, n_bytes=132, idle_age=20, tcp,vlan_
tci=0x0000,dl_src=12:45:b7:8b:17:8e,dl_dst=fe:76:1b:6e:fd:6d,nw_src=10.0.2.102,nw_dst=10
.0.1.101,nw_tos=0,tp_src=5001,tp_dst=56833 actions=output:41
 cookie=0x0, duration=20.696s, table=0, n_packets=130498, n_bytes=8618920, idle_age=15,
tcp,vlan_tci=0x0000,dl_src=12:45:b7:8b:17:8e,dl_dst=fe:76:1b:6e:fd:6d,nw_src=10.0.2.102,
nw_dst=10.0.1.101,nw_tos=0,tp_src=5001,tp_dst=56834 actions=output:41
 cookie=0x0, duration=20.903s, table=0, n_packets=3, n_bytes=198, idle_age=20, tcp,vlan_
tci=0x0000,dl_src=fe:76:1b:6e:fd:6d,dl_dst=12:45:b7:8b:17:8e,nw_src=10.0.1.101,nw_dst=10
.0.2.102,nw_tos=16,tp_src=56833,tp_dst=5001 actions=output:42
 cookie=0x0, duration=21.049s, table=0, n_packets=0, n_bytes=0, idle_age=21, arp,vlan_tc
i=0x0000,dl_src=fe:76:1b:6e:fd:6d,dl_dst=ff:ff:ff:ff:ff:ff,arp_spa=10.0.1.101,arp_tpa=10
.0.2.102,arp_op=1 actions=FLOOD
 cookie=0x0, duration=20.979s, table=0, n_packets=0, n_bytes=0, idle_age=20, arp,vlan_tc
i=0x0000,dl_src=12:45:b7:8b:17:8e,dl_dst=fe:76:1b:6e:fd:6d,arp_spa=10.0.2.102,arp_tpa=10
.0.1.101,arp_op=2 actions=FLOOD
```

From the screenshots above displaying the flow tables of switches 1, 2 and core switch; we can see that the rules has been applied where the packet from h1 to h2 has to be passed through port 11 for switch 1, then pass through port 42 for core switch, and finally pass through port 2 for switch 2. Additionally by running iperf for host1 to untrusted host we can see the following:

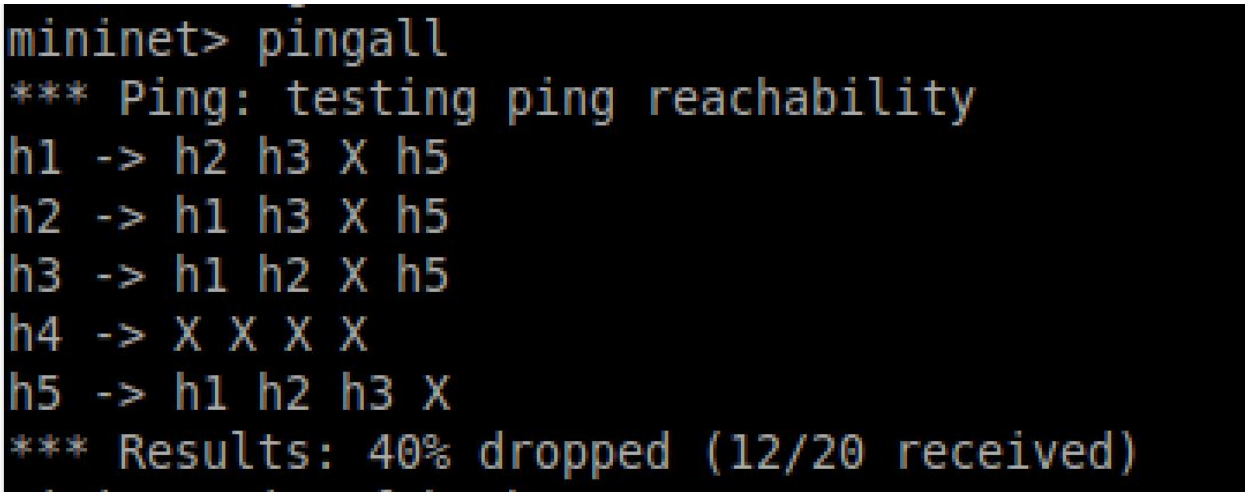
```
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['44.6 Gbits/sec', '44.6 Gbits/sec']
mininet>
```

where host 1 is able to communicate to the untrusted host through TCP connection. However, if untrusted host tries to communicate to the server, then the packet will be blocked which is proved by the following screenshot:

```
mininet> iperf h4 h5
*** Iperf: testing TCP bandwidth between h4 and h5
c^C
Interrupt
mininet>
```

where it will continue to hang as it keeps attempting to connect the untrusted host with the server, but the packet from host4 is constantly being dropped by the core switch.

Additionally by running pingall or any sort of ping command, we get the following:



which satisfies the criteria where any packet from host4 should be dropped if the packet is an ICMP protocol. In this instance, all hosts are able to communicate with each other, and with the server as well, however, with the exception of the untrusted host (host4). This shows that my code not only works with ARP and TCP packets, but also with ICMP packets.