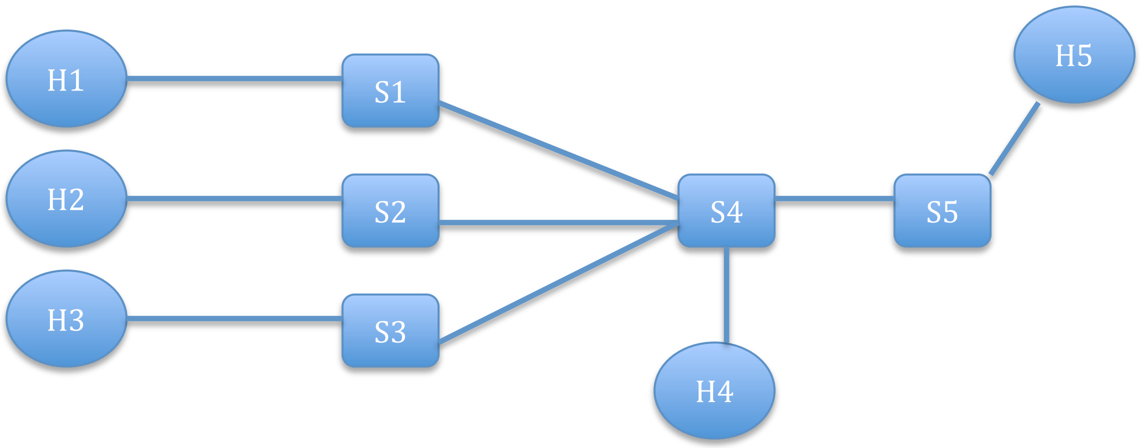
Edbel Basaldua
Ebasaldu@ucsc.edu
ID: 1413712
CE150: FinalProject
March 20,2018

Final Project Explanations and Commands:

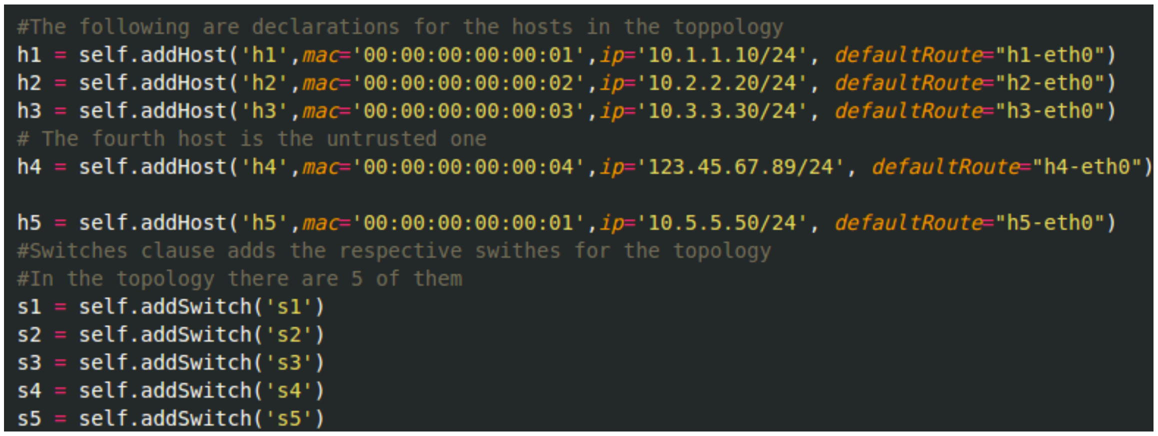
Note: This project is incomplete It is mostly due to the confusion in setting up the ports and not properly tracing where everything is going.

Below Is the commands I would run to demo my program However these are not the right results but I will provide details explanations as to what should be there and why it gave me the wrong output. I will try to explain bits of my code and show what I was attempting to do and explain my logic behind it:

Final.PY code Explanation:
So the purpose of this was to set up the topology to reflect what was shown in the assignment pdf.



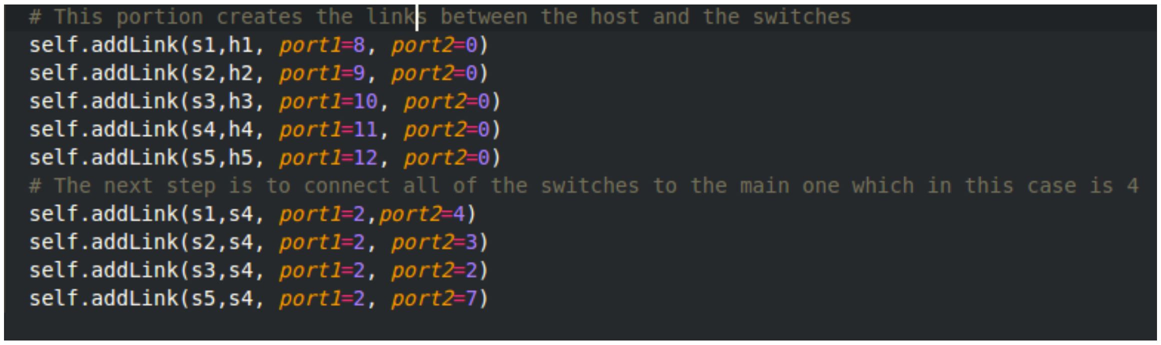
In my code I did the following to set up the hosts with their respective Ip addresses as was stated in the assignment and then declared all hosts.



```
#The following are declarations for the hosts in the toppology
h1 = self.addHost('h1',mac='00:00:00:00:00:01',ip='10.1.1.10/24', defaultRoute="h1-eth0")
h2 = self.addHost('h2',mac='00:00:00:00:00:02',ip='10.2.2.20/24', defaultRoute="h2-eth0")
h3 = self.addHost('h3',mac='00:00:00:00:00:03',ip='10.3.3.30/24', defaultRoute="h3-eth0")
# The fourth host is the untrusted one
h4 = self.addHost('h4',mac='00:00:00:00:00:04',ip='123.45.67.89/24', defaultRoute="h4-eth0")

h5 = self.addHost('h5',mac='00:00:00:00:00:01',ip='10.5.5.50/24', defaultRoute="h5-eth0")
#Switches clause adds the respective swithes for the topology
#In the topology there are 5 of them
s1 = self.addSwitch('s1')
s2 = self.addSwitch('s2')
s3 = self.addSwitch('s3')
s4 = self.addSwitch('s4')
s5 = self.addSwitch('s5')
```

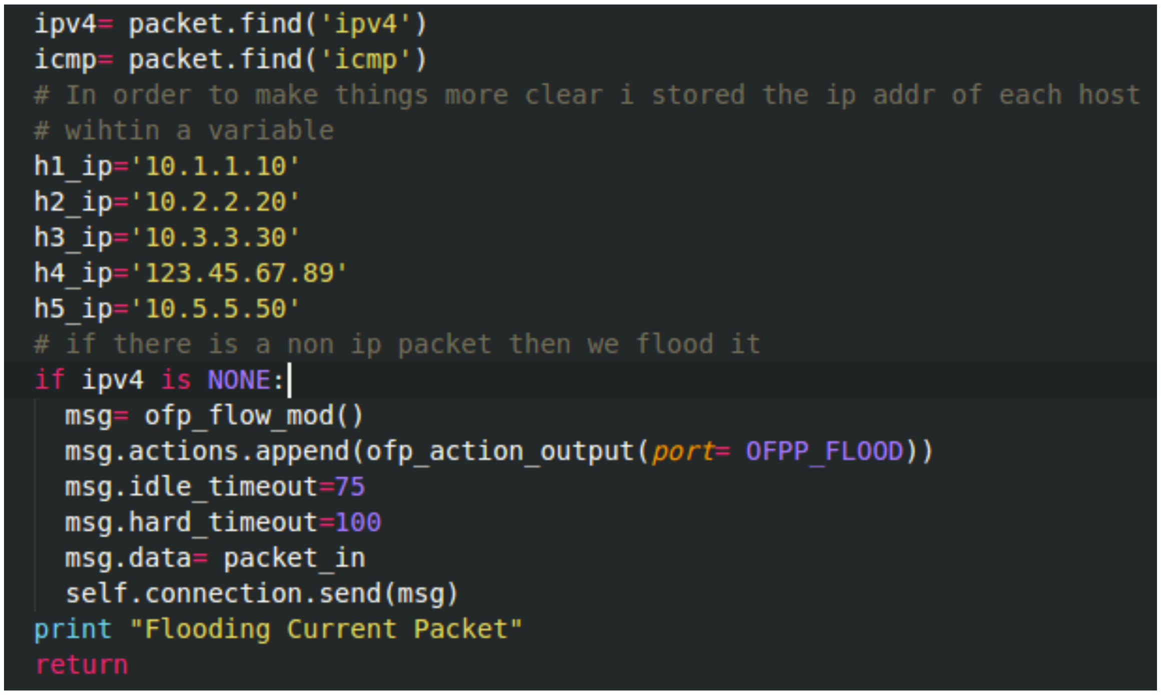
In this next section is set up the links between the hosts and the switches and then from the switches to the fourth switch.



```
# This portion creates the links between the host and the switches
self.addLink(s1,h1, port1=8, port2=0)
self.addLink(s2,h2, port1=9, port2=0)
self.addLink(s3,h3, port1=10, port2=0)
self.addLink(s4,h4, port1=11, port2=0)
self.addLink(s5,h5, port1=12, port2=0)
# The next step is to connect all of the switches to the main one which in this case is 4
self.addLink(s1,s4, port1=2,port2=4)
self.addLink(s2,s4, port1=2, port2=3)
self.addLink(s3,s4, port1=2, port2=2)
self.addLink(s5,s4, port1=2, port2=7)
```

Final Controller.py Code logic Explanation:

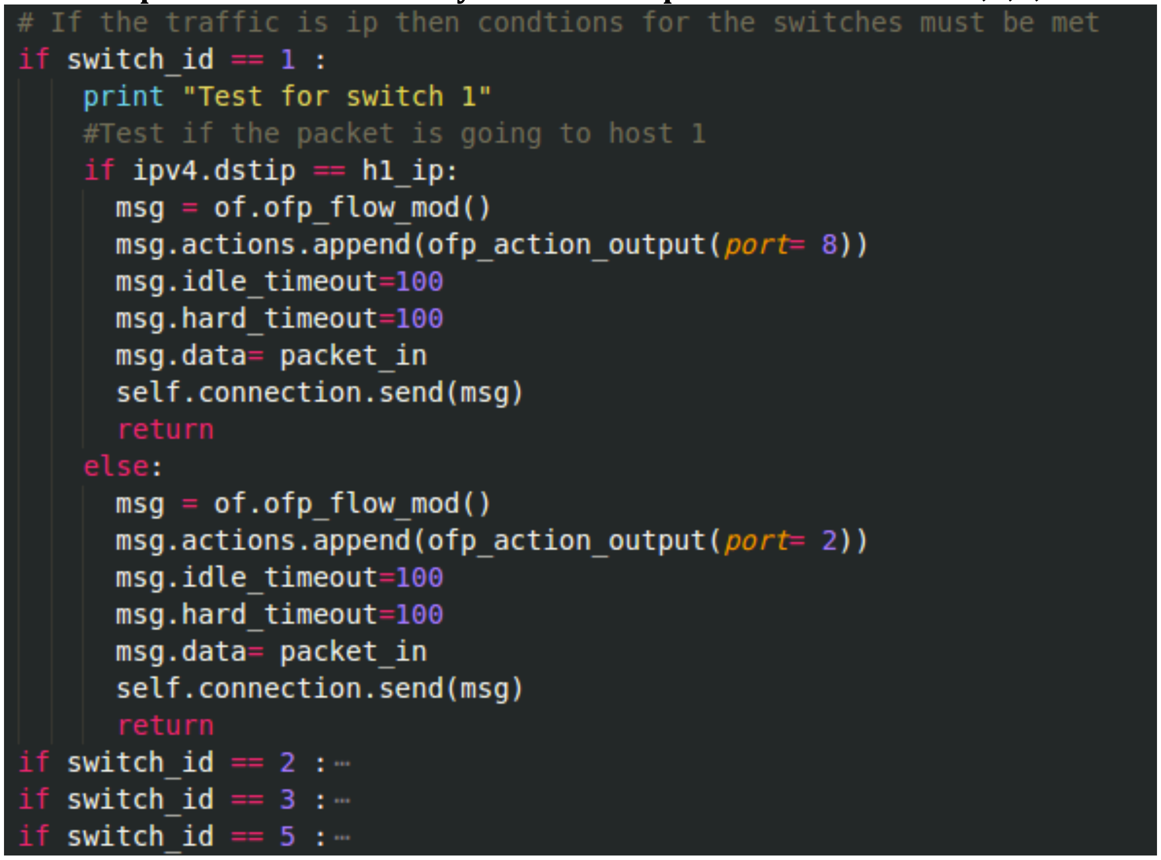
1. **In this secton of the code there is a few data declarations these make use of the find function which identifys the packet type. In this case the two we want are the ip packets and the icmp packets.**
2. **Secondly I then used variables to store the appropriate ip addresses for later use on conditionals**
3. **The first conditional clause is used to check if there is any Non IP traffic if so then the block of code attempts to FLOOD these packets**



```
ipv4= packet.find('ipv4')
icmp= packet.find('icmp')
# In order to make things more clear i stored the ip addr of each host
# wihtin a variable
h1_ip='10.1.1.10'
h2_ip='10.2.2.20'
h3_ip='10.3.3.30'
h4_ip='123.45.67.89'
h5_ip='10.5.5.50'
# if there is a non ip packet then we flood it
if ipv4 is NONE:
  msg= ofp_flow_mod()
  msg.actions.append(ofp_action_output(port= OFPP_FLOOD))
  msg.idle_timeout=75
  msg.hard_timeout=100
  msg.data= packet_in
  self.connection.send(msg)
print "Flooding Current Packet"
return
```

Edbel Basaldua
Ebasaldu@ucsc.edu
ID: 1413712
CE150: FinalProject
March 20,2018

The next part of the code is very clear. It is repeated for switches 1 ,2,3, and 5:

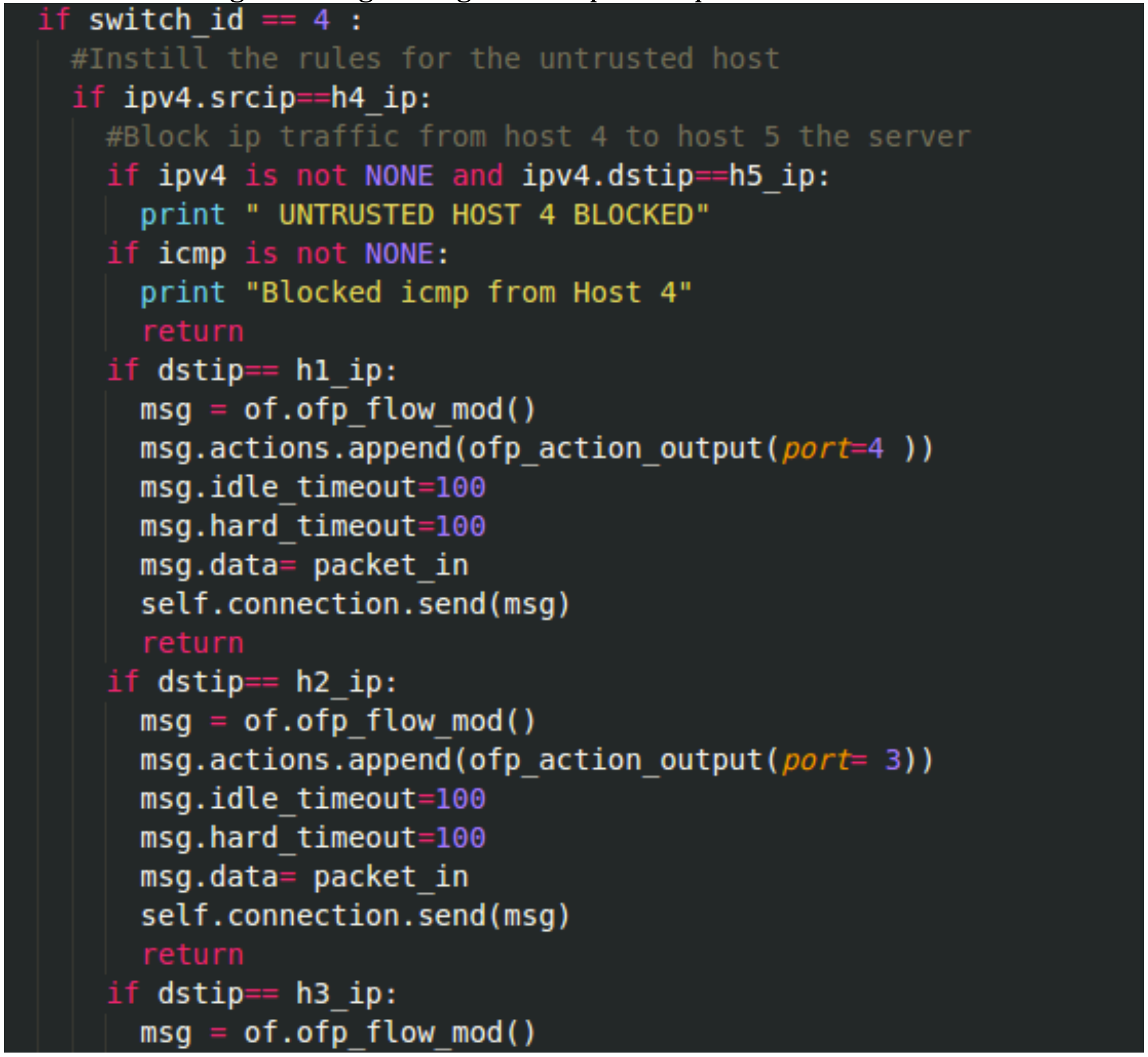


```
# If the traffic is ip then condtions for the switches must be met
if switch_id == 1 :
    print "Test for switch 1"
    #Test if the packet is going to host 1
    if ipv4.dstip == h1_ip:
      msg = of.ofp_flow_mod()
      msg.actions.append(ofp_action_output(port= 8))
      msg.idle_timeout=100
      msg.hard_timeout=100
      msg.data= packet_in
      self.connection.send(msg)
      return
    else:
      msg = of.ofp_flow_mod()
      msg.actions.append(ofp_action_output(port= 2))
      msg.idle_timeout=100
      msg.hard_timeout=100
      msg.data= packet_in
      self.connection.send(msg)
      return
if switch_id == 2 :⋯
if switch_id == 3 :⋯
if switch_id == 5 :⋯
```

Here what I was attempting to do was to identify which switch should be checked for using **switch_id=# .** Then I have a debugging statement for my own purposes. The next conditionals **branches check to see whether the packet destination is going to the host.** IN this case it checks if the packet is going to host 1. If so then it send the packet to the specified port. Otherwise if the packet destination is not the host connected to the switch then we send it to the other port.

Note: This process was done for switch 1, switch 2, switch 3, and switch 5. The only lines that changed was the line :
switch_id=#
if ipv4.dstip == h#_ip:
and then the respective ports for each switch and host.

Switch 4 Procedures:
In this case if the switch was the 4th one we checked what the packet source was and compared it to the untrusted host ip. We then attempt to block any traffic from the 4th host to the server. We also block any of the icmp traffic too.
The next few clauses of code check it the destination is anything but the server and allows for message sending through the respective ports.



```
if switch_id == 4 :
  #Instill the rules for the untrusted host
  if ipv4.srcip==h4_ip:
    #Block ip traffic from host 4 to host 5 the server
    if ipv4 is not NONE and ipv4.dstip==h5_ip:
      print " UNTRUSTED HOST 4 BLOCKED"
    if icmp is not NONE:
      print "Blocked icmp from Host 4"
      return
    if dstip== h1_ip:
      msg = of.ofp_flow_mod()
      msg.actions.append(ofp_action_output(port=4 ))
      msg.idle_timeout=100
      msg.hard_timeout=100
      msg.data= packet_in
      self.connection.send(msg)
      return
    if dstip== h2_ip:
      msg = of.ofp_flow_mod()
      msg.actions.append(ofp_action_output(port= 3))
      msg.idle_timeout=100
      msg.hard_timeout=100
      msg.data= packet_in
      self.connection.send(msg)
      return
    if dstip== h3_ip:
      msg = of.ofp_flow_mod()
```

Commands to be run:

If my program was fully functional I would have run these commands.

1. **Nodes**
2. **Links**
3. **Dump**
4. **Pingall**

The following is the output I get running the current code I have but I will explain:

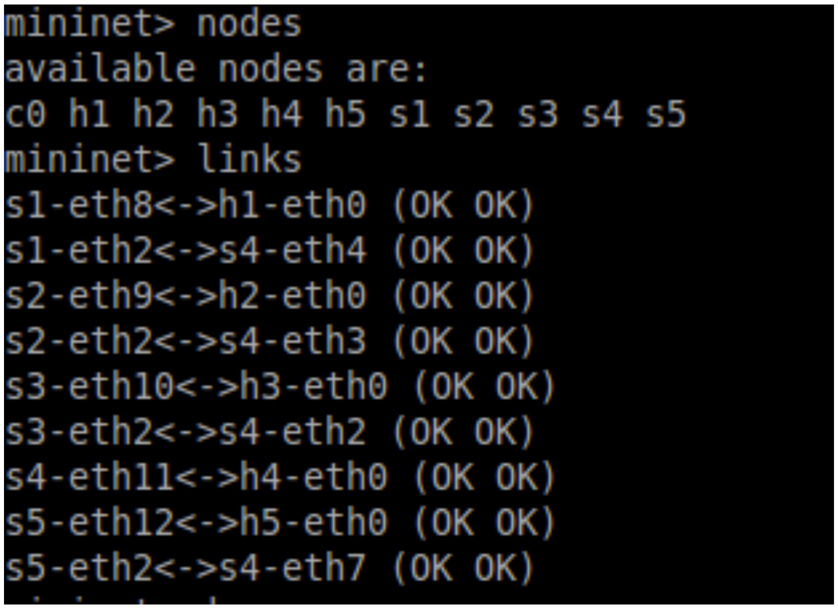
Edbel Basaldua
Ebasaldu@ucsc.edu
ID: 1413712
CE150: FinalProject
March 20,2018

Nodes Command and the Links Command:

When I run the nodes command it gives me all the hosts available and the total Amount of switches present in the topology

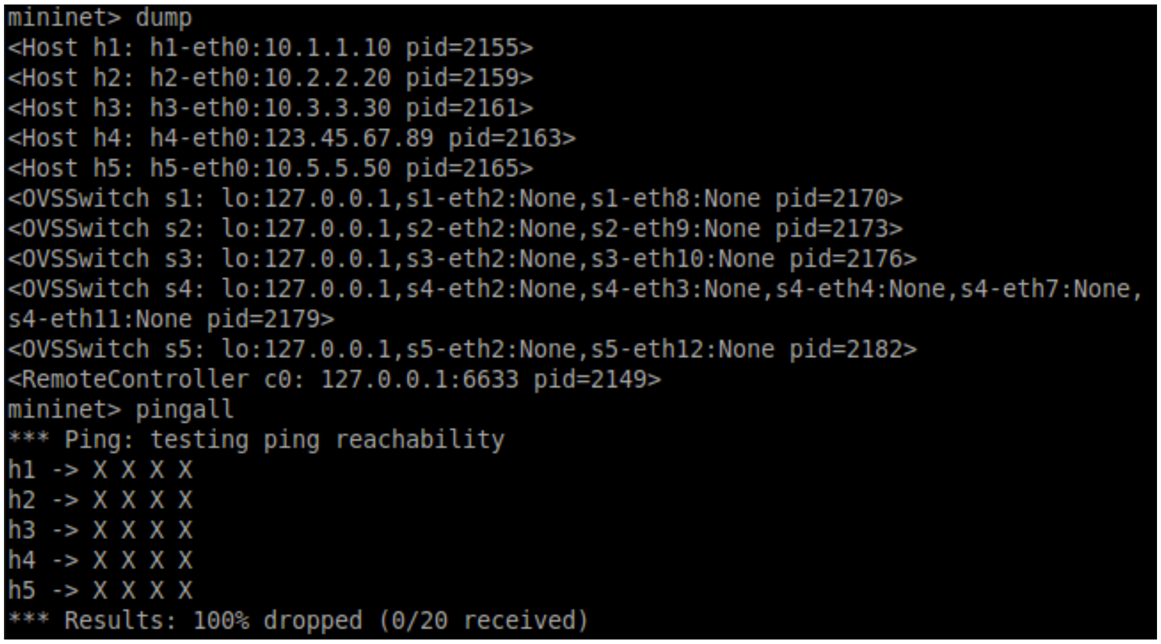
When I ran the Links command it showed me all the links that I had defined within the topology file. It shows all the respective port numbers and how they are linked.

These did not suffer in term of functionality as of the confirmation tag on them.



```
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 h5 s1 s2 s3 s4 s5
mininet> links
s1-eth8<->h1-eth0 (OK OK)
s1-eth2<->s4-eth4 (OK OK)
s2-eth9<->h2-eth0 (OK OK)
s2-eth2<->s4-eth3 (OK OK)
s3-eth10<->h3-eth0 (OK OK)
s3-eth2<->s4-eth2 (OK OK)
s4-eth11<->h4-eth0 (OK OK)
s5-eth12<->h5-eth0 (OK OK)
s5-eth2<->s4-eth7 (OK OK)
```

DUMP and PINGALL command:



```
mininet> dump
<Host h1: h1-eth0:10.1.1.10 pid=2155>
<Host h2: h2-eth0:10.2.2.20 pid=2159>
<Host h3: h3-eth0:10.3.3.30 pid=2161>
<Host h4: h4-eth0:123.45.67.89 pid=2163>
<Host h5: h5-eth0:10.5.5.50 pid=2165>
<OVSSwitch s1: lo:127.0.0.1,s1-eth2:None,s1-eth8:None pid=2170>
<OVSSwitch s2: lo:127.0.0.1,s2-eth2:None,s2-eth9:None pid=2173>
<OVSSwitch s3: lo:127.0.0.1,s3-eth2:None,s3-eth10:None pid=2176>
<OVSSwitch s4: lo:127.0.0.1,s4-eth2:None,s4-eth3:None,s4-eth4:None,s4-eth7:None,
s4-eth11:None pid=2179>
<OVSSwitch s5: lo:127.0.0.1,s5-eth2:None,s5-eth12:None pid=2182>
<RemoteController c0: 127.0.0.1:6633 pid=2149>
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X
h2 -> X X X X
h3 -> X X X X
h4 -> X X X X
h5 -> X X X X
*** Results: 100% dropped (0/20 received)
```

The dump command is useful in showing that I had the proper IP addresses set up for each of the hosts. It also gives me the default port and the PID.

Most importantly it shows me the switches I believe that the correct output should not have the word NONE. I believe this error occurred because I do not have the port numbers set up the way I am visualizing them.

Lastly using the PINGALL command would have been useful in showing the connectivity among all the hosts and seeing who is connected to who. I think the correct output should have had just the UNTRUSTED HOST being marked with an X on the SERVER.

Again I think my logic for this assignment is correct but my visualization for the ports is incorrect that translated into my code and hence it is not fully functional.