Alex Davila
1465932
CSE150/L
Chen Qian

Introduction to Computer Networks

Final Project - Implementing a Simple Router

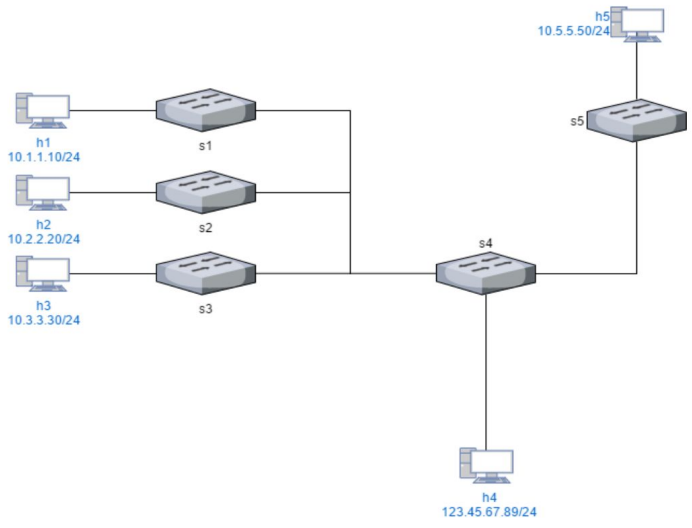
For my final project, I expanded on previous labs for implementing a simple firewall that allowed ARP and TCP traffic to implement routing between devices on different subnets, and implementing firewalls for certain subnets. The idea is to simulate an actual production network.

For this lab, I will be constructing a network for a small company. The company has a 3-floor building, with each floor having its own switch and subnet. Additionally, I have a switch and subnet for all the servers in the data center, and a core switch connecting everything together.

My device's roles and IP addresses are as follows:

Device	Mininet Name	IP Address	Description
Floor 1 Host	h1	10.1.1.10/24	A computer on floor 1 of the company.
Floor 2 Host	h2	10.2.2.20/24	A computer on floor 2 of the company.
Floor 3 Host	h3	10.3.3.30/24	A computer on floor 3 of the company.
Untrusted Host	h4	123.45.67.89/24	A computer outside our network. We treat this computer as a potential hacker.
Server	h5	10.5.5.50/24	A server used by our internal hosts.

And my topology looks as follows:



Goal: My goal is to allow traffic to be transmitted between all the hosts. I flood all non-IP traffic in the same method that I did in Lab 3 (using a destination port of of.OFPP_FLOOD). However, I specified specific ports for all IP traffic. There are many ways to do this-- however, I found it easiest to determine the correct destination port by using the destination IP address and source IP address, as well as the source port on the switch that the packet originated from. Additional information was given to me in the *do_final()* function which allowed me to make these decisions. Additionally, to protect our servers from the untrusted Internet, I will be blocking all IP traffic from the Untrusted Host to the Server. To block the Internet from discovering our internal IP addresses, I will also block all ICMP traffic from the Untrusted Host to anywhere internally.

For this project, I was provided with the Mininet configuration starter code (skeleton file), *final_skel.py*, to setup the network and topology which I placed under my home (~) directory. To create the topology I created a host with a default route of the ethernet interface. I set the default gateway like this for every host I made on this assignment to make sure all packets are sent out that port. I also made sure to change the h# in the defaultRoute area and the MAC address when I added more hosts!

For this project, I was also provided with a skeleton POX controller *final_controller_skel.py,* where I programmed a simple router to implement routing between devices on different subnets, and to implement firewalls for certain subnets as way to simulate an actual production network.

Summary of Goals

- Create a Mininet Topology to represent the above topology.
- Create a Pox controller with the following features:
    - All hosts able to communicate, EXCEPT:
        - Untrusted Host cannot send ICMP traffic to Host 1, Host 2, Host 3, or the Server.
        - Untrusted Host cannot send any IP traffic to the Server.

Running the Code

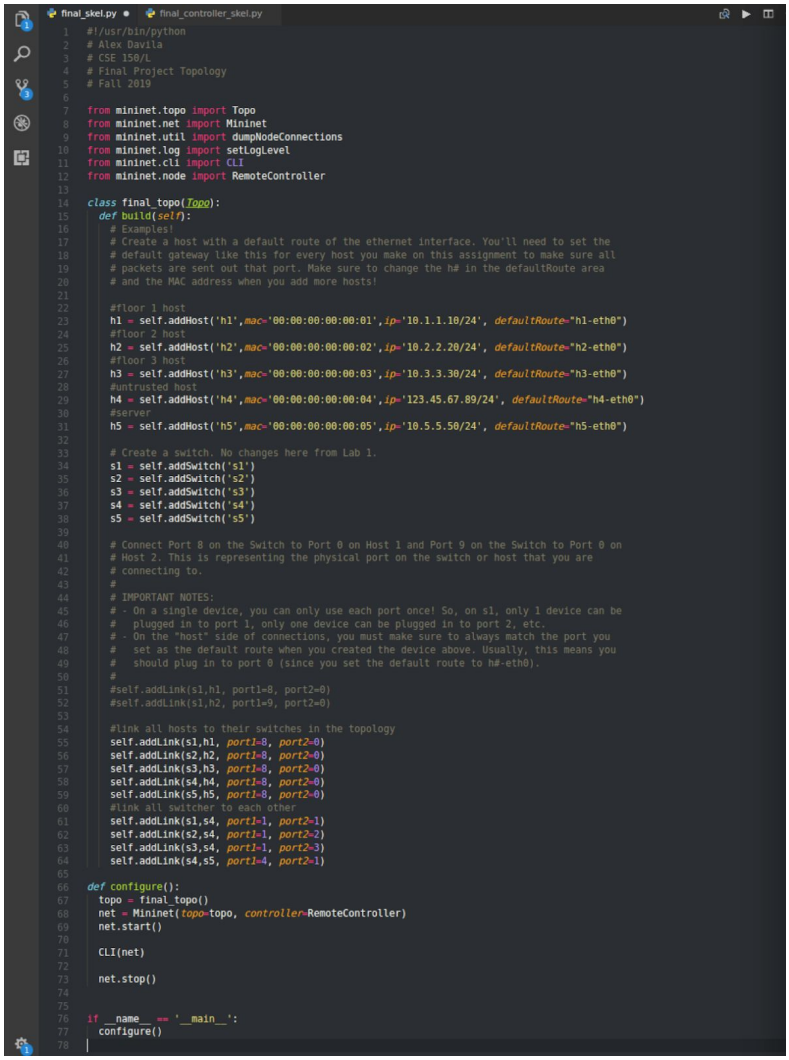
1. To run the controller, I placed *final_controller_skel.py* in the ~/pox/pox/misc directory. I could then launch the controller with the command **sudo ~/pox/pox.py misc final_controller_skel**.
2. To run the mininet file, I placed it in ~ and ran the command **sudo python ~/final_skel.py**

To do this assignment, I ran both files at the same time (in 2 different terminal windows).

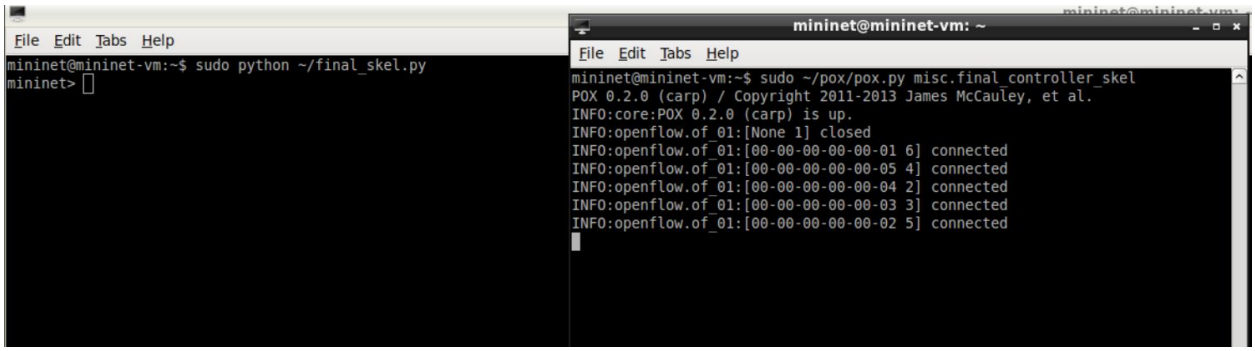
Testing/Submission/Grading

[30 points] Mininet Topology

- 10: Devices successfully created.
- 10: Links successfully created.
- 10: IP addresses correct.



```python
#!/usr/bin/python
# Alex Davila
# CSE 150/L
# Final Project Topology
# Fall 2019

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.node import RemoteController

class final_topo(Topo):
  def build(self):
    # Examples!
    # Create a host with a default route of the ethernet interface. You'll need to set the
    # default gateway like this for every host you make on this assignment to make sure all
    # packets are sent out that port. Make sure to change the h# in the defaultRoute area
    # and the MAC address when you add more hosts!

    #floor 1 host
    h1 = self.addHost('h1',mac='00:00:00:00:00:01',ip='10.1.1.10/24', defaultRoute="h1-eth0")
    #floor 2 host
    h2 = self.addHost('h2',mac='00:00:00:00:00:02',ip='10.2.2.20/24', defaultRoute="h2-eth0")
    #floor 3 host
    h3 = self.addHost('h3',mac='00:00:00:00:00:03',ip='10.3.3.30/24', defaultRoute="h3-eth0")
    #untrusted host
    h4 = self.addHost('h4',mac='00:00:00:00:00:04',ip='123.45.67.89/24', defaultRoute="h4-eth0")
    #server
    h5 = self.addHost('h5',mac='00:00:00:00:00:05',ip='10.5.5.50/24', defaultRoute="h5-eth0")

    # Create a switch. No changes here from Lab 1.
    s1 = self.addSwitch('s1')
    s2 = self.addSwitch('s2')
    s3 = self.addSwitch('s3')
    s4 = self.addSwitch('s4')
    s5 = self.addSwitch('s5')

    # Connect Port 8 on the Switch to Port 0 on Host 1 and Port 9 on the Switch to Port 0 on
    # Host 2. This is representing the physical port on the switch or host that you are
    # connecting to.
    #
    # IMPORTANT NOTES:
    # - On a single device, you can only use each port once! So, on s1, only 1 device can be
    #   plugged in to port 1, only one device can be plugged in to port 2, etc.
    # - On the "host" side of connections, you must make sure to always match the port you
    #   set as the default route when you created the device above. Usually, this means you
    #   should plug in to port 0 (since you set the default route to h#-eth0).
    #
    #self.addLink(s1,h1, port1=8, port2=0)
    #self.addLink(s1,h2, port1=9, port2=0)

    #link all hosts to their switches in the topology
    self.addLink(s1,h1, port1=8, port2=0)
    self.addLink(s2,h2, port1=8, port2=0)
    self.addLink(s3,h3, port1=8, port2=0)
    self.addLink(s4,h4, port1=8, port2=0)
    self.addLink(s5,h5, port1=8, port2=0)
    #link all switcher to each other
    self.addLink(s1,s4, port1=1, port2=1)
    self.addLink(s2,s4, port1=1, port2=2)
    self.addLink(s3,s4, port1=1, port2=3)
    self.addLink(s4,s5, port1=4, port2=1)

def configure():
  topo = final_topo()
  net = Mininet(topo=topo, controller=RemoteController)
  net.start()

  CLI(net)

  net.stop()


if __name__ == '__main__':
  configure()
```



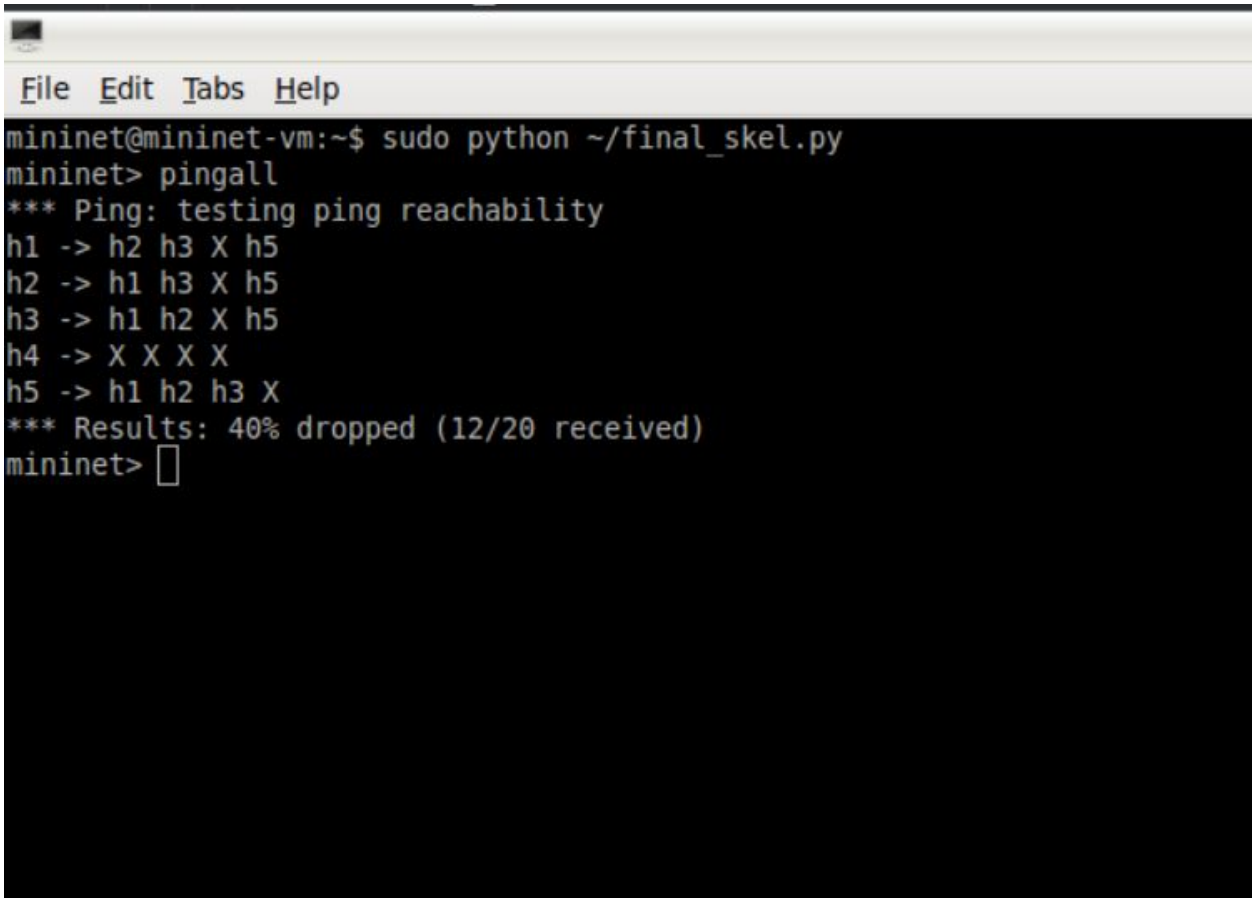
```
File  Edit  Tabs  Help
mininet@mininet-vm:~$ sudo python ~/final_skel.py
mininet>
```

```
mininet@mininet-vm: ~
File  Edit  Tabs  Help
mininet@mininet-vm:~$ sudo ~/pox/pox.py misc.final_controller_skel
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 6] connected
INFO:openflow.of_01:[00-00-00-00-00-05 4] connected
INFO:openflow.of_01:[00-00-00-00-00-04 2] connected
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
INFO:openflow.of_01:[00-00-00-00-00-02 5] connected
```

[50 points] Pox Controller

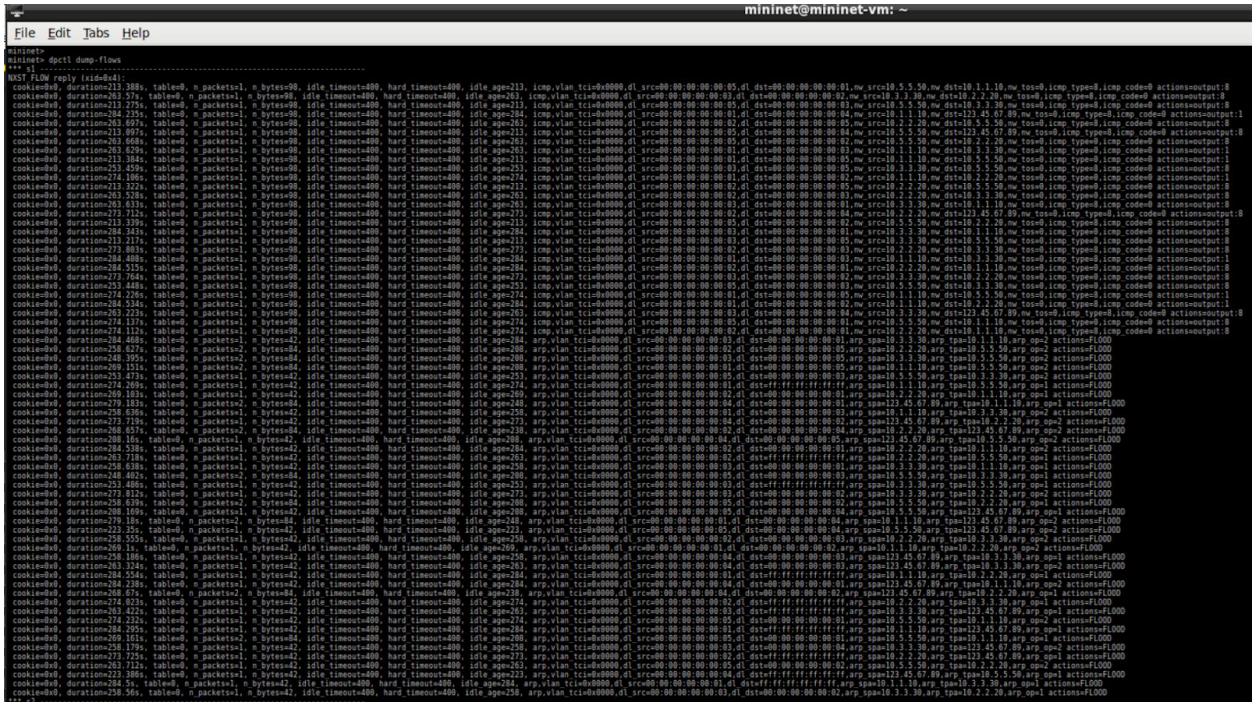
- 25: All hosts can communicate.
  - 15 point deduction if rules not installed in flow table.
  - 20 point deduction if IP traffic is implemented using OFPP_FLOOD.
- 15: Untrusted Host cannot send ICMP traffic to Host 1, Host 2, Host 3, Server
  - 10 point deduction if Untrusted Host cannot send ANY traffic to these hosts.
- 10: Untrusted Host cannot send any IP traffic to Server

Running pingall - shows that all hosts can communicate Except from the Untrusted Host to anywhere in the network. In my code, I blocked ICMP traffic from the untrusted host to anywhere internally hence the 40% of ICMP packets being dropped in the screenshot below.

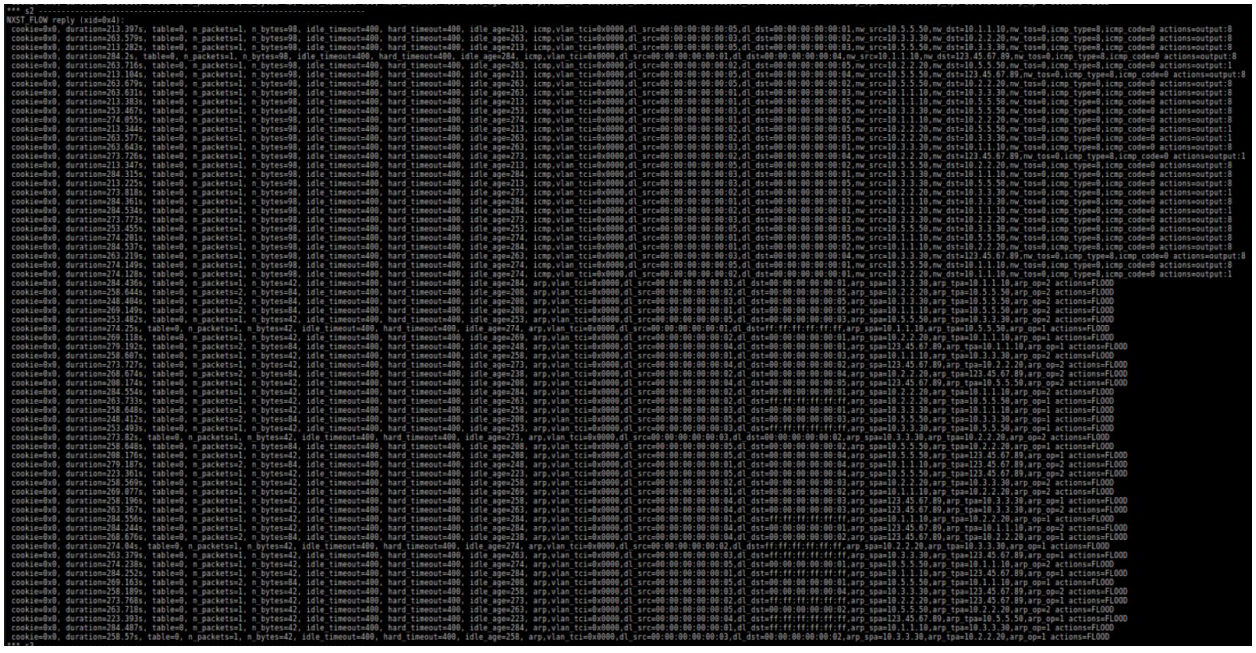


Running dpctl dump-flows: This shows a few entries. These are the entries that I installed into the switch with of_flow_mod. I did this within the timeout specified in the of_flow_mod for the entries to show up! Look at the screenshots below for s1, s2, s3, s4, and s5:

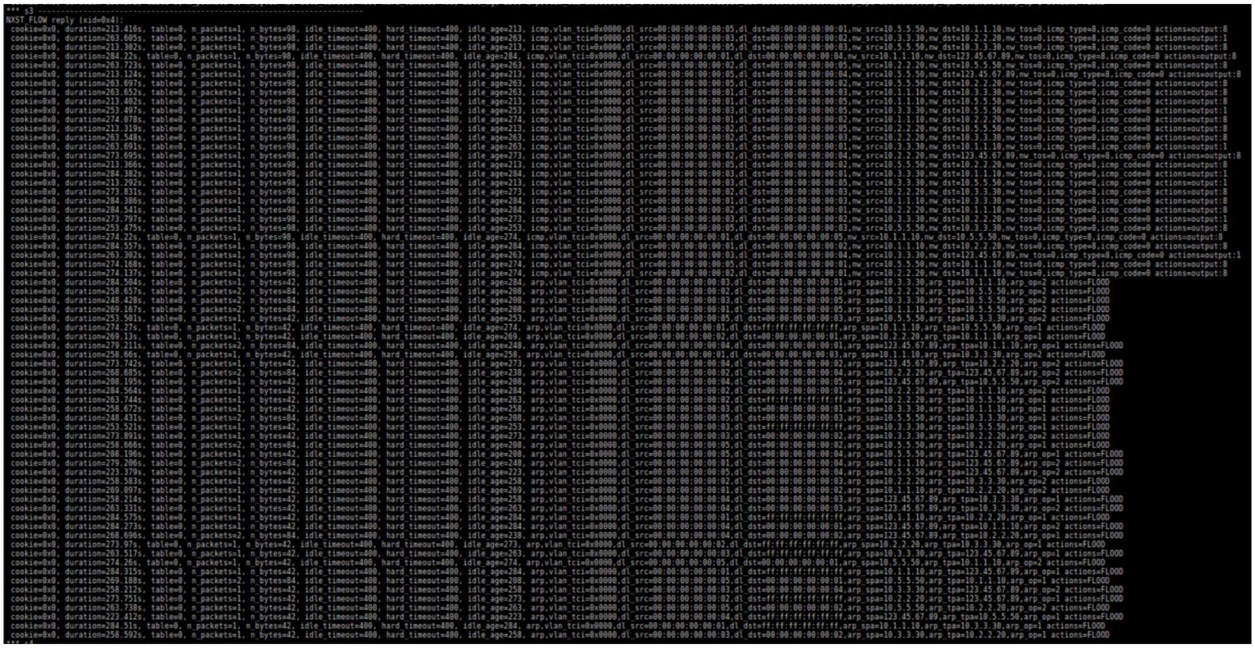
Switch 1



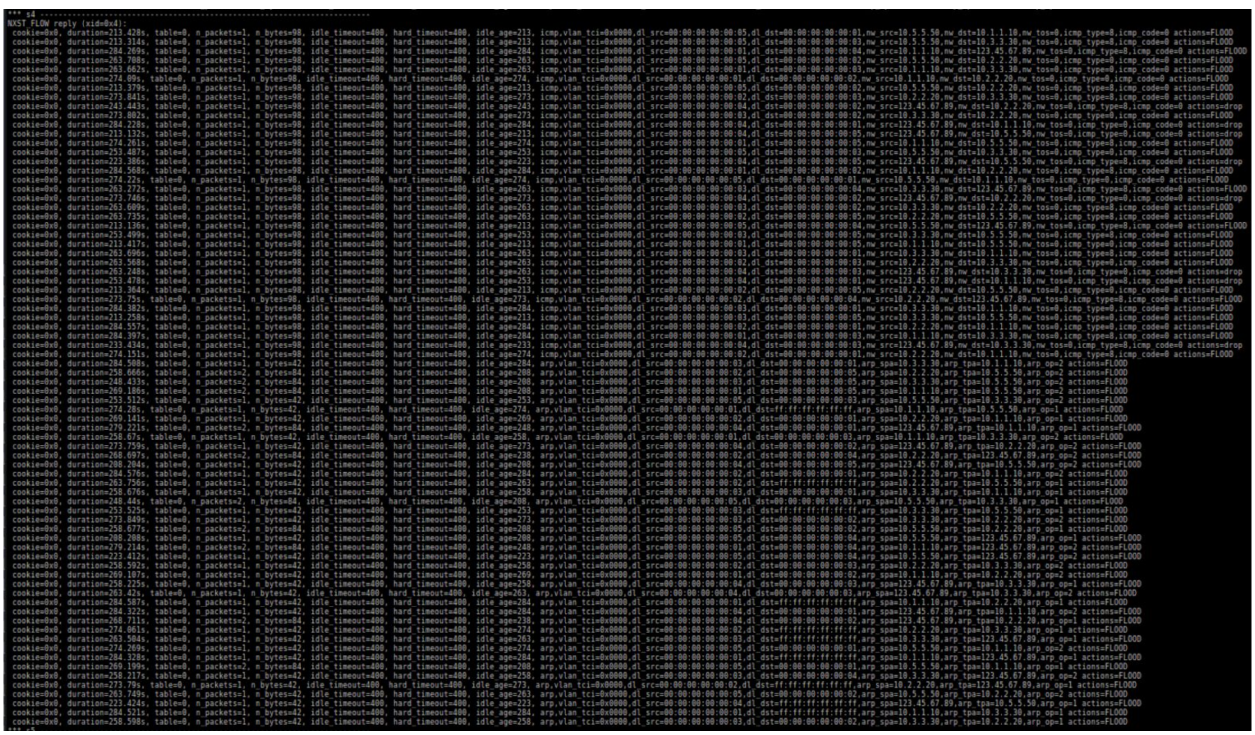
Switch 2



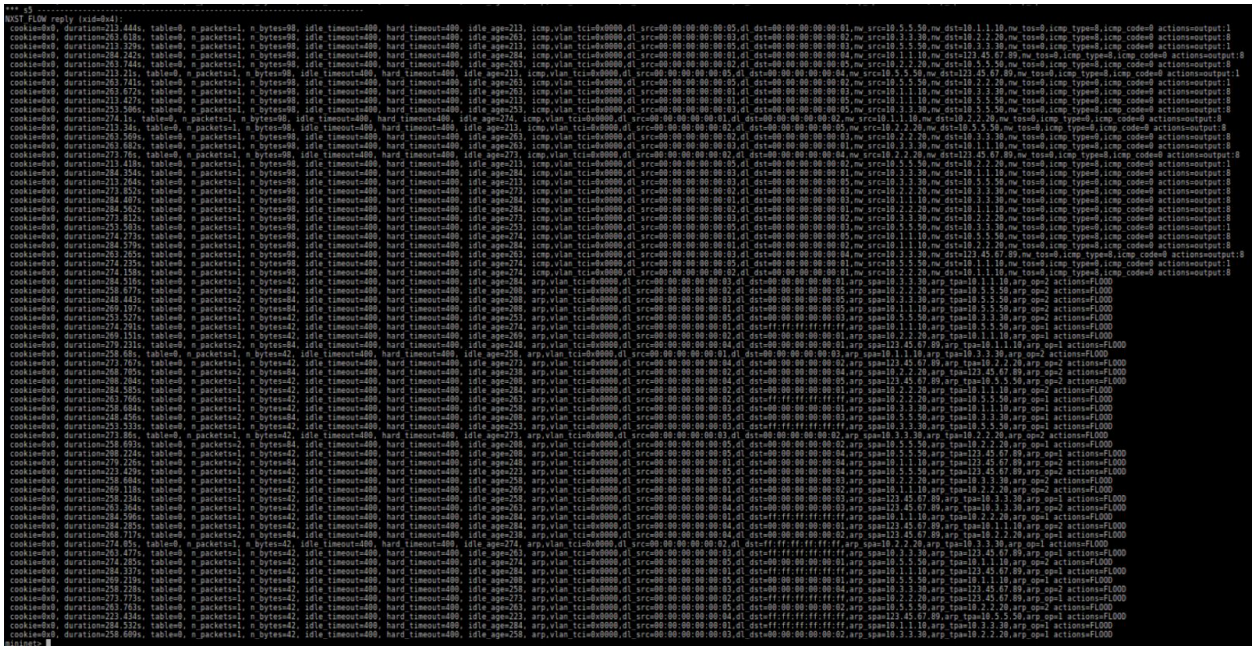
Switch 3



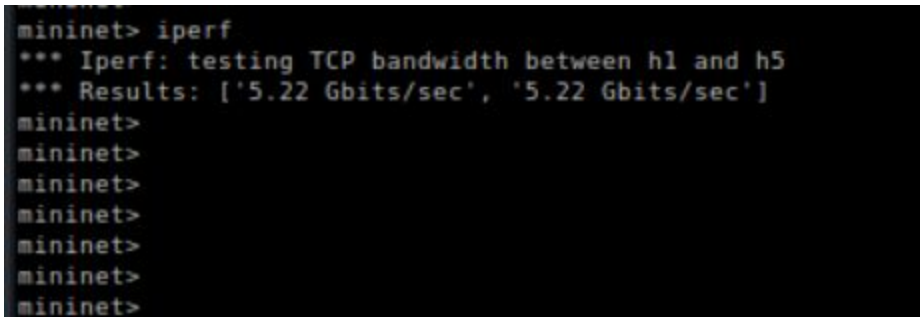
Switch 4



Switch 5



Running iperf : This should succeed.



DEMO: I ran my demo with the TA on Friday Dec 6

- I got working code
- Correct output
- Perfect implementation of my rules

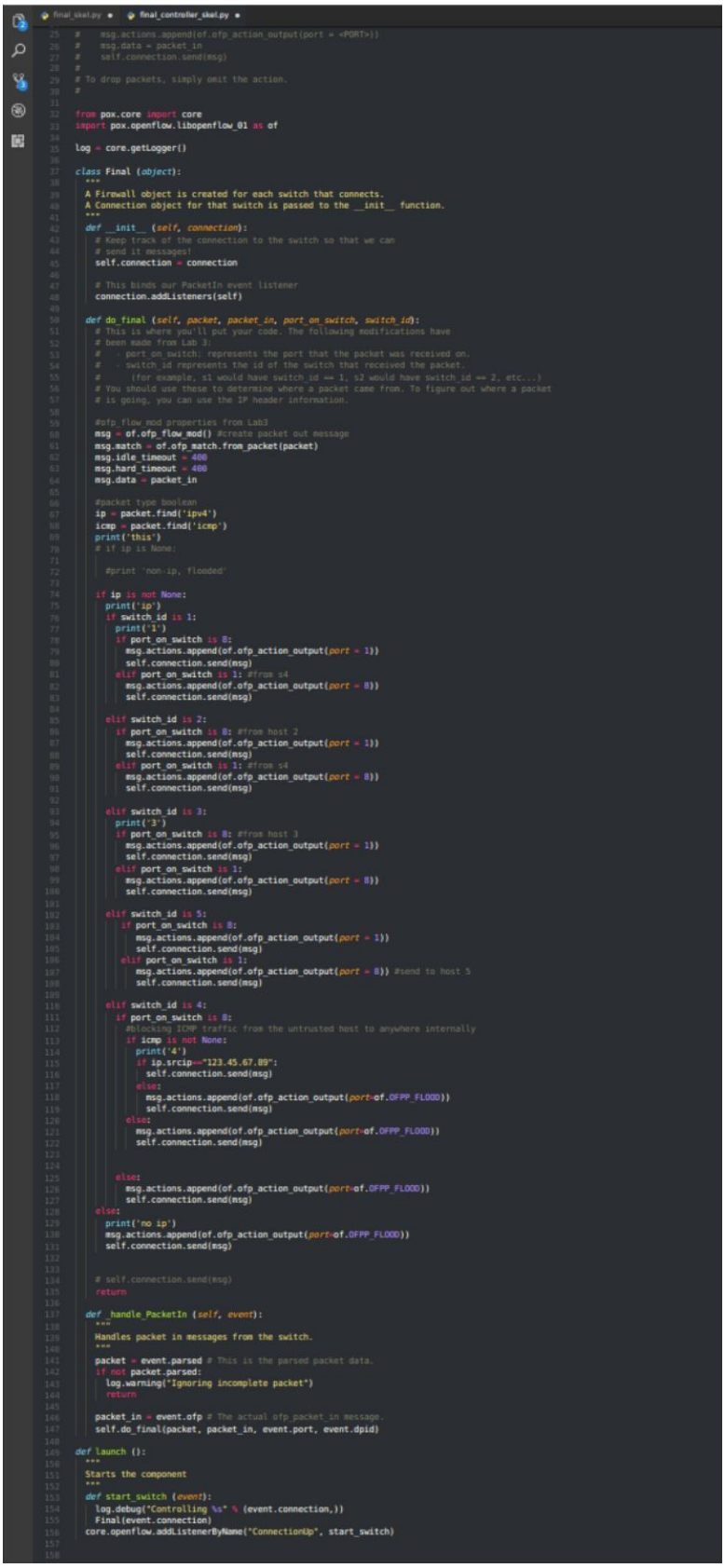
Deliverables:

final_project.pdf: Screenshots of the above commands.

1. **final_controller_skel.py:** Your firewall code
2. **final_skel.py**
3. **README (or README.txt)**: A readme file explaining your submission.

Note: You do not need to submit lab3.py. You SHOULD NOT modify lab3.py.

Screenshot of my code:



```
final_skel.py •  final_controller_skel.py •
#    msg.actions.append(of.ofp_action_output(port = <PORT>))
#    msg.data = packet_in
#    self.connection.send(msg)
#
# To drop packets, simply omit the action.
#

from pox.core import core
import pox.openflow.libopenflow_01 as of

log = core.getLogger()

class Final (object):
  """
  A Firewall object is created for each switch that connects.
  A Connection object for that switch is passed to the __init__ function.
  """
  def __init__ (self, connection):
    # Keep track of the connection to the switch so that we can
    # send it messages!
    self.connection = connection

    # This binds our PacketIn event listener
    connection.addListeners(self)

  def do_final (self, packet, packet_in, port_on_switch, switch_id):
    # This is where you'll put your code. The following modifications have
    # been made from Lab 3:
    #   - port_on_switch: represents the port that the packet was received on.
    #   - switch_id represents the id of the switch that received the packet.
    #      (for example, s1 would have switch_id == 1, s2 would have switch_id == 2, etc...)
    # You should use these to determine where a packet came from. To figure out where a packet
    # is going, you can use the IP header information.

    #sfp_flow_mod properties from Lab3
    msg = of.ofp_flow_mod() #Create packet out message
    msg.match = of.ofp_match.from_packet(packet)
    msg.idle_timeout = 400
    msg.hard_timeout = 400
    msg.data = packet_in

    #packet type boolean
    ip = packet.find('ipv4')
    icmp = packet.find('icmp')
    print('this')
    # if ip is None:

      #print 'non-ip, flooded'

    if ip is not None:
      print('ip')
      if switch_id is 1:
        print('1')
        if port_on_switch is 8:
          msg.actions.append(of.ofp_action_output(port = 1))
          self.connection.send(msg)
        elif port_on_switch is 1: #from s4
          msg.actions.append(of.ofp_action_output(port = 8))
          self.connection.send(msg)

      elif switch_id is 2:
        if port_on_switch is 8: #from host 2
          msg.actions.append(of.ofp_action_output(port = 1))
          self.connection.send(msg)
        elif port_on_switch is 1: #from s4
          msg.actions.append(of.ofp_action_output(port = 8))
          self.connection.send(msg)

      elif switch_id is 3:
        print('3')
        if port_on_switch is 8: #from host 3
          msg.actions.append(of.ofp_action_output(port = 1))
          self.connection.send(msg)
        elif port_on_switch is 1:
          msg.actions.append(of.ofp_action_output(port = 8))
          self.connection.send(msg)

      elif switch_id is 5:
        if port_on_switch is 8:
          msg.actions.append(of.ofp_action_output(port = 1))
          self.connection.send(msg)
        elif port_on_switch is 1:
          msg.actions.append(of.ofp_action_output(port = 8)) #send to host 5
          self.connection.send(msg)

      elif switch_id is 4:
        if port_on_switch is 8:
          #blocking ICMP traffic from the untrusted host to anywhere internally
          if icmp is not None:
            print('4')
            if ip.srcip=="123.45.67.89":
              self.connection.send(msg)
            else:
              msg.actions.append(of.ofp_action_output(port=of.OFPP_FLOOD))
              self.connection.send(msg)
          else:
            msg.actions.append(of.ofp_action_output(port=of.OFPP_FLOOD))
            self.connection.send(msg)


        else:
          msg.actions.append(of.ofp_action_output(port=of.OFPP_FLOOD))
          self.connection.send(msg)
      else:
        print('no ip')
        msg.actions.append(of.ofp_action_output(port=of.OFPP_FLOOD))
        self.connection.send(msg)


      # self.connection.send(msg)
      return

  def _handle_PacketIn (self, event):
    """
    Handles packet in messages from the switch.
    """
    packet = event.parsed # This is the parsed packet data.
    if not packet.parsed:
      log.warning("Ignoring incomplete packet")
      return

    packet_in = event.ofp # The actual ofp_packet_in message.
    self.do_final(packet, packet_in, event.port, event.dpid)

def launch ():
  """
  Starts the component
  """
  def start_switch (event):
    log.debug("Controlling %s" % (event.connection,))
    Final(event.connection)
  core.openflow.addListenerByName("ConnectionUp", start_switch)
```