Prelab3 - Introduction to SDN and OpenFlow
Building a Firewall
100 points

References:

- Firewalls
  - Computer Networking: Section 4.3 “Inspecting Datagrams: Firewalls and Intrusion Detection Systems”
  -
- Mininet:
  - On the Mininet site, the API Reference will be an excellent resource for figuring out how to run pings or open the command prompt in between the `net.start()` and `net.stop()` lines.
  - A Mininet Walkthrough can be found on this page
- POX Controller:
  - https://noxrepo.github./pox-doc/html/#
  - POX WIKI
  - Inside your VM, the `pox/forwarding/l2_learning.py` example file.
- OpenFlow:
  - **Link to OpenFlow Tutorial**: here
  - OpenFlow 1.3 specification
- IP Header
  - Protocol Numbers in IP Header (Protocol Field)

Lab Environment: This lab works on the VM image in the lab(s). We cannot provide support for other environments – some things might work and others not etc. If you have an M1/M2 Mac which does not work properly or are otherwise unable to run the VM, please use the computers in Jack Baskin 109 or Soc Sci to complete the assignment. Also, make sure your scripts are written in Python 2, not Python 3.

This lab assumes you will spend significant time reading the documentation and learning about Mininet, OpenFlow and the POX controller. ***It's important to read the below write up before starting the lab.***

Information on screenshots: For all questions that require a screenshot, make sure that a **date timestamp** is visible next to your results.

For wireshark screenshots, display the date and time by this setting: `"view" ->`

`"Time Display Format" -> Select data and time of the day` Make sure

to highlight as necessary in the screenshots to get full credit.

Submission:

1. **prelab3.pdf**: Answers and screenshots for all the questions. Please use the "date" command to **show a timestamp** in your screenshots. If you are not able to get the expected results, below your screenshot, explain what you think is going on (for partial credit).
2. **`prelab3controller.py`**: Your firewall code.
3. **README.txt**: A readme file explaining your submission.
4. **`prelab3.py`: You do not need to submit `prelab3.py`.** Questions 6-8 do not require `prelab3.py` modification, but Question 10 requires you to modify `prelab3.py`.

Goals of this lab:

This lab builds on the knowledge acquired in Lab 1 where you were first introduced to the Mininet environment and some of Mininet's basic functionality. In this lab, we will take that one step further by introducing you to Software-Defined Networking (SDN) and the OpenFlow protocol. (See tutorial in References.) After familiarizing yourself with these topics you will build your own firewall.

Software-Defined Networking & OpenFlow:

Software-Defined Networking (SDN) is a networking paradigm in which the data and the control planes are decoupled from one another. One can think of the control plane as being the network's "brain," i.e., it is responsible for making all decisions (for example, **how** to forward data), while the data plane is what actually moves the data. In traditional networks, both the control and data planes are tightly integrated and implemented in the forwarding devices (such as the IP Routers).

The SDN control plane is implemented by the "controller" and the data plane by the "switches." The controller acts as the "brain" of the network and sends commands (AKA "rules") to the switches on how to handle traffic (packets). The OpenFlow protocol has emerged as the de facto SDN standard that specifies how the controller and the switches communicate with one another, as well as how the controllers install the rules onto the switches.

Mininet and OpenFlow:

In Lab 1, we experimented with Mininet using its internal controller (and as such did not need to worry about OpenFlow). In this lab (and the next one), we will use our own controller (AKA a remote controller) to send commands to the switches. In the SDN network, the controller can be the default one in Mininet or any one of many different **remote controllers** that are available, such as POX, Ryu, *et. al.* The remote controller we are using is the POX controller and it is written in Python. The commands sent by the POX controller will be written by you!

OpenFlow 1.3 Overview:

OpenFlow 1.3 is the version of the OpenFlow protocol supported within the Mininet environment. The OpenFlow protocol is a standard communication protocol to enable and manage the communication between the controller and switches. As you would expect with any standard protocol, OpenFlow operates irrespective of the vendor of the controller or switches and also different hardware.

The following diagram explains the operation of OpenFlow switches.

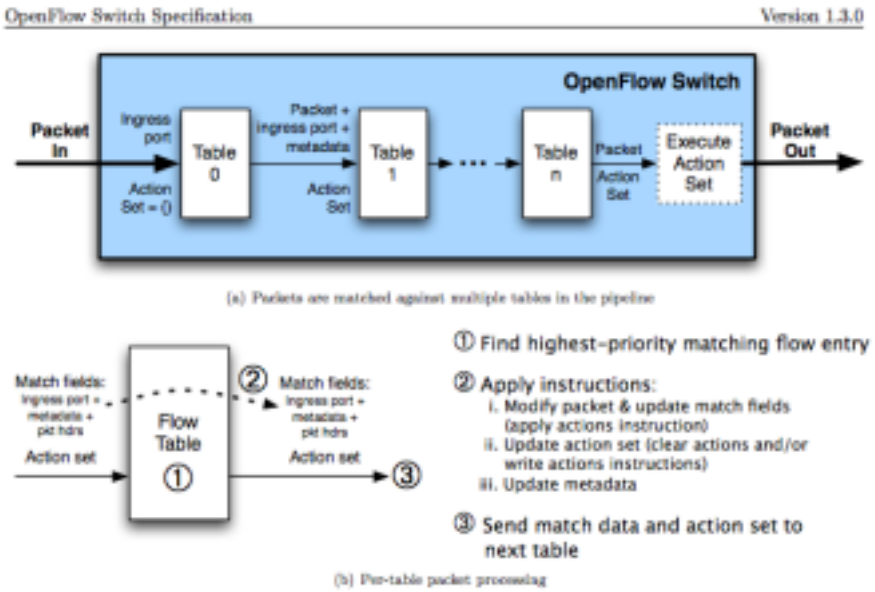


Figure 2: Packet flow through the processing pipeline

Note that when the packet comes into an OpenFlow switch, the switch will reference a table containing “rules” and “actions”. This “flow” table contains the following fields:

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Table 1: Main components of a flow entry in a flow table.

The flowchart below shows the flow of execution as follows:

- If an `ofp_packet_in` does not match any of the flow entries in the flow table and the flow table does not have a “table-miss” flow entry, the packet will be dropped.

- If the packet matches the “table-miss” flow entry, it will be forwarded to the controller.

- If there is a match-entry in the flow table for the packet, the switch will execute the action stored in the instruction field of the corresponding flow table.

5.3 Matching

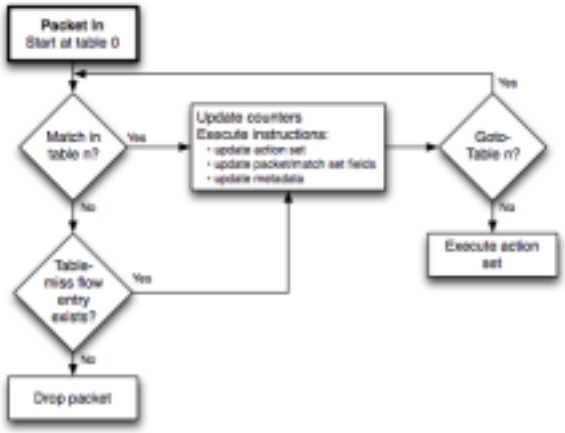


Figure 3: Flowchart detailing packet flow through an OpenFlow switch.

All of the figures and information in this section are from the OpenFlow 1.3 specification, which you can reference for additional information

Assignment

[10 pts] Introductory Questions:

1. **[4 pts] Use your own words to summarize the following and provide your reference from the list of References above or from within this document:**

a) What is Software Defined Networking?

- Software Defined Networking is a networking model in which the data plane and control plane are separated. The control plane is responsible for controlling the decisions, and rules for how to handle data, and the data plane is the instrument which actually moves data through switches. (p.2 in this document)

b) What is Mininet?

- Mininet is a network emulator that creates a network consisting of virtual hosts, switches, controllers, and links. (http://mininet.org/overview)

c) What is OpenFlow?

- OpenFlow is a general protocol that controls the communications between the controller and switches. It operates regardless of the brand of controller, switches, or hardware. (p.3 in this document)

d) What is the POX Controller? How is it different from the Mininet default controller?

- The pox controller provides a programmable environment for developers to

write controller applications using Python. The default controller in Mininet is a simpler version of the Openflow protocol that provides basic functionality for network management, although does not allow for the same level of customization and control as the POX controller. The POX controller is also designed to work with multiple OpenFlow switches and can handle more complex networks. (https://noxrepo.github./pox-doc/html/# , ○ POX WIKI , mininet walkthrough page)

2. [4 pts] In your Mininet environment, the "`sudo mn`" command invokes Mininet, along with its internal controller, whereas

`sudo mn --controller=remote` invokes a remote controller.

a. What command would you use to invoke the remote controller with IP Address `127.0.0.1` and port `6600`?

- **sudo mn -- controller=remote,ip=127.0.0.1,port=6600**

b. What command would you use to verify which controller is being used by Mininet, its IP address and port number?

- **sh ovs-vsctl show**

Take a screenshot and highlight to show your commands and results for (a) and (b).

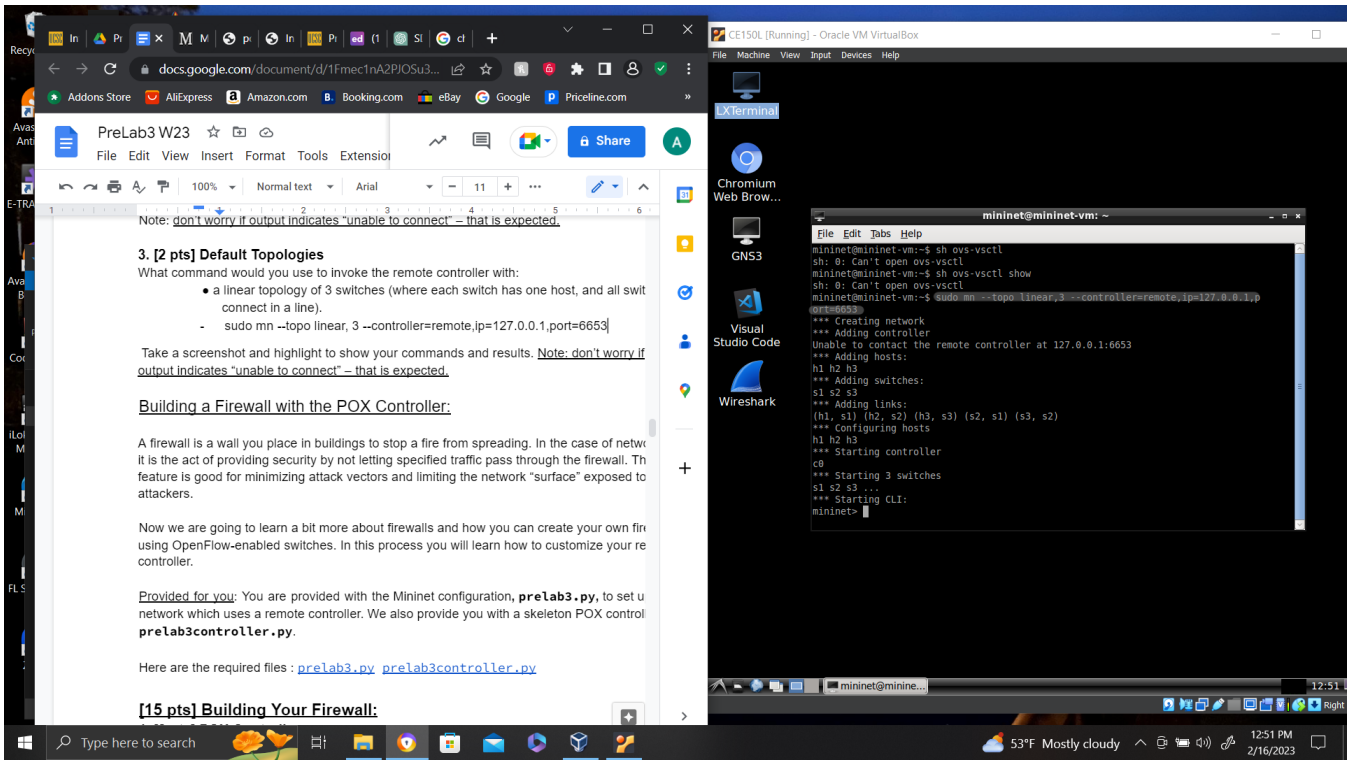
Note: don't worry if output indicates "unable to connect" – that is expected.

3. [2 pts] Default Topologies

What command would you use to invoke the remote controller with:

- a linear topology of 3 switches (where each switch has one host, and all switches connect in a line).

- sudo mn --topo linear, 3 --controller=remote,ip=127.0.0.1,port=6653



Take a screenshot and highlight to show your commands and results. Note: don't worry if output indicates "unable to connect" – that is expected.

Building a Firewall with the POX Controller:

A firewall is a wall you place in buildings to stop a fire from spreading. In the case of networking, it is the act of providing security by not letting specified traffic pass through the firewall. This feature is good for minimizing attack vectors and limiting the network "surface" exposed to attackers.

Now we are going to learn a bit more about firewalls and how you can create your own firewall using OpenFlow-enabled switches. In this process you will learn how to customize your remote controller.

Provided for you: You are provided with the Mininet configuration, **`prelab3.py`**, to set up a network which uses a remote controller. We also provide you with a skeleton POX controller, **`prelab3controller.py`**.

Here are the required files : prelab3.py prelab3controller.py

[15 pts] Building Your Firewall:

4. [2 pts] POX Controller

Why are we using the POX controller (instead of the default controller) to implement the firewall? What is the default IP Address and port used by the remote controller? (Hint: see Mininet Walkthrough.)

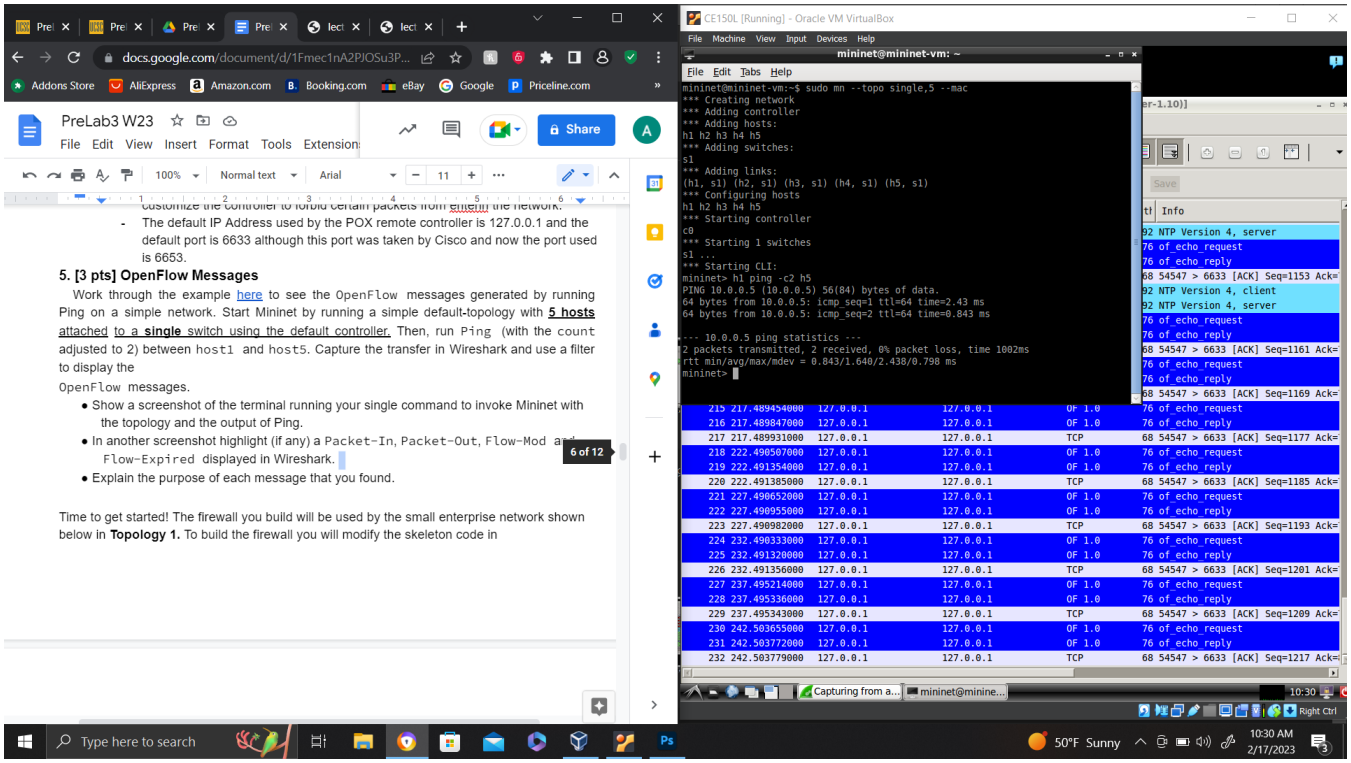
- We are using the POX controller to implement the firewall so that we can customize the controller to forbid certain packets from enterin the network.
- The default IP Address used by the POX remote controller is 127.0.0.1 and the default port is 6633 although this port was taken by Cisco and now the port used is 6653.

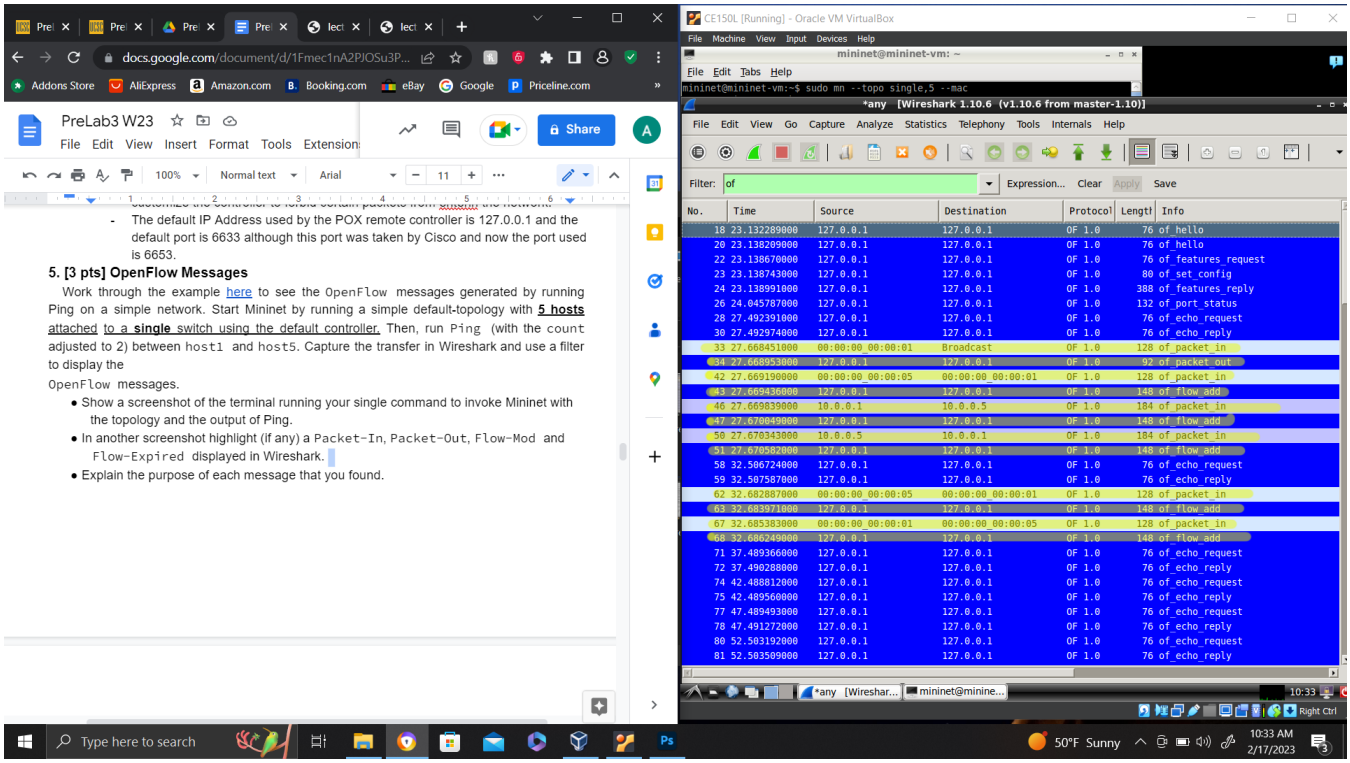
5. [3 pts] OpenFlow Messages

Work through the example here to see the `OpenFlow` messages generated by running Ping on a simple network. Start Mininet by running a simple default-topology with **5 hosts** attached to a **single** switch using the default controller. Then, run `Ping` (with the `count` adjusted to 2) between `host1` and `host5`. Capture the transfer in Wireshark and use a filter to display the `OpenFlow` messages.

- Show a screenshot of the terminal running your single command to invoke Mininet with the topology and the output of Ping.
- In another screenshot highlight (if any) a `Packet-In`, `Packet-Out`, `Flow-Mod` and `Flow-Expired` displayed in Wireshark.
- Explain the purpose of each message that you found.

- The of_packet_in message is an OpenFlow message that is used to send a copy of a packet to the OpenFlow controller to be processed. When a switch receives a packet that does not match present flow table entries, this type of packet is generated with instructions and information for the controller.
- the of_packet_out message is used by an OpenFlow controller to send a packet to a switch to be forwarded. The message contains the packet and instructions on where to send the packet, how the headers should be modified, and sending the packet to a specific port.
- The of_flow_add is used by an OpenFlow controller to add a flow entry to an OpenFlow switch flow table.





Time to get started! The firewall you build will be used by the small enterprise network shown

below in **Topology 1.** To build the firewall you will modify the skeleton code in `prelab3controller.py`. The basic rules that you will need to implement in `OpenFlow` are given below in **Table 1.** Remember, this topology has already been created for you and is in `prelab3.py`. You will use the POX controller.

Requirement and Guidance:

When you create a rule in the POX controller, you need to have POX do the following: ● "install" the rule in the switch so that the switch "remembers" what to do for **30 seconds.** You will be downgraded if your switch simply asks the controller what to do for every packet it receives. Hint: To do this, look up `ofp_flow_mod` in the POX WIKI ● Packet flooding is allowed in the switches.

Running the POX Controller:

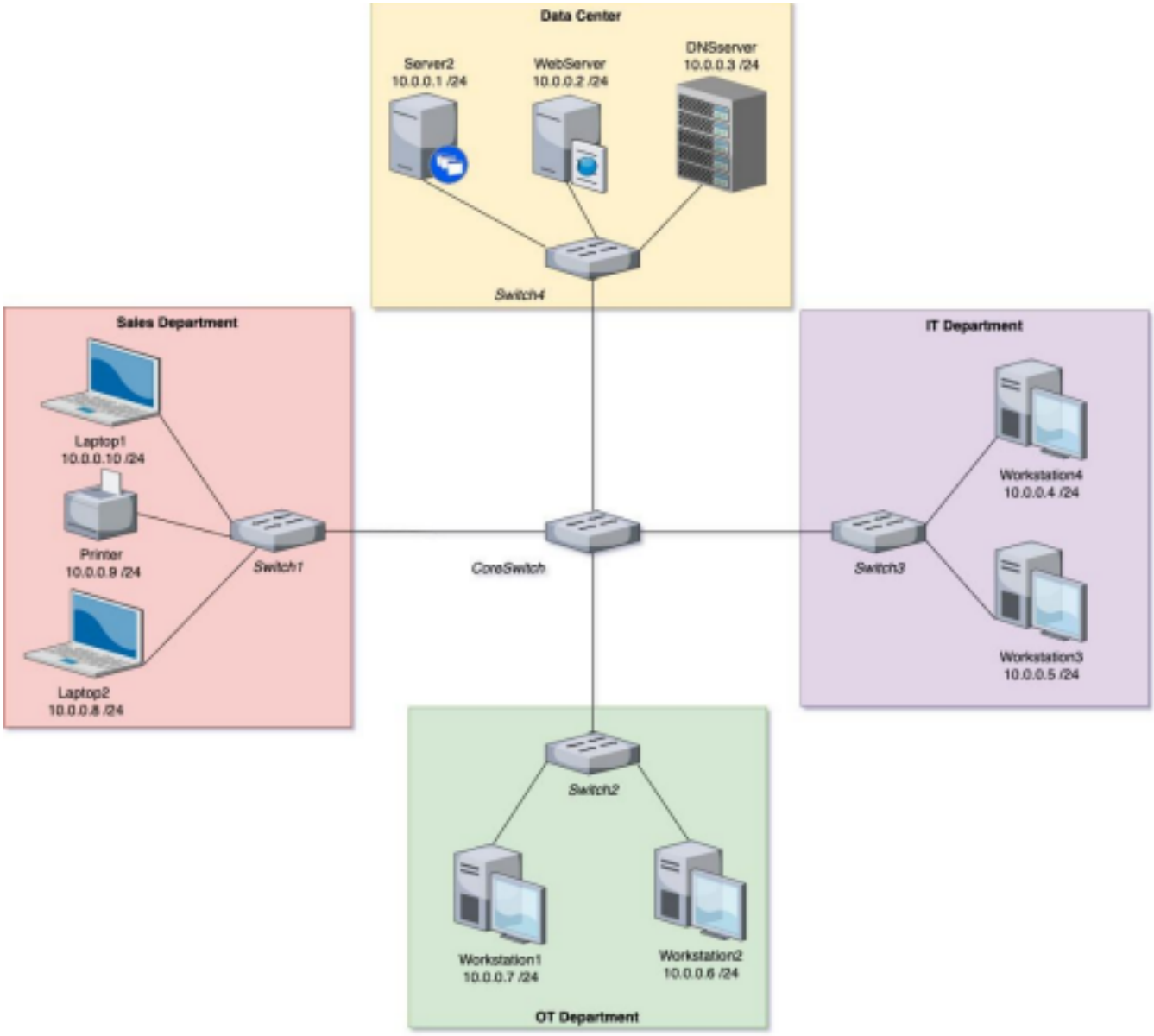
Place your firewall code in `prelab3controller.py` into the `~/pox/pox/misc` directory. You can then launch the controller with the command:

```
sudo ~/pox/pox.py misc.prelab3controller
```

To run the mininet file, place it in `~` and run the command **`sudo python ~/prelab3.py`**

In 2 different terminal windows: **first** start the controller file and **then** run the mininet file. (It should make sense to you why the order matters.)

Note: To do this assignment, both files must be running at the same time.



Topology 1

The rules that you need to implement in OpenFlow for a Basic Firewall are given below in **Table 1:**

Rule #	SRC IP	DST IP	Protocol	Action
1	Any	Any	ARP	accept
2	Any IPv4	Any IPv4	ICMP	accept
3	10.0.0.6	10.0.0.7	TCP	accept
4	10.0.0.7	10.0.0.6	TCP	accept

5	10.0.0.4	10.0.0.10	TCP	accept
6	10.0.0.10	10.0.0.4	TCP	accept
7	10.0.0.6	10.0.0.1	TCP	accept
8	10.0.0.1	10.0.0.6	TCP	accept
9	any	any	-	drop

Table 1- Basic Firewall

6. [7 pts] Basic Firewall Rules (Table 1):

a. Briefly explain the meaning for Rules 1, 3, 4 and 9. A sentence or two for each rule is sufficient.

- 1. This rule indicates that the firewall will allow any traffic into and out of the network that is going to, or from, any IP Address if the traffic is running on ARP protocol.
- 3. This rule indicates the firewall will allow traffic running on the TCP protocol, to be transmitted out of the host at IP Address 10.0.0.7, and to be accepted into the host at IP Address 10.0.0.6.
- 4. This rule indicates that the firewall will allow traffic running on the TCP protocol, to be transmitted out of the host at IP Address 10.0.0.6, and to be accepted into the host at IP Address 10.0.0.7.
- 9. This rule indicates the firewall will drop any traffic sent through the firewall that does not comply with the other eight rules.

b. If you run `ping` between *Laptop2* and *WebServer*, do you expect it to work? Explain your answer based on the rules in Table 1.

- Yes because the rules of the firewall allow ICMP requests between any of the hosts on the network.

c. If you run `iperf` between *Workstation1* and *Server2*, do you expect it to work? Explain your answer based on the rules in Table 1.

- I do expect it to work because iperf supports the IPv4 protocol which is allowed between all hosts in the network, in the rules of the firewall.

7. [3 pts] Write pseudocode for your Firewall. (Use the host names instead of the IP addresses in your pseudocode. e.g., use `Laptop1` rather than `10.0.0.10`.) Check `l2_learning.py` to identify key functions that might be useful in your firewall.

If packet is ARP:

    accept(packet)

If packet is ICMP and Ipv4 is not none:

    accept(packet)

If packet is TCP:

    If source ip address is Workstation 1 or Workstation 2, and destination address is Workstation 1 or Workstation 2:

        accept(packet)

    Elif source ip address is Workstation 4 or Laptop 1, and destination address is Laptop 1 or Workstation 4:

        accept(packet)

    Elif source ip address is Workstation 2 or Server 2, and destination address is Workstation 2 or Server 2:

        accept(packet)

    Else:

        drop(packet)

Else:

    drop(packet)

How to Get Started:

- Read documentation / resources as needed
- Study the file `/forwarding/l2_learning.py`

[15 pts] Testing the Basic Firewall:

8. Protocol Testing

After implementing the rules in **Table 1** it is time to test your basic firewall!

Once the files are started (see **Running the POX Controller** above) and you are prompted with the Mininet CLI, run the following commands:

a. [5 pts] pingallfull:

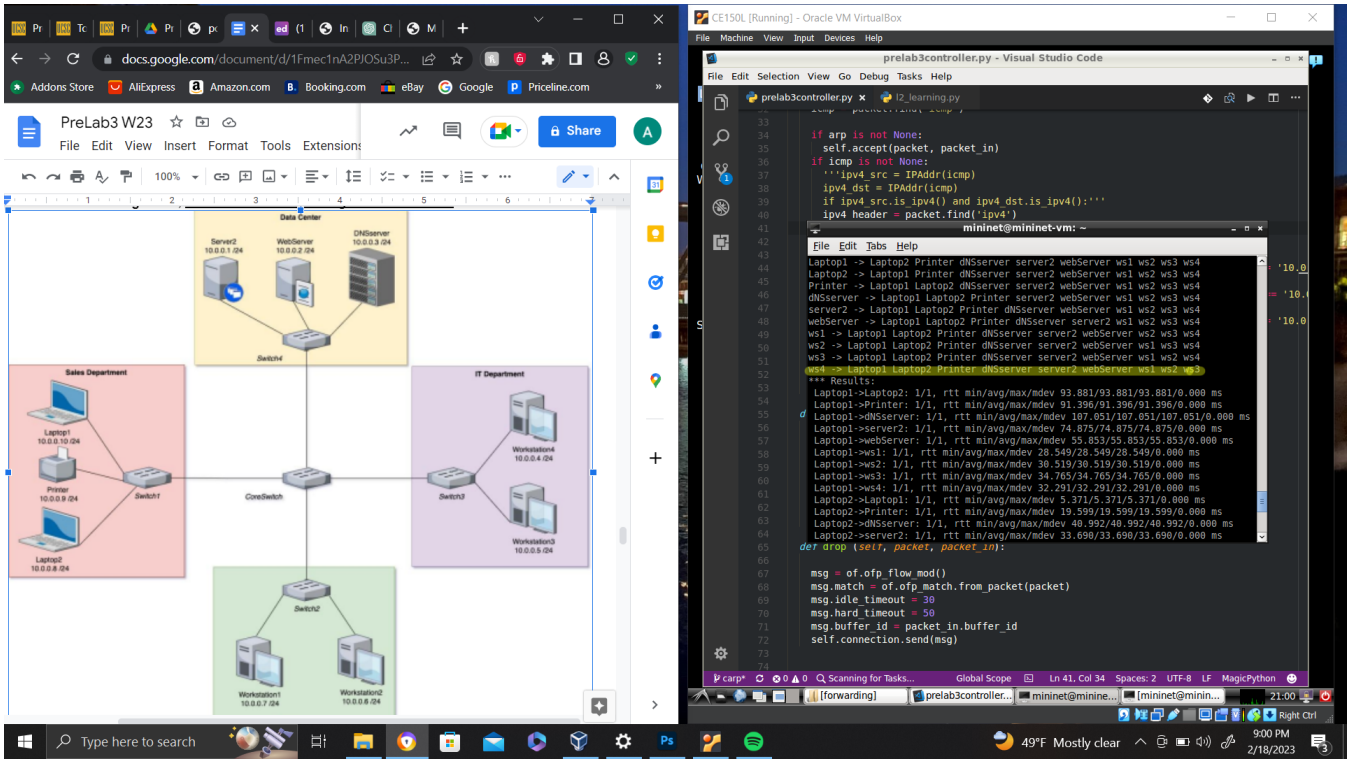
i. Describe the expected result of running this command.

- I expect the pingall to connect each host to each other.

ii. In few sentences explain which of the firewall rules contributed to the result.

- The firewall rule which contributed to this was the acceptance of ICMP packets. As a result of ICMP packets being accepted each of the hosts in the network was able to ping each other host in the network as they are all IPv4 IP addresses. To check this I checked if the packet was an ICMP packet and then I checked if there was an ipv4 header in the packet.

iii. Provide a screenshot of the results and highlight *Workstation4 → Printer*



b. [5 pts] iperf:

Let's test the TCP connection of your basic firewall. For this, use the iperf command (see man page) to simulate TCP traffic and fill out Table 2. Are your results what you expect? Why or why not?

- This is what I expected because under the rules of the firewall Workstation2 is allowed to send TCP packets to Server2, and Laptop2 is allowed to send TCP packets to workstation 3. All of the other links are not allowed to send TCP packets to one another under the firewall rules.

#	Link	Iperf command Pass or Fail?
1	*Workstation2 - Server2*	`Iperf ws2 server2 pass`

2	*Workstation1 - WebServer*	Fail
3	*Workstation2 - DNSserver*	Fail
4	*Laptop1 - Workstation4*	pass
5	*Laptop2 - Printer*	*Fail*
6	*Laptop2 - Workstation 3*	*Fail*
7	*Laptop1 - Printer*	*Fail*

Table 2 - Iperf testing table

c. [5 pts] Traceroute:D

Let's test `ICMP` and `UDP` connections of your firewall using Traceroute commands. Ensure that you force Traceroute probes with the protocol specified in column 1 from the below table. Is the output what you expected based on the rules of Table 1?

Protocol	Link Traceroute Command	Pass or Fail?
ICMP	*Workstation1 - DNSserver*	*Pass*
UDP	*Workstation1 - DNSserver*	*Fail*

- The ICMP ping passed and when and the UDP test failed. This is what I would accept as the firewall allows Icmp and currently drops all other packets not stated in the table.

[30 pts] Adding your own Firewall rules:

9. [8 pts] DNS Server

Ping flood is a common Denial of Service (DoS) attack in which an attacker takes down a DNS server by overwhelming it with Ping requests. To protect your DNS server from the Ping flood attack, add a Firewall rule for the DNS Server. Additionally, add any rules required to ensure that the DNS server can accept queries against its database.

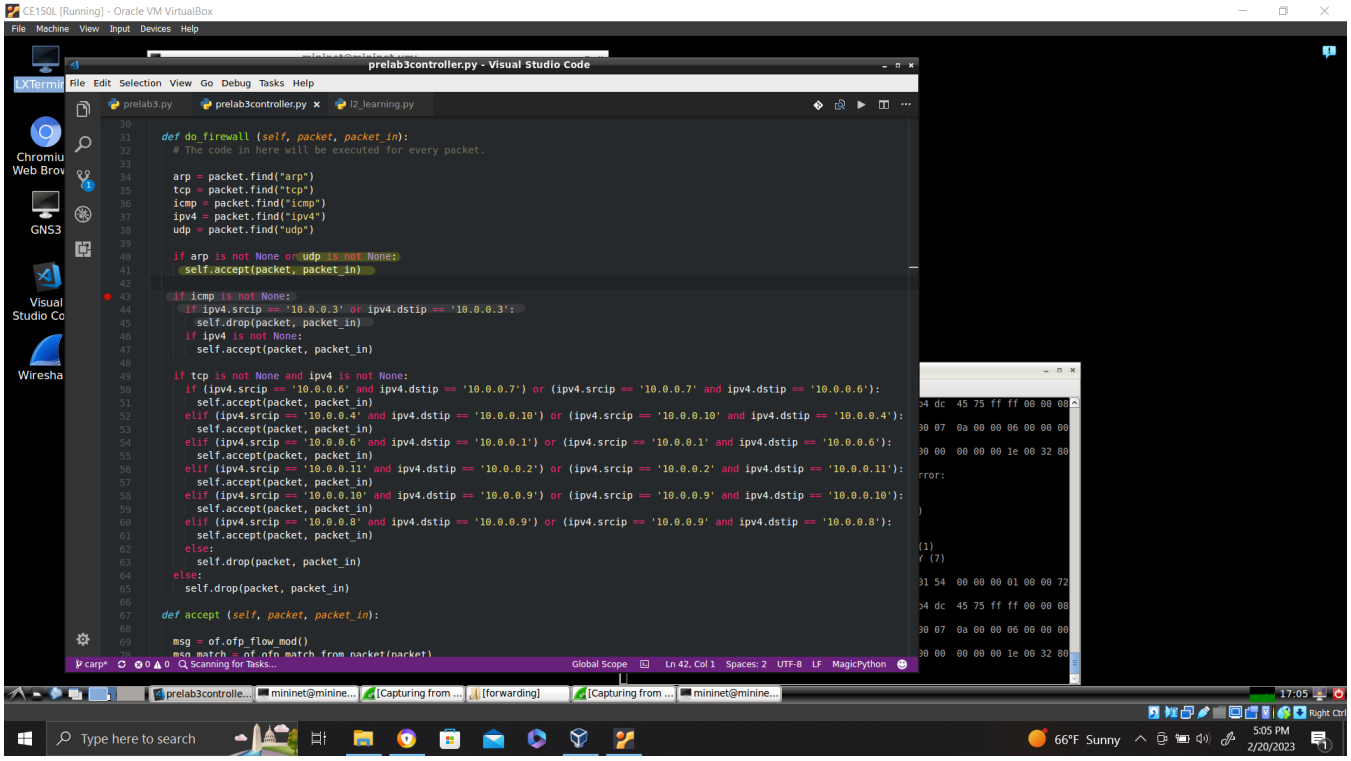
a) State the rules you have implemented and explain your reasoning

- To protect the DNS server from Ping flood I have implemented a rule to drop icmp packets that attempt to ping the DNS server. Also to ensure that the DNS server can accept queries against its database I have implemented a rule that will allow the DNS server to accept and respond to UDP queries.

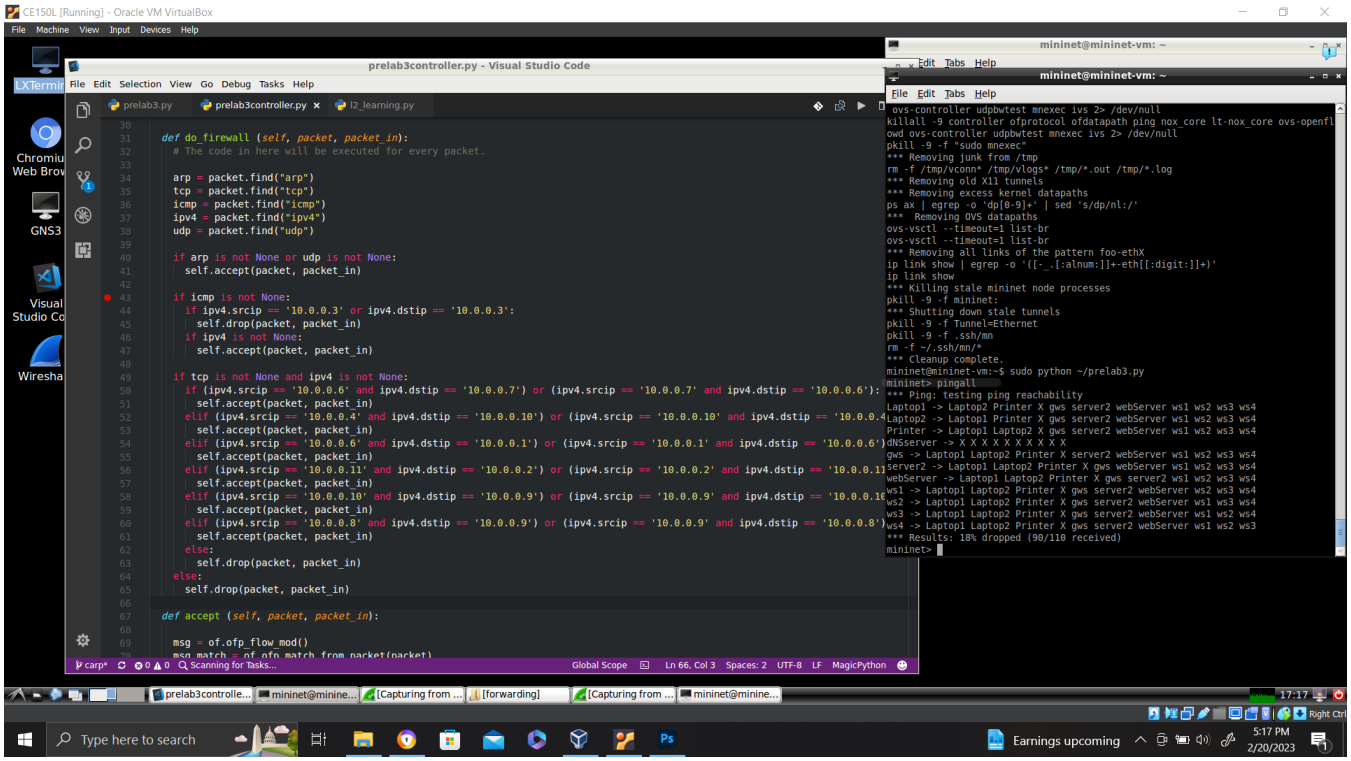
b) Explain your test plan

- First I plan to use the pingall command to verify that no other hosts can ping the DNS server. Then to test the UDP connection I plan to use wireshark to see the behavior of UDP packets on the network to test the connections of the DNS server with a couple hosts as they rule should work the same for any host.

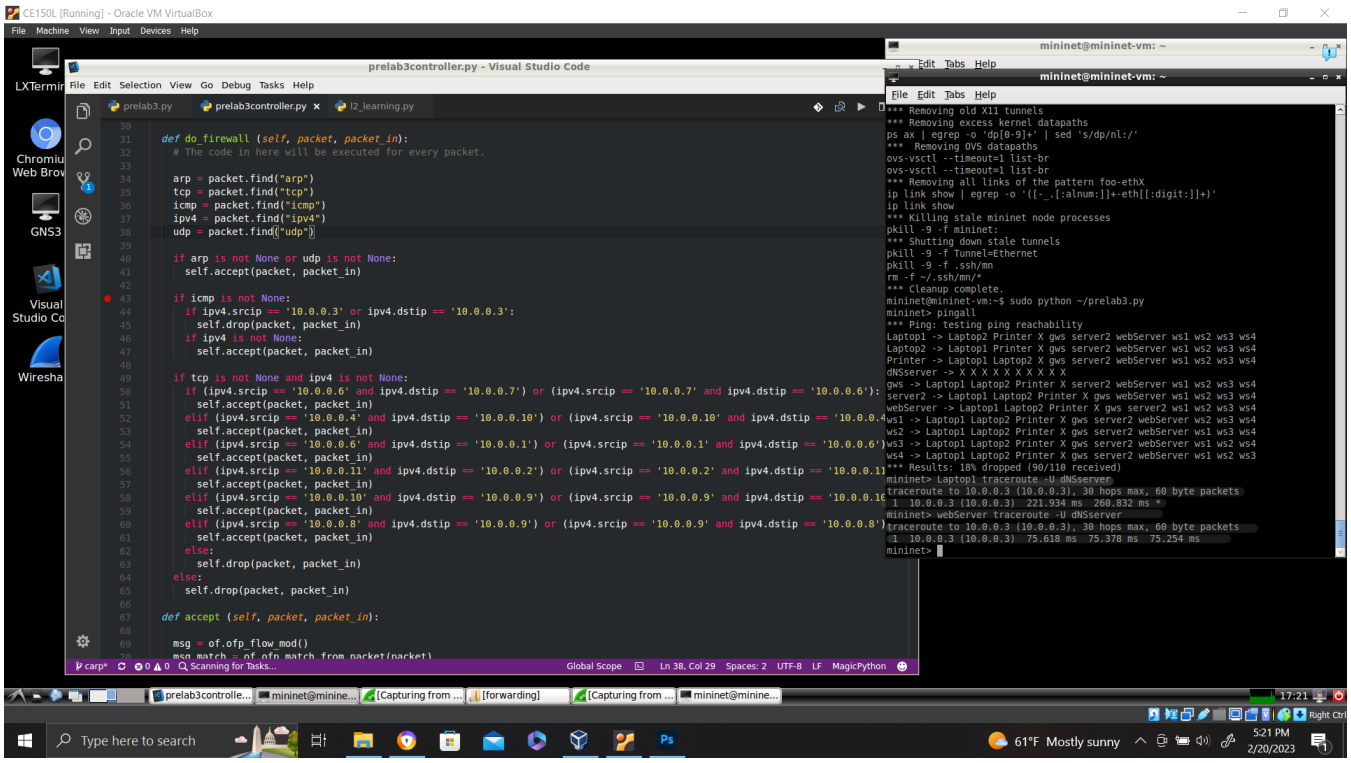
c) Run your test(s) and verify with screenshots and explanation that your rules are working.



Here is the code implementation of the rules for the DNS server. The UDP rules are highlighted in yellow and the rules for the ping requests are highlighted in pink.



In the screenshot above the test to ensure that the DNS server cannot accept pings is shown through the pingall command which tests the connectivity between all hosts through pings.



The screenshot above ensures that the DNS server can accept UDP queries. This is shown through two UDP traceroute commands from two different hosts to the DNS server, highlighted in pink.

10. [10 pts] Guest Access

The company has a guest network through which guests in the network can access the **Web Server**. Change your topology to add a new **Guest** Workstation to the IT department and assign it the IP address `10.0.0.11/24`. Add Firewall rules for guests to access the **Web Server** to make `HTTP` requests.

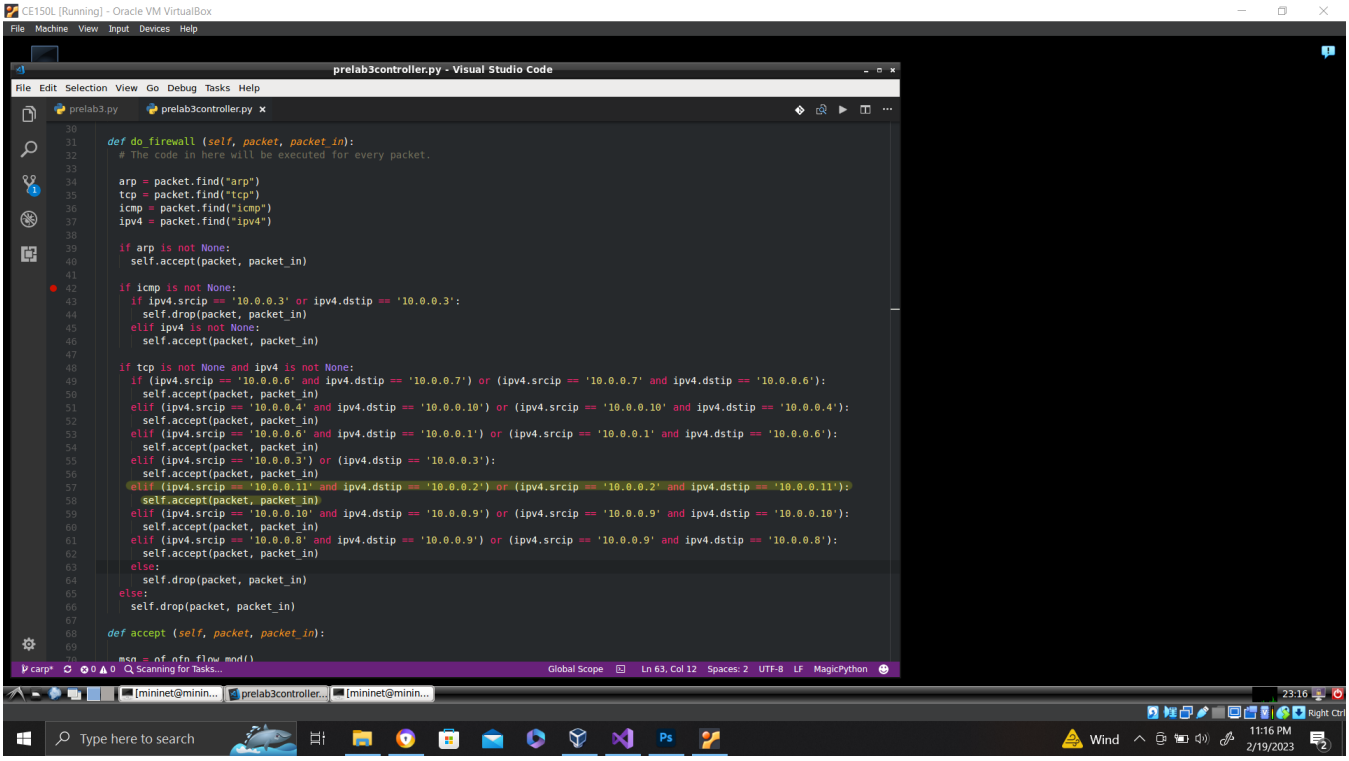
a. State the rules you have implemented and explain your reasoning

- I changed the topology so that the Guest Workstation is a part of the IT Department by adding a link between the Guest Workstation and switch 4. I also implemented a rule in my firewall to allow TCP from Guest Workstation to Web Server, since HTTP requests use TCP protocol.

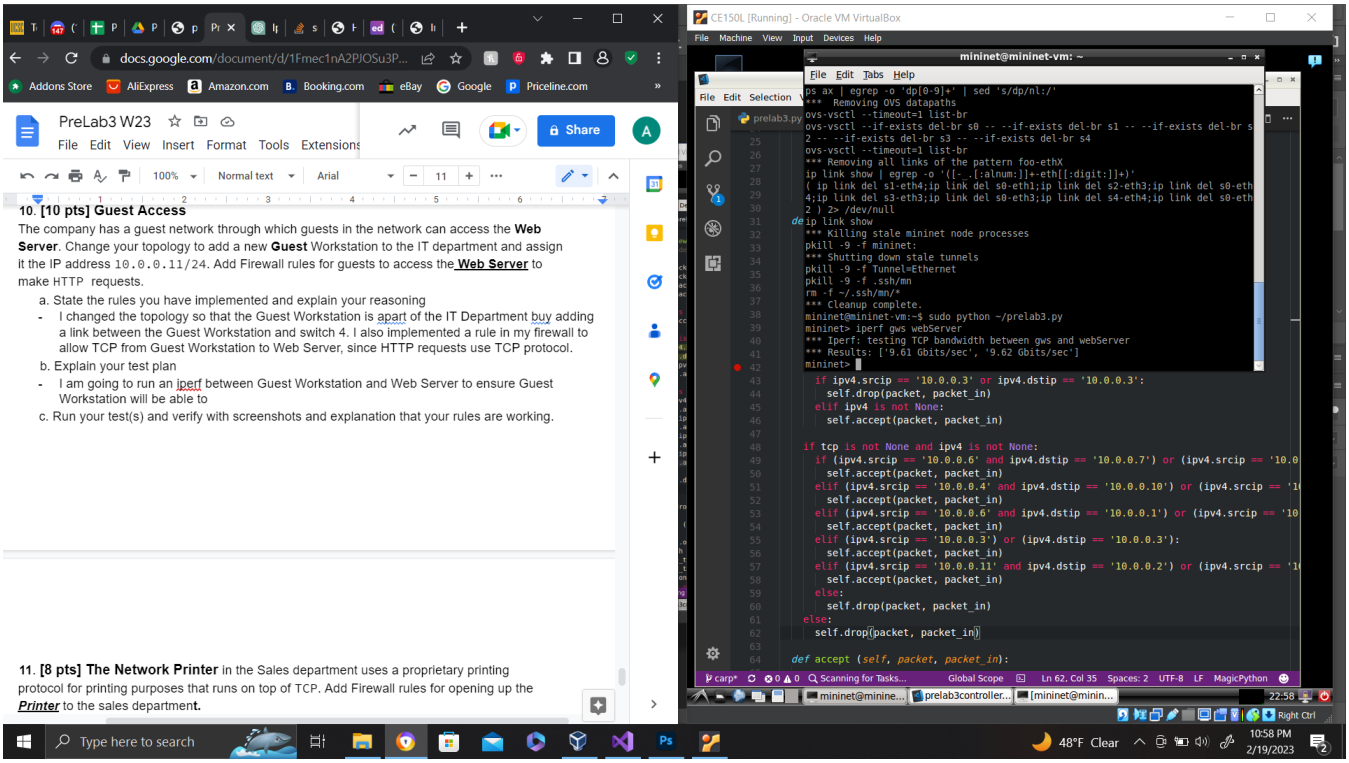
b. Explain your test plan

- I am going to run an iperf between Guest Workstation and Web Server to ensure Guest Workstation will be able to make HTTP requests to the Web Server.

c. Run your test(s) and verify with screenshots and explanation that your rules are working.



Here you can see that I have allowed Guest Workstation with IP Address 10.0.0.11 to send TCP traffic, and thus HTTP request, with Web Server, IP Address 10.0.0.2.



Above you can see the results of the iperf command affirming that the data was transferred in the

Results stating the data transfer rate.

11. **[8 pts] The Network Printer** in the Sales department uses a proprietary printing protocol for printing purposes that runs on top of TCP. Add Firewall rules for opening up the ***Printer*** to the sales departmen**t.**

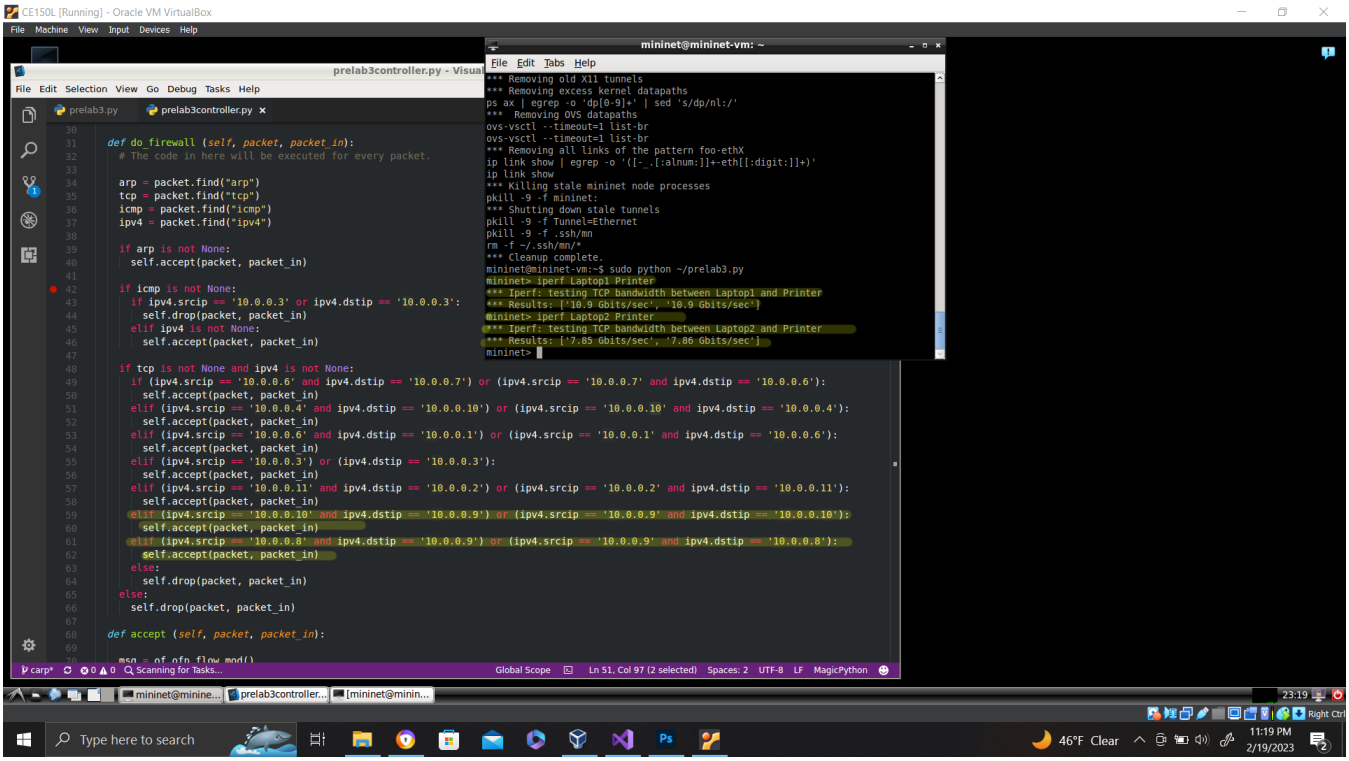
a) State the rules you have implemented and explain your reasoning

- I have allowed Laptop 1 and Laptop 2 to send TCP traffic to and from the printe so that they can use their proprietary printing protocol.

b) How can it be tested?

- This can be tested by using iperf between Laptop 1, and Printer, and iperf between Laptop 2, and Printer.

c) Run your test(s) and verify with screenshots and explanation that your rules are working.



Here you can see the code I have written to allow Laptop1, IP Address 10.0.0.10, and Laptop2, IP Address 10.0.0.8, to send TCP traffic to Printer, IP Address 10.0.0.9. Also visible and highlighted are the passing iperf TCP connection tests, as shown by the data rates, between Laptop1 and Printer, and Laptop2 and Printer.

12. **[4 pts]** Add your new rules to Table 1 and show the “new” Table 1. (Note that the order of the rules matter)

“new “ Table 1

Rule #	SRC IP	DST IP	Protocol	Action
1	Any	Any	ARP	accept
2	Any IPv4	Any IPv4	ICMP	accept
3	10.0.0.6	10.0.0.7	TCP	accept
4	10.0.0.7	10.0.0.6	TCP	accept
5	10.0.0.4	10.0.0.10	TCP	accept
6	10.0.0.10	10.0.0.4	TCP	accept
7	10.0.0.6	10.0.0.1	TCP	accept
8	10.0.0.1	10.0.0.6	TCP	accept
9	any	any	-	drop
10	any	any	UDP	accept
11	10.0.0.11	10.0.0.2	TCP	accept
12	10.0.0.2	10.0.0.11	TCP	accept
13	10.0.0.10	10.0.0.9	TCP	accept
14	10.0.0.9	10.0.0.10	TCP	accept

15	10.0.0.8	10.0.0.9	TCP	accept
16	10.0.0.9	10.0.0.8	TCP	accept

[30 pts] Testing the Completed Firewall: Let's check out your full Firewall!

13. [9 pts] Random Commands

Perform the following commands in mininet and take a screenshot of the results. Use the screenshot to explain the reasoning for the output of each command.

a. `ws1 ping ws2 -c 10`

- Passes because pings are accepted to and from all hosts that are not the DNS server, in the rules of the firewall.

b. `Laptop1 ping ws4 -c 10`

- Again, passes because pings are accepted to and from all hosts that are not the DNS server, in the rules of the firewall.

c. `ws3 ping dNSserver -c 10`

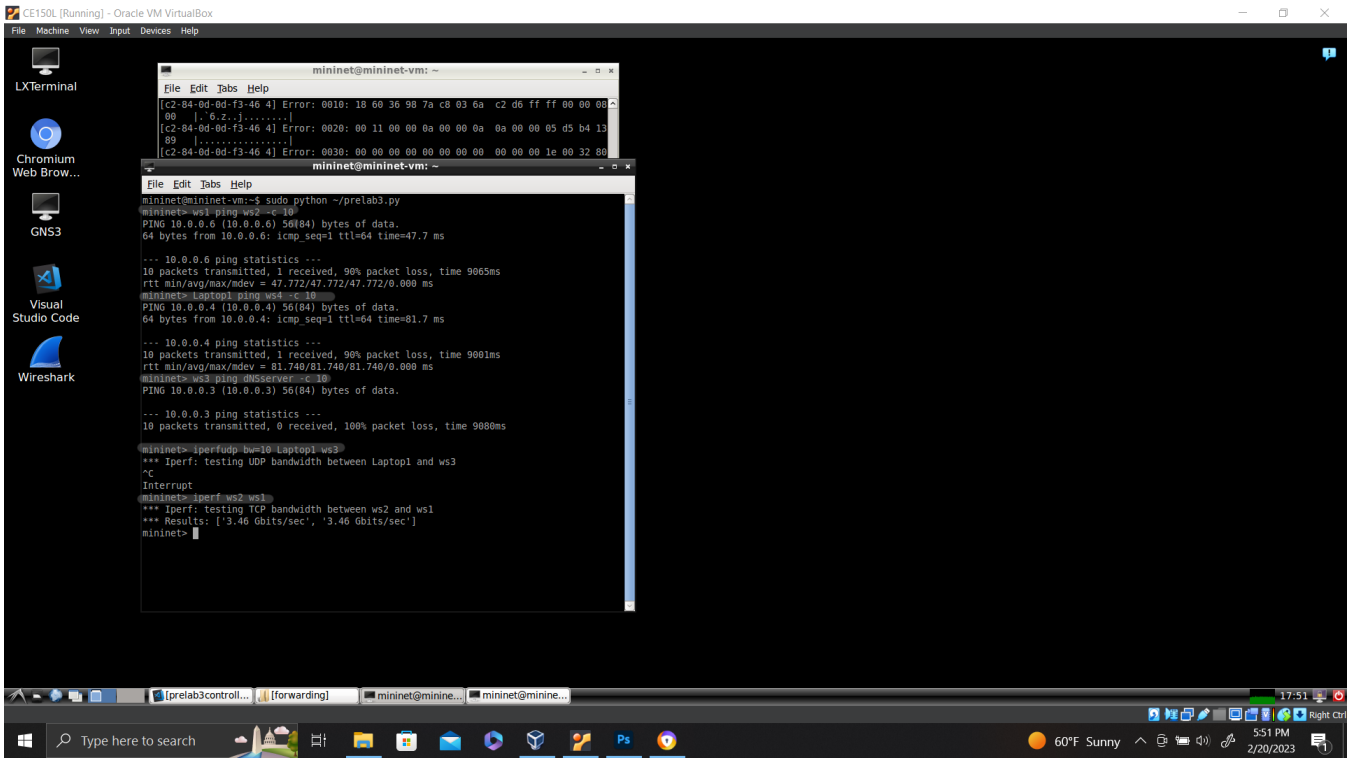
- Fails because the firewall prohibits pings to the DNS server.

d. `iperfudp bw=10 Laptop1 ws3`

- Fails because the firewall has not implemented rules to allow udp exchanges between Laptop1, and Workstation 3. Just keeps hanging, indicating the packets were dropped.

e. `iperf ws2 ws1`

- Passes, because TCP connections are allowed from Workstation2 to Workstation1 in the rules of the firewall.



14. [9 pts] Pingallfull

Stop the controller and exit the mininet prompt. Now open the `prelab3.py` file and change the IP address of Laptop2 to `10.0.5.6`. Restart the controller and run the `prelab3.py` file again (using the modified file). Run `pingallfull` from the mininet prompt.

- Is there a difference between the output of pingallfull here compared to running the command in 8a? What is different?
- Why is it different?

- Yes there is a difference. In the pingallfull Laptop2 is no longer included in any of the allowed traffic through the firewall. Also, in this pingallfull icmp traffic is not allowed to the DNS server, which was also shown in this pingallfull run.

15. [5 pts] If you wanted to run Traceroute between Workstation1 and Server2, would your firewall currently support that operation? Why or why not? State any assumptions you are making, such as the protocol(s) used by Traceroute. Explain your answer.

- My firewall would currently support this operation. If Traceroute was running on ICMP then the firewall would support the traffic fr0m Workstation1 and Server2.

16. [7 pts] Lifetime of Rules

Run a `TCP` command on the topology of your choosing. This command must run successfully. When done, run "`dpctl dump-flows`" without any delay. Explain the output. Provide a screenshot of the output of the `dpctl` command and highlight on one of the sections of the output the lines with "`idle_timeout`" and "`hard_timeout`".

- The **dpctl dump-flows** outputs the current flow table entries of an OpenFlow switch. A list of all the current flow table entries of the switch, including the flow's match fields, actions, and statistics are shown. Here the command displays the current flow of the core switch and switches 1-4.