Prelab3 - Introduction to SDN and OpenFlow

100 points

This lab builds on the knowledge acquired through Lab 1 where you were first introduced to the Mininet environment. It will also help you prepare for the class project. In Lab 1, you were introduced to some basic functionality of Mininet. In this lab, we will take that one step further by introducing you to Software-Defined Networking (SDN) and the OpenFlow protocol. **Link to OpenFlow Tutorial**: here

Important: This lab requires the old VM image. We cannot provide support for environments other than the old VM for this assignment. If you have an M1/M2 Mac which does not work properly or are otherwise unable to run the old VM, please use the computers in Jack Baskin 109 to complete the assignment. Also, make sure your scripts are written in Python 2, not Python 3.

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) that comprise a network.

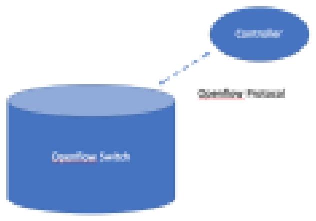
The SDN control plane is implemented by the "controller" and the data plane by "switches." The controller acts as the "brain" of the network and sends commands (a.k.a. "rules") to the switches on how to handle traffic. The OpenFlow protocol has emerged as the de facto SDN standard and specifies how the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

Mininet and OpenFlow:

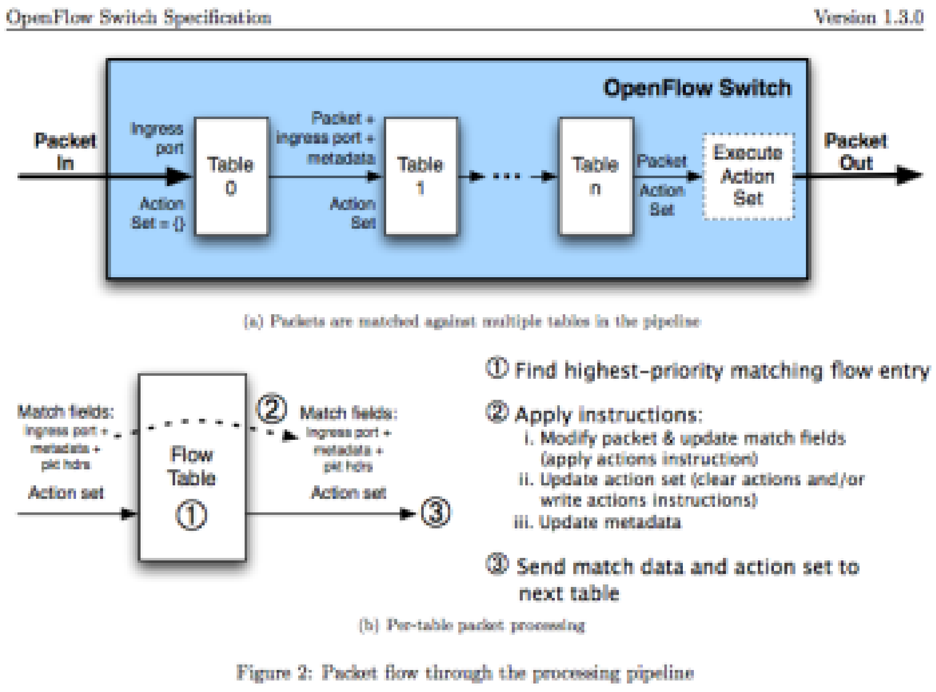
In Lab 1, we experimented with Mininet using its internal controller (and as such did not need to worry about OpenFlow). In this lab (and the next one), we will instead be using our own controller (also called remote controller) to send commands to the switches. We will be using the POX controller, which is written in Python.

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, irrespective of the vendor of the controller and switches (as you would expect with a standard protocol), and also different hardware types of the controller - switches.



The following diagram explains the operation of OpenFlow switches.


OpenFlow Switch Specification Version 1.3.0

OpenFlow Switch
Packet In
Ingress port
Action Set = {}
Table 0
Packet + ingress port + metadata
Action Set
Table 1
Table n
Packet
Action Set
Execute Action Set
Packet Out

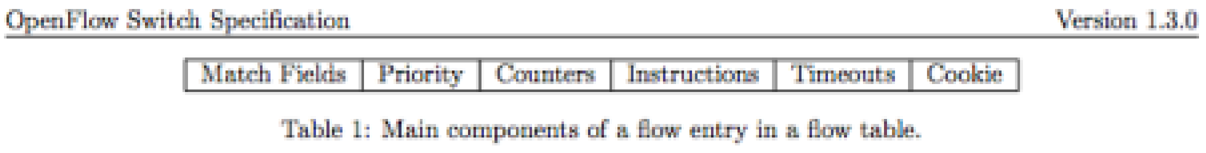
(a) Packets are matched against multiple tables in the pipeline

Match fields: Ingress port + metadata + pkt hdrs
Action set
Flow Table ①
② Match fields: Ingress port + metadata + pkt hdrs
Action set
③
① Find highest-priority matching flow entry
② Apply instructions:
i. Modify packet & update match fields (apply actions instruction)
ii. Update action set (clear actions and/or write actions instructions)
iii. Update metadata
③ Send match data and action set to next table

(b) Per-table packet processing

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:


OpenFlow Switch Specification Version 1.3.0

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

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

The figure below shows the flow of execution that 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

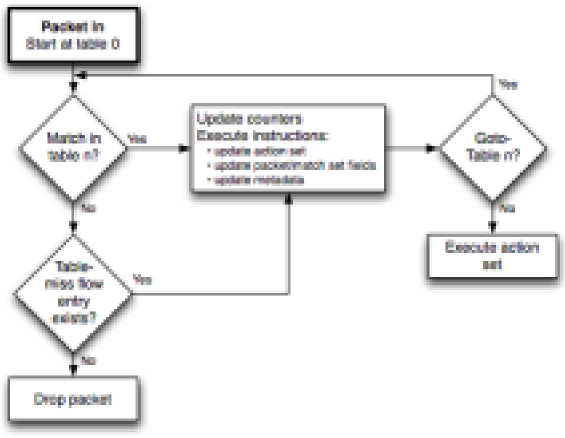


Figure 3: Flowchart detailing packet flow through an OpenFlow switch.

All of the figures and information in this section are from the [OpenFlow 1.3 specification](), which you can reference if you would like additional information

In the SDN network, the controller can be the default one in Mininet, also it can be the controller personally defined. For using a remote controller, you can find the information from the [OpenFlow tutorial](). There are different types of remote controllers you can use, for example POX and Ryu,etc. As stated above, in this assignment we are using the POX remote controller.

Assignment

Deliverables

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 3-6 do not require prelab3.py modification, but Question 7 requires you to modify prelab3.py

Introductory Questions:

1) **[5pts]** In your Mininet environment on Virtualbox, the “sudo mn” command invokes Mininet, along with its internal controller, whereas `sudo mn --controller=remote` invokes the remote controller.
   a. [2 pts] What command would you use to invoke the remote controller in a **single**

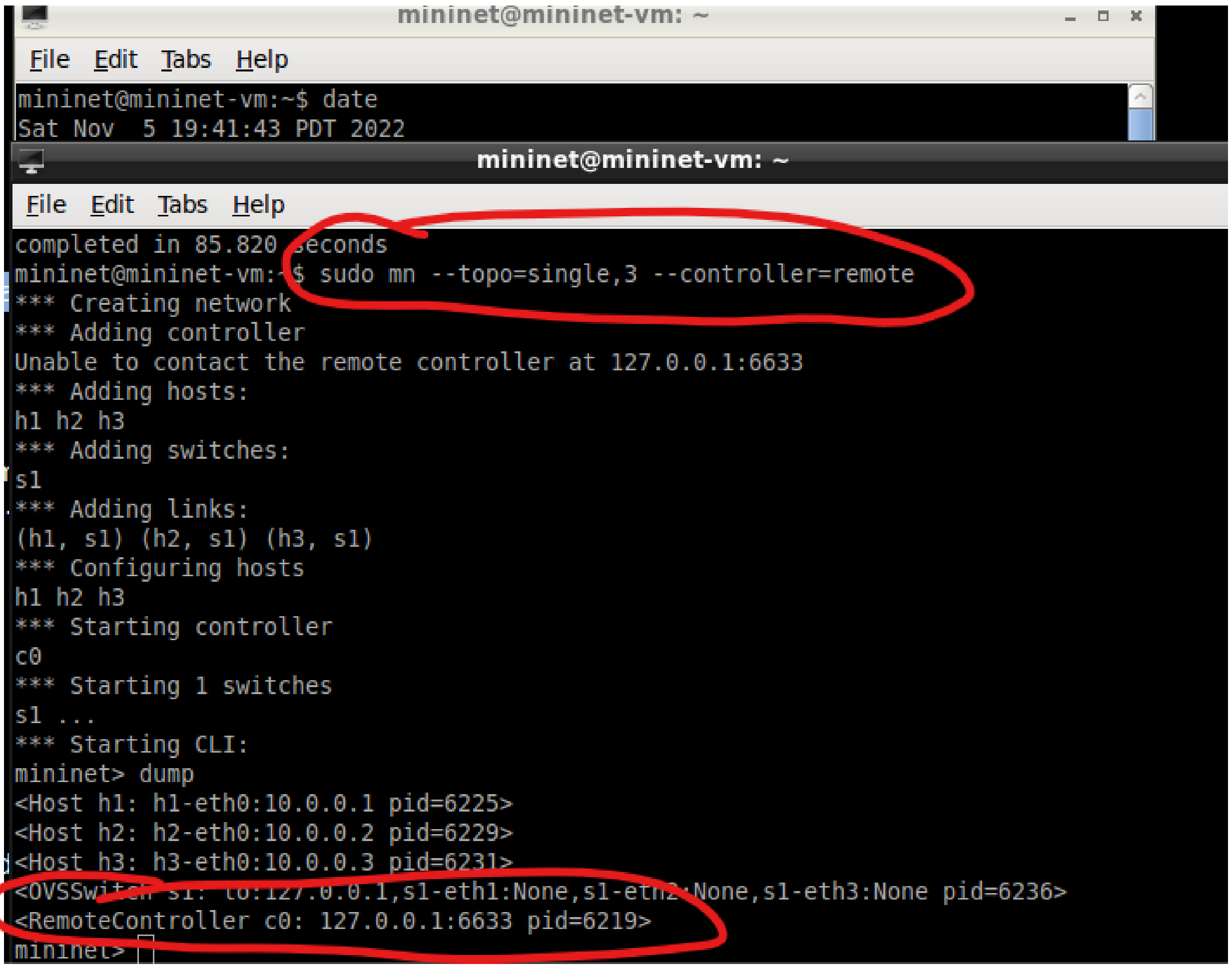
switch default-topology with **3 hosts** attached?

Use the command *sudo mn –topo=single,3 –controller=remote*

b. [3 pts] What command would you use to see which controller is being used by Mininet? What is the IP address and the port number used by the remote controller?

Use the dump command. The IP address is 127.0.0.1 and the port number is 6633.

Take a screenshot and highlight your answers for (a) and (b).



```
mininet@mininet-vm:~$ date
Sat Nov  5 19:41:43 PDT 2022
```

```
completed in 85.820 seconds
mininet@mininet-vm:~$ sudo mn --topo=single,3 --controller=remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=6225>
<Host h2: h2-eth0:10.0.0.2 pid=6229>
<Host h3: h3-eth0:10.0.0.3 pid=6231>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=6236>
<RemoteController c0: 127.0.0.1:6633 pid=6219>
mininet>
```

Building a Firewall with the POX Controller:

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.

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.

In this lab, we provide you with the Mininet configuration**, prelab3.py,** to set up your network which assumes a remote controller. You do not need to (and should not) modify this file. We also provide you with a skeleton POX controller**, prelab3controller.py**. This file requires modifications to create the firewall.

You can find the required python files here : prelab3.py prelab3controller.py

Building Your Firewall:

In this assignment you will build a firewall for the network shown below in **Topology 1.** To build the firewall you will modify the skeleton code in prelab3controller.py . The rules that you will need to implement in OpenFlow for this assignment are given below in **Table 1.** Be careful! Flow tables match the rule with highest priority first, where priority is established based on the order rules are placed in the table.

When you create a rule in the POX controller, you need to also have POX "install" the rule in the switch. This makes it so the switch "remembers" what to do for a few 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.

Useful Resources:

- POX WIKI
- Inside your VM, the `pox/forwarding/l2_learning.py` example file.

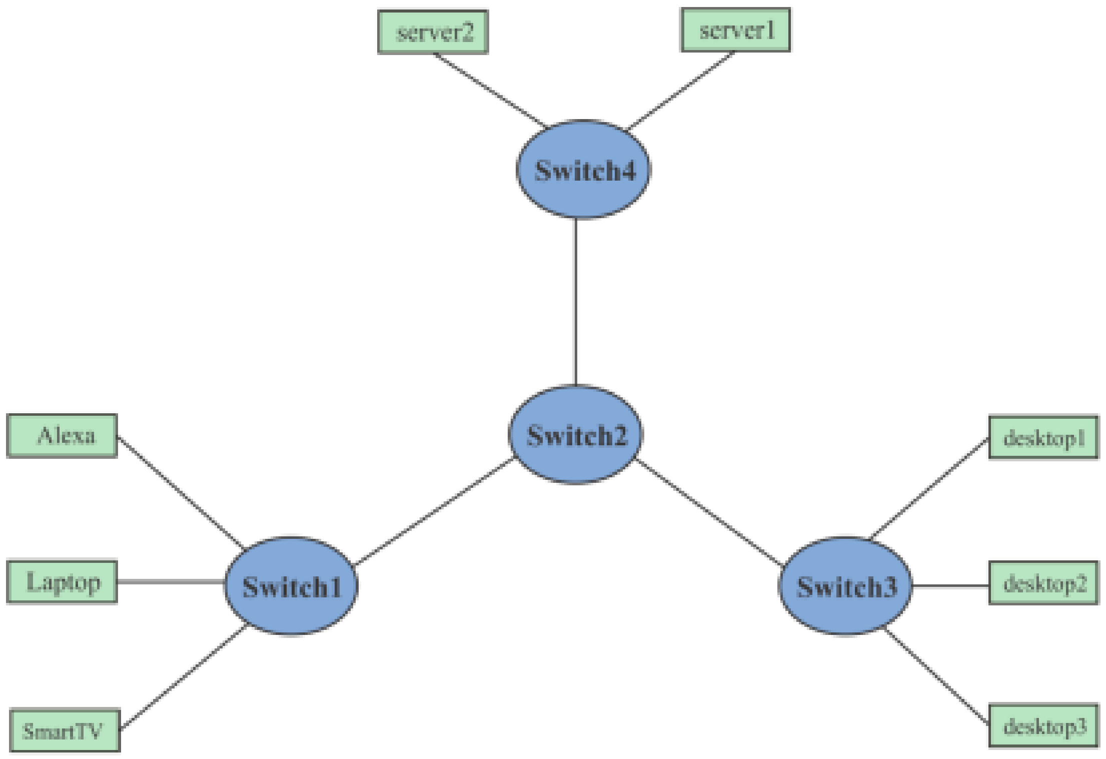
Running the Controller:

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



Topology 1

The rules that you will need to implement in OpenFlow for this assignment are given below in **Table 1:**

Rule #	src ip	dst ip protocol action
1	any ipv4	any ipv4 icmp accept
2	any	any arp accept
3	10.0.0.1	10.0.0.2 tcp accept
4	10.0.0.2	10.0.0.1 tcp accept
5	10.0.0.2	10.0.0.3 tcp accept
6	10.0.0.3	10.0.0.2 tcp accept
7	10.0.0.5	10.0.0.2 tcp accept
8	10.0.0.1	10.0.0.6 tcp accept

9 10	10.0.0.6 10.0.0.4	10.0.0.1 tcp accept 10.0.0.3 tcp accept
11 12	10.0.0.3 any	10.0.0.4 tcp accept any - drop

Table 1

2) [10 pts] Table 1: Briefly explain the meaning for Rules 1, 2, 3 and 11. A sentence or two for each rule is sufficient for the explanation.

Rule 1: If the source IP is any ipv4 address, the destination IP will accept ICMP protocols.

Rule 2: If the source IP is any other address type that isn't ipv4, the destination IP will accept ARP protocols.

Rule 3: If the source IP is specifically 10.0.0.1, only the destination IP of 10.0.0.2 will accept a TCP protocol.

Rule 11: If the source IP is 10.0.0.3, only the destination IP of 10.0.4 will accept a TCP protocol. If the destination IP is any other address, the packet will be dropped.

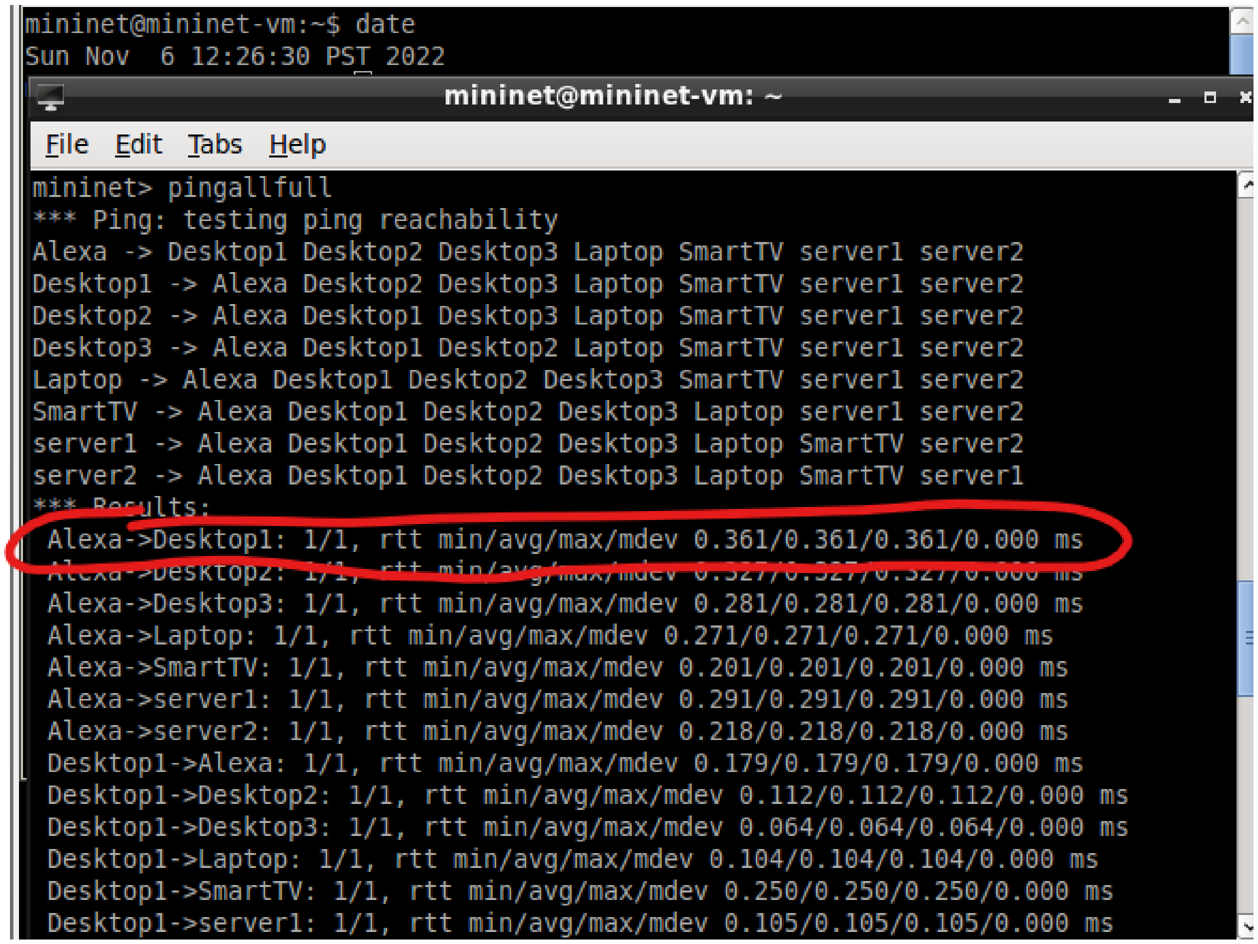
What happens if a packet is sent from Desktop1 to Desktop2?

The packet will be dropped, because Desktop1 IP address is 10.0.0.6 and that address can only accept traffic from 10.0.0.1 and 10.0.0.3. Desktop2 address is 10.0.0.7.

<u>Testing the Firewall:</u>

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

3. **[20 points]**
   a**. pingallfull [10 points]**:
      i. [3 points] Describe the expected result of running this command
         I expect mininet to test the connectivity between each host, and to show 0 packet loss.
      ii. [7 points] Explain which of the firewall rules contributed to the result.
      Rule 1 contributed to this result, because it allows the firewall to accept any ICMP traffic and pingallfull uses ICMP.
      Provide a screenshot of the results and highlight Alexa → laptop



```
mininet@mininet-vm:~$ date
Sun Nov  6 12:26:30 PST 2022
```

```
mininet> pingallfull
*** Ping: testing ping reachability
Alexa -> Desktop1 Desktop2 Desktop3 Laptop SmartTV server1 server2
Desktop1 -> Alexa Desktop2 Desktop3 Laptop SmartTV server1 server2
Desktop2 -> Alexa Desktop1 Desktop3 Laptop SmartTV server1 server2
Desktop3 -> Alexa Desktop1 Desktop2 Laptop SmartTV server1 server2
Laptop -> Alexa Desktop1 Desktop2 Desktop3 SmartTV server1 server2
SmartTV -> Alexa Desktop1 Desktop2 Desktop3 Laptop server1 server2
server1 -> Alexa Desktop1 Desktop2 Desktop3 Laptop SmartTV server2
server2 -> Alexa Desktop1 Desktop2 Desktop3 Laptop SmartTV server1
*** Results:
 Alexa->Desktop1: 1/1, rtt min/avg/max/mdev 0.361/0.361/0.361/0.000 ms
 Alexa->Desktop2: 1/1, rtt min/avg/max/mdev 0.327/0.327/0.327/0.000 ms
 Alexa->Desktop3: 1/1, rtt min/avg/max/mdev 0.281/0.281/0.281/0.000 ms
 Alexa->Laptop: 1/1, rtt min/avg/max/mdev 0.271/0.271/0.271/0.000 ms
 Alexa->SmartTV: 1/1, rtt min/avg/max/mdev 0.201/0.201/0.201/0.000 ms
 Alexa->server1: 1/1, rtt min/avg/max/mdev 0.291/0.291/0.291/0.000 ms
 Alexa->server2: 1/1, rtt min/avg/max/mdev 0.218/0.218/0.218/0.000 ms
 Desktop1->Alexa: 1/1, rtt min/avg/max/mdev 0.179/0.179/0.179/0.000 ms
 Desktop1->Desktop2: 1/1, rtt min/avg/max/mdev 0.112/0.112/0.112/0.000 ms
 Desktop1->Desktop3: 1/1, rtt min/avg/max/mdev 0.064/0.064/0.064/0.000 ms
 Desktop1->Laptop: 1/1, rtt min/avg/max/mdev 0.104/0.104/0.104/0.000 ms
 Desktop1->SmartTV: 1/1, rtt min/avg/max/mdev 0.250/0.250/0.250/0.000 ms
 Desktop1->server1: 1/1, rtt min/avg/max/mdev 0.105/0.105/0.105/0.000 ms
```

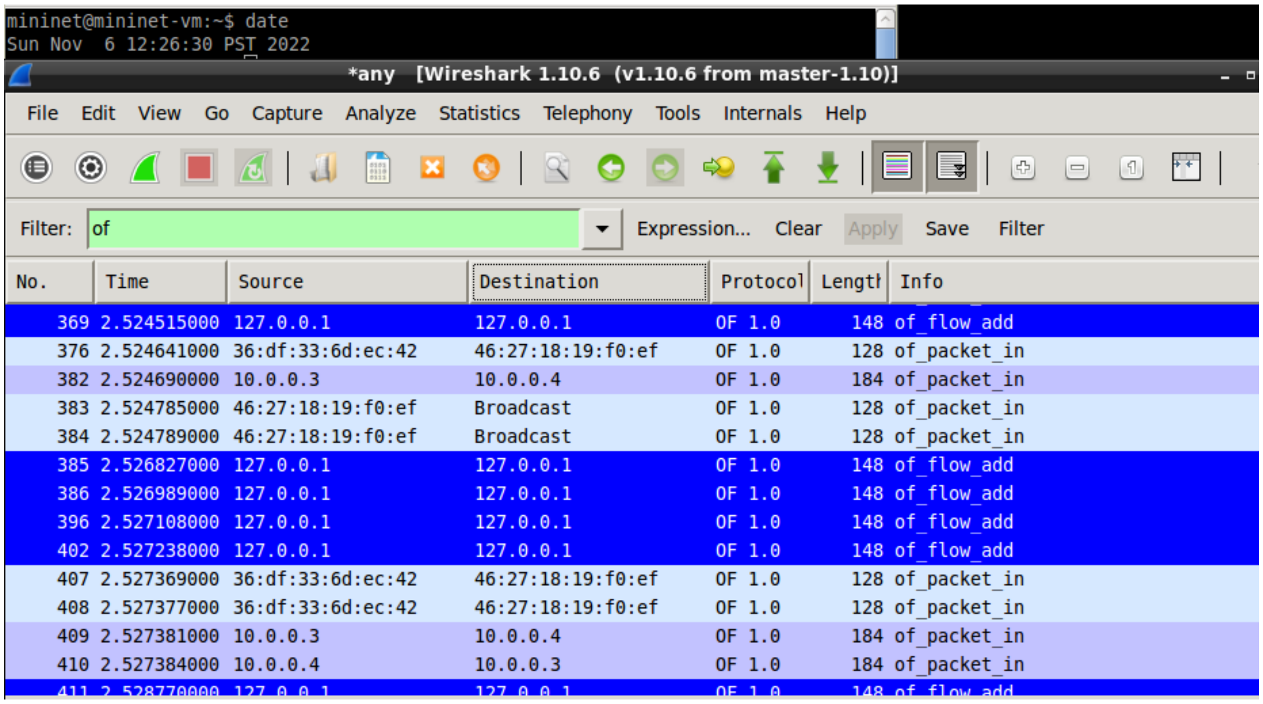
   b. **Wireshark Analysis of ping [10 points]:**

If your controller and topology script are running, stop them and re-start both scripts. Start Wireshark and capture on the 'any' interface. Re-run the pingall command to completion and stop the capture.

With the help of the display filter, display all OpenFlow traffic (of), capturing packets sent from the switch to the controller. Scroll down until you see multiple **of_packet_in** packets.

What do you observe? How are these packets used? Take a screenshot showing your results.

The packets are going through the firewall, and the firewall is deciding which ones to accept.



```
mininet@mininet-vm:~$ date
Sun Nov  6 12:26:30 PST 2022
```

*any [Wireshark 1.10.6 (v1.10.6 from master-1.10)]

Filter: of

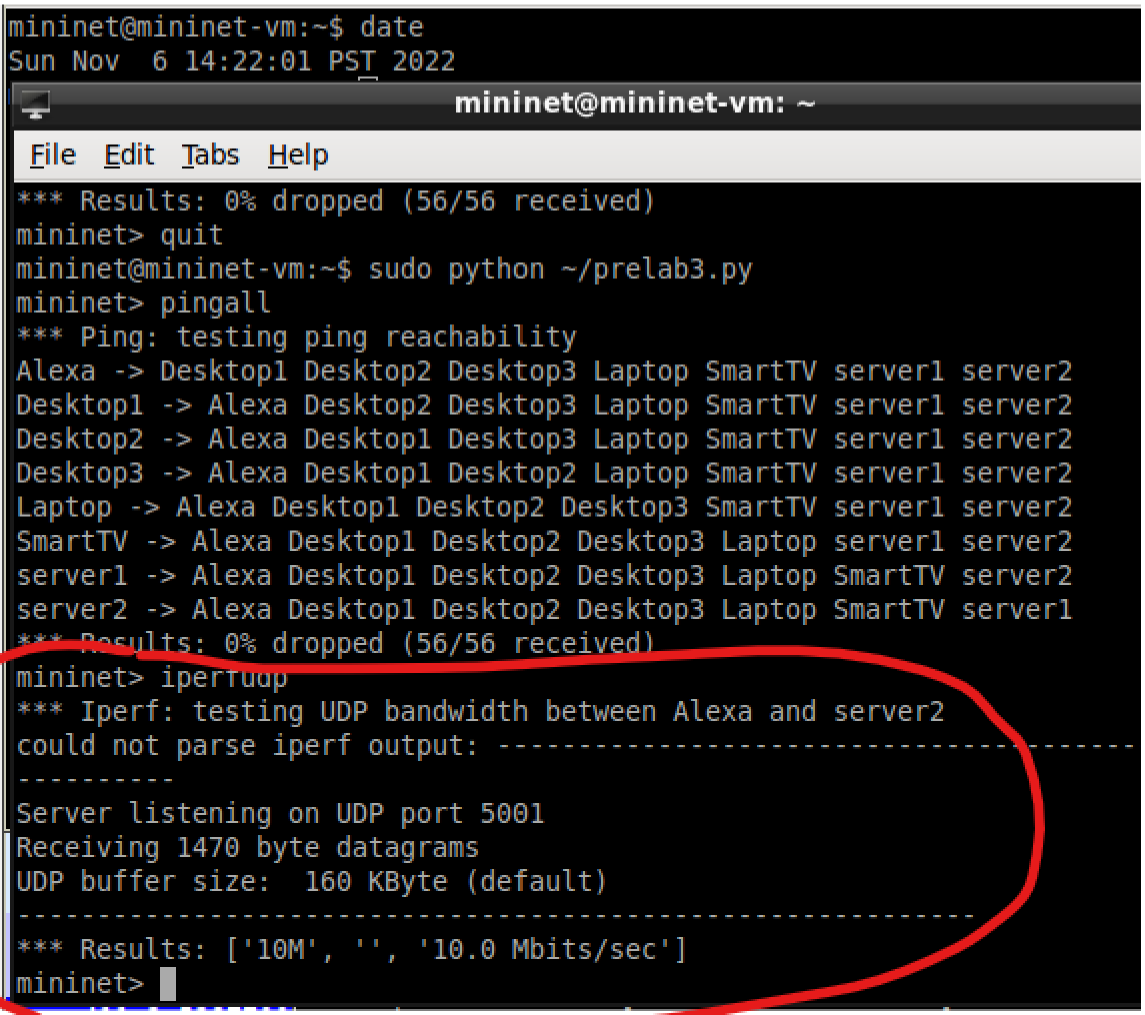
```
No.  Time        Source             Destination        Protocol Length Info
 369 2.524515000 127.0.0.1          127.0.0.1          OF 1.0   148 of_flow_add
 376 2.524641000 36:df:33:6d:ec:42  46:27:18:19:f0:ef  OF 1.0   128 of_packet_in
 382 2.524690000 10.0.0.3           10.0.0.4           OF 1.0   184 of_packet_in
 383 2.524785000 46:27:18:19:f0:ef  Broadcast          OF 1.0   128 of_packet_in
 384 2.524789000 46:27:18:19:f0:ef  Broadcast          OF 1.0   128 of_packet_in
 385 2.526827000 127.0.0.1          127.0.0.1          OF 1.0   148 of_flow_add
 386 2.526989000 127.0.0.1          127.0.0.1          OF 1.0   148 of_flow_add
 396 2.527108000 127.0.0.1          127.0.0.1          OF 1.0   148 of_flow_add
 402 2.527238000 127.0.0.1          127.0.0.1          OF 1.0   148 of_flow_add
 407 2.527369000 36:df:33:6d:ec:42  46:27:18:19:f0:ef  OF 1.0   128 of_packet_in
 408 2.527377000 36:df:33:6d:ec:42  46:27:18:19:f0:ef  OF 1.0   128 of_packet_in
 409 2.527381000 10.0.0.3           10.0.0.4           OF 1.0   184 of_packet_in
 410 2.527384000 10.0.0.4           10.0.0.3           OF 1.0   184 of_packet_in
 411 2.528770000 127.0.0.1          127.0.0.1          OF 1.0   148 of_flow_add
```

4. **[15 points] `iperfudp`**

Run the `iperfudp` command from the mininet terminal.

a. [5 points] Describe the outcome and provide an explanation for this result. Justify your results with a screenshot.

The terminal says it cannot parse the UDP bandwidth because no rule in the flow table allows a UDP connection.

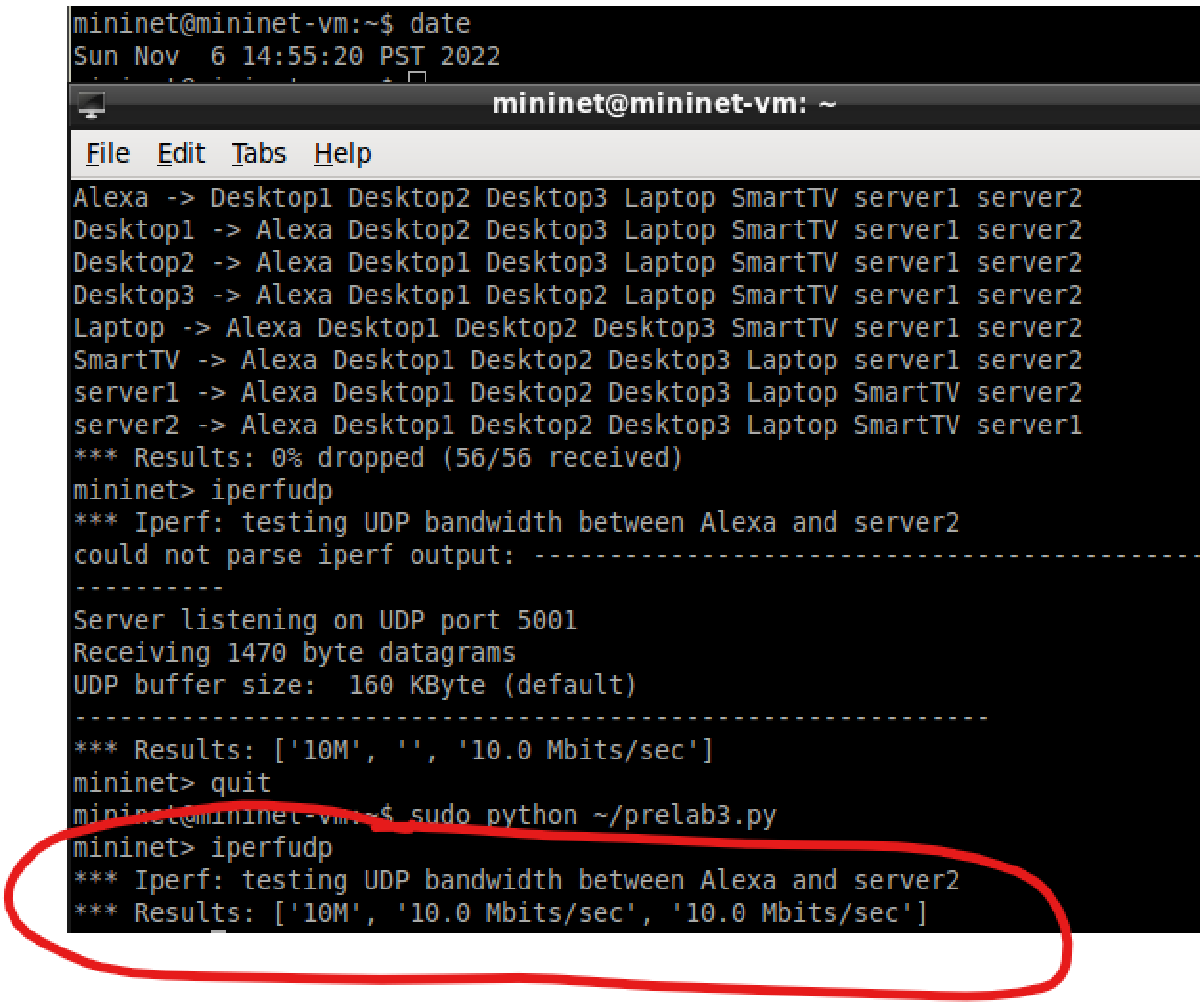


```
mininet@mininet-vm:~$ date
Sun Nov  6 14:22:01 PST 2022
```

```
*** Results: 0% dropped (56/56 received)
mininet> quit
mininet@mininet-vm:~$ sudo python ~/prelab3.py
mininet> pingall
*** Ping: testing ping reachability
Alexa -> Desktop1 Desktop2 Desktop3 Laptop SmartTV server1 server2
Desktop1 -> Alexa Desktop2 Desktop3 Laptop SmartTV server1 server2
Desktop2 -> Alexa Desktop1 Desktop3 Laptop SmartTV server1 server2
Desktop3 -> Alexa Desktop1 Desktop2 Laptop SmartTV server1 server2
Laptop -> Alexa Desktop1 Desktop2 Desktop3 SmartTV server1 server2
SmartTV -> Alexa Desktop1 Desktop2 Desktop3 Laptop server1 server2
server1 -> Alexa Desktop1 Desktop2 Desktop3 Laptop SmartTV server2
server2 -> Alexa Desktop1 Desktop2 Desktop3 Laptop SmartTV server1
*** Results: 0% dropped (56/56 received)
mininet> iperfudp
*** Iperf: testing UDP bandwidth between Alexa and server2
could not parse iperf output: ----------------------------------------
----------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
*** Results: ['10M', '', '10.0 Mbits/sec']
mininet> 
```

b. [10 points] Implement a new rule in the firewall that allows UDP traffic. Show the new rule in a table format as shown in Table 1 (all of the column headings). Take a screenshot showing the successful iperfudp output.

Rule #	src ip	dst ip	protocol	action
13	any	any	udp	accept



```
mininet@mininet-vm:~$ date
Sun Nov  6 14:55:20 PST 2022
```

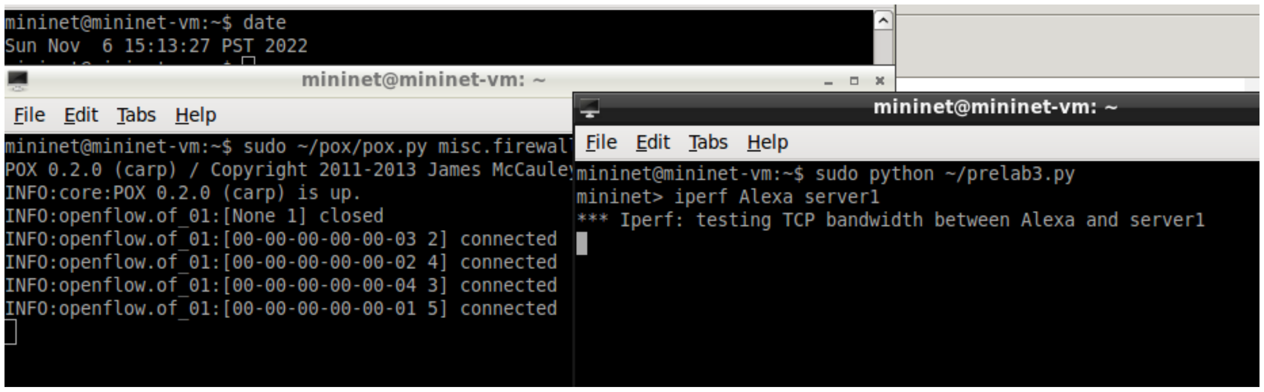
```
Alexa -> Desktop1 Desktop2 Desktop3 Laptop SmartTV server1 server2
Desktop1 -> Alexa Desktop2 Desktop3 Laptop SmartTV server1 server2
Desktop2 -> Alexa Desktop1 Desktop3 Laptop SmartTV server1 server2
Desktop3 -> Alexa Desktop1 Desktop2 Laptop SmartTV server1 server2
Laptop -> Alexa Desktop1 Desktop2 Desktop3 SmartTV server1 server2
SmartTV -> Alexa Desktop1 Desktop2 Desktop3 Laptop server1 server2
server1 -> Alexa Desktop1 Desktop2 Desktop3 Laptop SmartTV server2
server2 -> Alexa Desktop1 Desktop2 Desktop3 Laptop SmartTV server1
*** Results: 0% dropped (56/56 received)
mininet> iperfudp
*** Iperf: testing UDP bandwidth between Alexa and server2
could not parse iperf output: -----------------------------------------
----------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
*** Results: ['10M', '', '10.0 Mbits/sec']
mininet> quit
mininet@mininet-vm:~$ sudo python ~/prelab3.py
mininet> iperfudp
*** Iperf: testing UDP bandwidth between Alexa and server2
*** Results: ['10M', '10.0 Mbits/sec', '10.0 Mbits/sec']
```

5. **[15 points] iperf**

   a. **[10 pts]** Perform `iperf` between the following pairs of hosts. Provide a screenshot and explain what you observe. Were the `iperf commands` successful? Why?

      i. `Alexa - server1`

`This command did not work because this connection is not a part of the rules in the flow table, so the connection just hangs.`



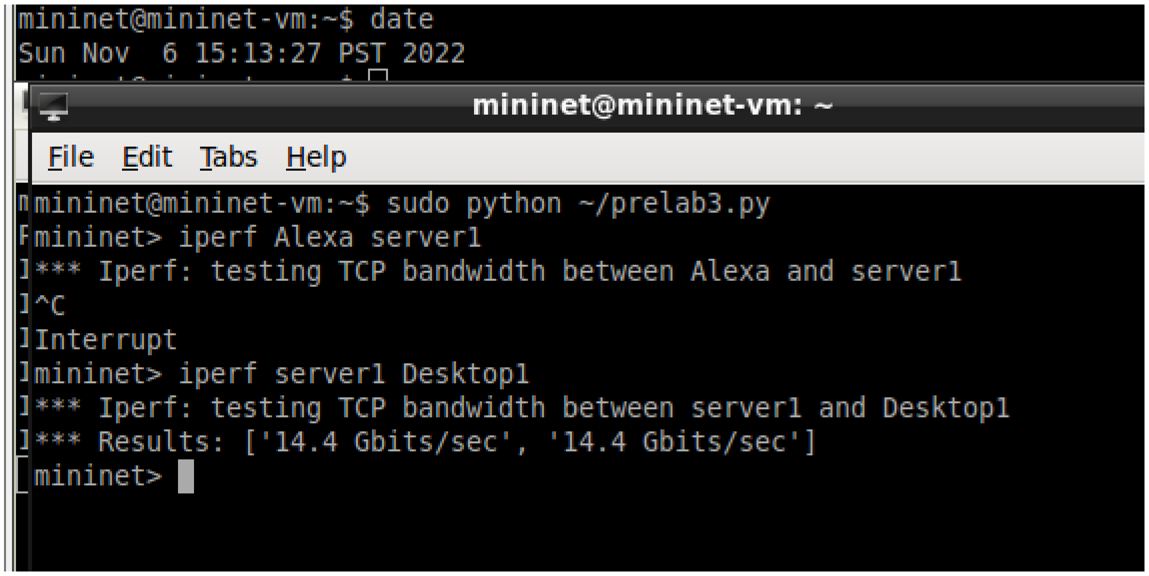
```
mininet@mininet-vm:~$ date
Sun Nov  6 15:13:27 PST 2022
```

```
mininet@mininet-vm:~$ sudo ~/pox/pox.py misc.firewal
POX 0.2.0 (carp) / Copyright 2011-2013 James McCaule
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-03 2] connected
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
INFO:openflow.of_01:[00-00-00-00-00-04 3] connected
INFO:openflow.of_01:[00-00-00-00-00-01 5] connected
```

```
mininet@mininet-vm:~$ sudo python ~/prelab3.py
mininet> iperf Alexa server1
*** Iperf: testing TCP bandwidth between Alexa and server1
```

ii. server1 - Desktop1

This command worked because the connection between these two hosts is allowed based on the flow table.

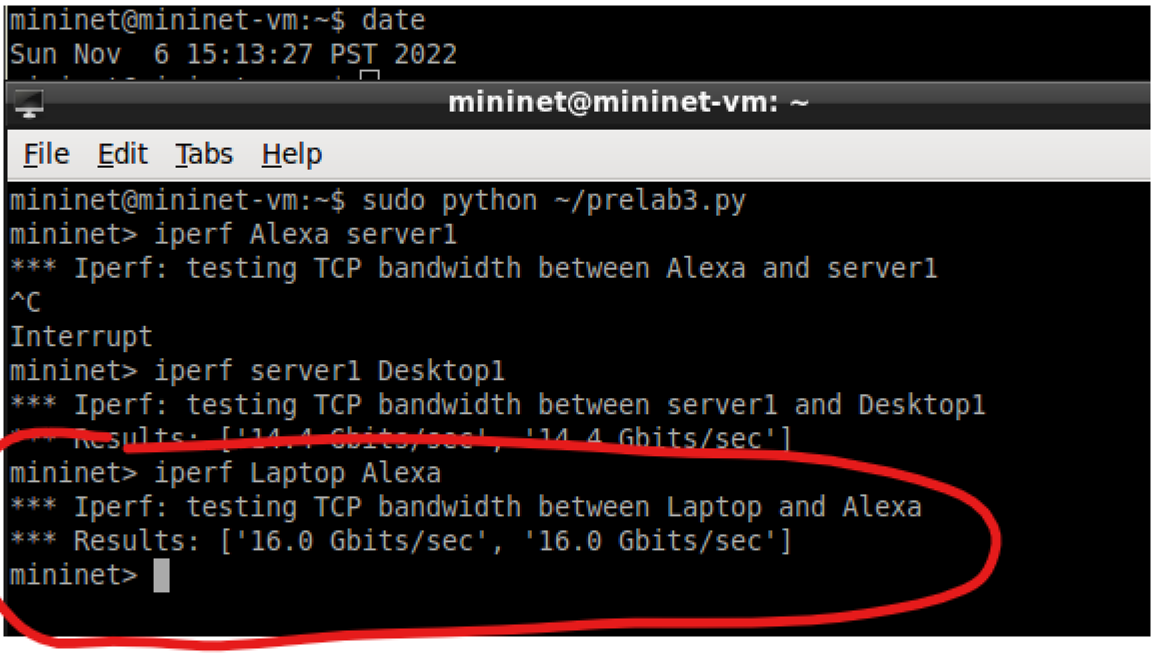


```
mininet@mininet-vm:~$ date
Sun Nov  6 15:13:27 PST 2022
```

```
mininet@mininet-vm:~$ sudo python ~/prelab3.py
mininet> iperf Alexa server1
*** Iperf: testing TCP bandwidth between Alexa and server1
^C
Interrupt
mininet> iperf server1 Desktop1
*** Iperf: testing TCP bandwidth between server1 and Desktop1
*** Results: ['14.4 Gbits/sec', '14.4 Gbits/sec']
mininet>
```

iii. Laptop - Alexa

This command worked because the connection between these two hosts is allowed based on the flow table.



```
mininet@mininet-vm:~$ date
Sun Nov  6 15:13:27 PST 2022
```

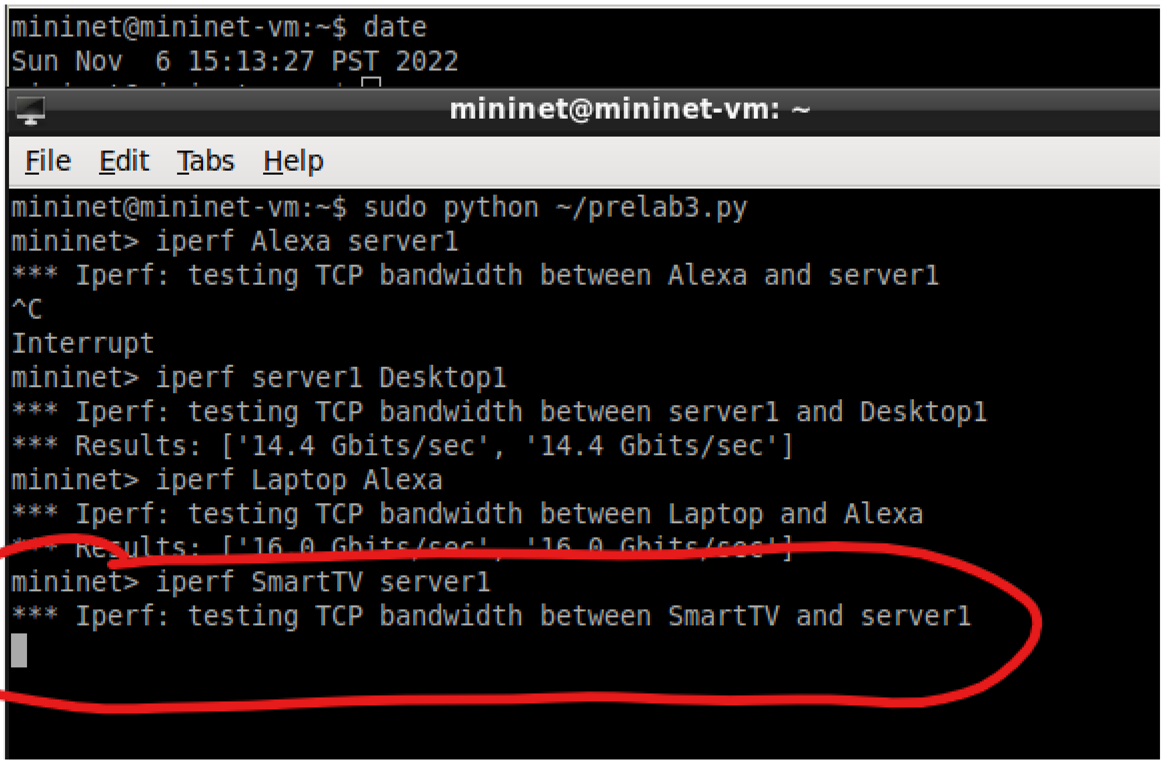
mininet@mininet-vm: ~

File Edit Tabs Help

```
mininet@mininet-vm:~$ sudo python ~/prelab3.py
mininet> iperf Alexa server1
*** Iperf: testing TCP bandwidth between Alexa and server1
^C
Interrupt
mininet> iperf server1 Desktop1
*** Iperf: testing TCP bandwidth between server1 and Desktop1
*** Results: ['14.4 Gbits/sec', '14.4 Gbits/sec']
mininet> iperf Laptop Alexa
*** Iperf: testing TCP bandwidth between Laptop and Alexa
*** Results: ['16.0 Gbits/sec', '16.0 Gbits/sec']
mininet>
```

iv. SmartTV - server1

This command did not work because this connection is not a part of the rules in the flow table, so the connection just hangs.



```
mininet@mininet-vm:~$ date
Sun Nov  6 15:13:27 PST 2022
```

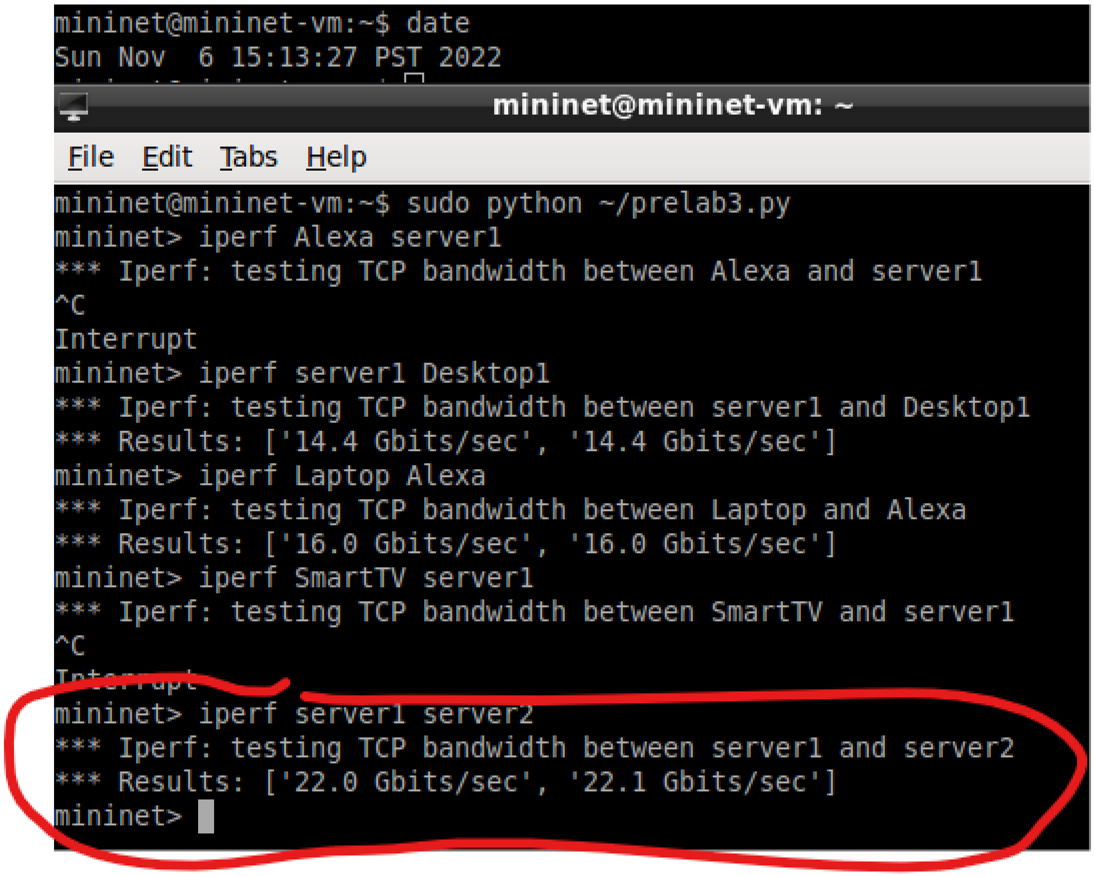
mininet@mininet-vm: ~

File Edit Tabs Help

```
mininet@mininet-vm:~$ sudo python ~/prelab3.py
mininet> iperf Alexa server1
*** Iperf: testing TCP bandwidth between Alexa and server1
^C
Interrupt
mininet> iperf server1 Desktop1
*** Iperf: testing TCP bandwidth between server1 and Desktop1
*** Results: ['14.4 Gbits/sec', '14.4 Gbits/sec']
mininet> iperf Laptop Alexa
*** Iperf: testing TCP bandwidth between Laptop and Alexa
*** Results: ['16.0 Gbits/sec', '16.0 Gbits/sec']
mininet> iperf SmartTV server1
*** Iperf: testing TCP bandwidth between SmartTV and server1
```

v. server1 - server2

This command worked because the connection between these two hosts is allowed based on the flow table.

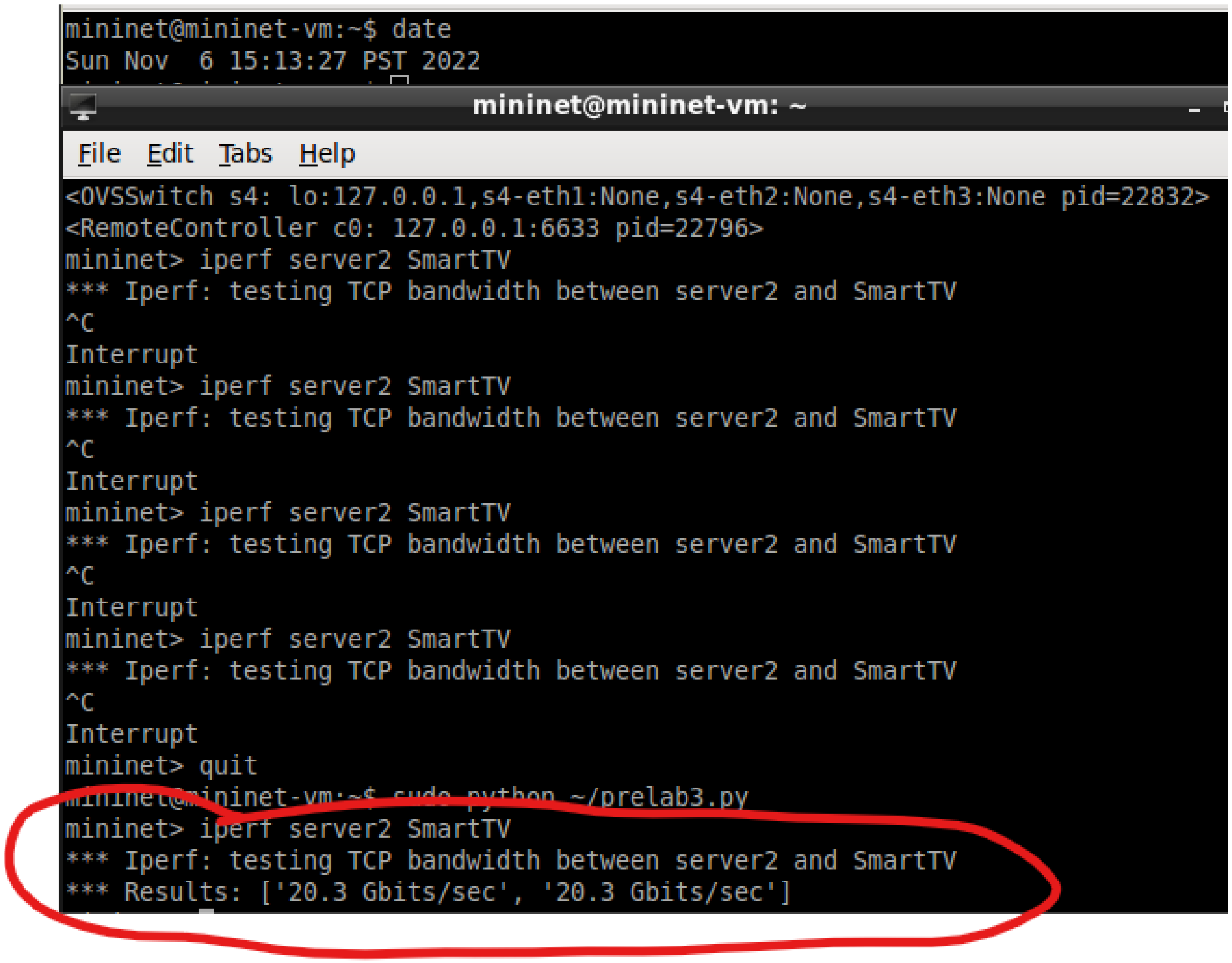


```
mininet@mininet-vm:~$ date
Sun Nov  6 15:13:27 PST 2022
```

```
mininet@mininet-vm:~$ sudo python ~/prelab3.py
mininet> iperf Alexa server1
*** Iperf: testing TCP bandwidth between Alexa and server1
^C
Interrupt
mininet> iperf server1 Desktop1
*** Iperf: testing TCP bandwidth between server1 and Desktop1
*** Results: ['14.4 Gbits/sec', '14.4 Gbits/sec']
mininet> iperf Laptop Alexa
*** Iperf: testing TCP bandwidth between Laptop and Alexa
*** Results: ['16.0 Gbits/sec', '16.0 Gbits/sec']
mininet> iperf SmartTV server1
*** Iperf: testing TCP bandwidth between SmartTV and server1
^C
Interrupt
mininet> iperf server1 server2
*** Iperf: testing TCP bandwidth between server1 and server2
*** Results: ['22.0 Gbits/sec', '22.1 Gbits/sec']
mininet> 
```

b. **[5 pts]** Run `iperf server2 SmartTV` - If you see all packets are being received, show a screenshot of the packet flow in Wireshark when `iperf` is run. If you do not see all packets are being exchanged, add a new rule (rule #14) such that `iperf` shows full packet exchange. What new rule did you add? Show a screenshot in your terminal verifying your results.

ANSWER BELOW



```
mininet@mininet-vm:~$ date
Sun Nov  6 15:13:27 PST 2022
```

```
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None pid=22832>
<RemoteController c0: 127.0.0.1:6633 pid=22796>
mininet> iperf server2 SmartTV
*** Iperf: testing TCP bandwidth between server2 and SmartTV
^C
Interrupt
mininet> iperf server2 SmartTV
*** Iperf: testing TCP bandwidth between server2 and SmartTV
^C
Interrupt
mininet> iperf server2 SmartTV
*** Iperf: testing TCP bandwidth between server2 and SmartTV
^C
Interrupt
mininet> iperf server2 SmartTV
*** Iperf: testing TCP bandwidth between server2 and SmartTV
^C
Interrupt
mininet> quit
mininet@mininet-vm:~$ sudo python ~/prelab3.py
mininet> iperf server2 SmartTV
*** Iperf: testing TCP bandwidth between server2 and SmartTV
*** Results: ['20.3 Gbits/sec', '20.3 Gbits/sec']
```

Rule #	src ip	dst ip	protocol	action
14	10.0.0.2	10.0.0.5	tcp	accept

6. **[15 points] pingallfull**
Stop the controller and exit the mininet prompt. Now open the prelab3.py file and change the IP address of Server2 to `10.0.10.20`. Restart the controller and run the prelab3.py file again using the steps given above. Run `pingallfull` from the mininet prompt.

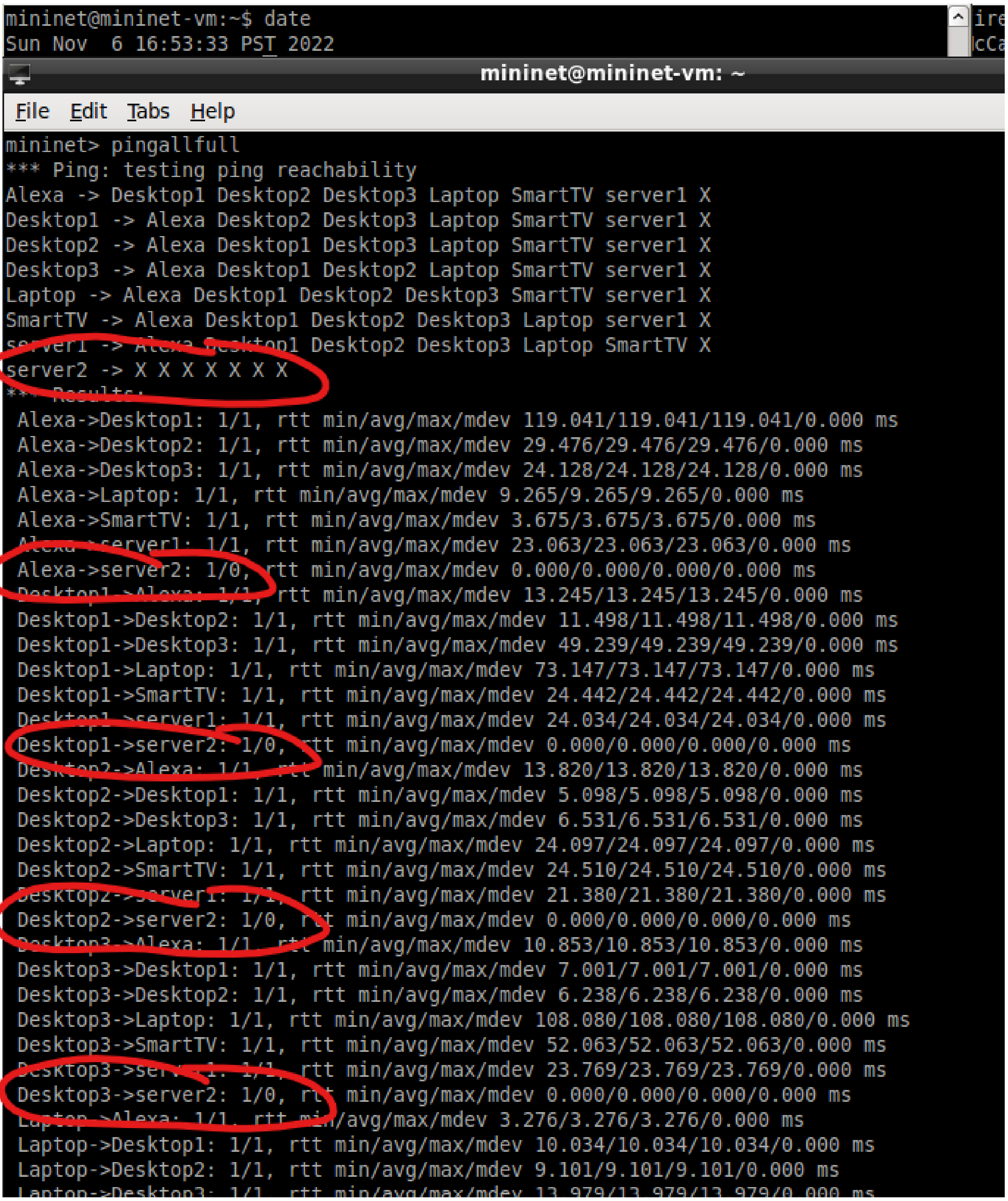
- Does this output match with the output from question 3a?

No this output does not match the output from question 3a.
- Explain how the outputs vary/match and why.

The outputs vary because pings are being dropped from server2. This is because we

changed the IP address to 10.0.10.20 which has a completely different subnet. The controller does not have a rule to route between subnets.

- Provide a screenshot of the terminal output and highlight any differences. Remove the rule once this problem is solved.



```
mininet@mininet-vm:~$ date
Sun Nov  6 16:53:33 PST 2022
```

mininet@mininet-vm: ~

File Edit Tabs Help

```
mininet> pingallfull
*** Ping: testing ping reachability
Alexa -> Desktop1 Desktop2 Desktop3 Laptop SmartTV server1 X
Desktop1 -> Alexa Desktop2 Desktop3 Laptop SmartTV server1 X
Desktop2 -> Alexa Desktop1 Desktop3 Laptop SmartTV server1 X
Desktop3 -> Alexa Desktop1 Desktop2 Laptop SmartTV server1 X
Laptop -> Alexa Desktop1 Desktop2 Desktop3 SmartTV server1 X
SmartTV -> Alexa Desktop1 Desktop2 Desktop3 Laptop server1 X
server1 -> Alexa Desktop1 Desktop2 Desktop3 Laptop SmartTV X
server2 -> X X X X X X X
*** Results:
 Alexa->Desktop1: 1/1, rtt min/avg/max/mdev 119.041/119.041/119.041/0.000 ms
 Alexa->Desktop2: 1/1, rtt min/avg/max/mdev 29.476/29.476/29.476/0.000 ms
 Alexa->Desktop3: 1/1, rtt min/avg/max/mdev 24.128/24.128/24.128/0.000 ms
 Alexa->Laptop: 1/1, rtt min/avg/max/mdev 9.265/9.265/9.265/0.000 ms
 Alexa->SmartTV: 1/1, rtt min/avg/max/mdev 3.675/3.675/3.675/0.000 ms
 Alexa->server1: 1/1, rtt min/avg/max/mdev 23.063/23.063/23.063/0.000 ms
 Alexa->server2: 1/0, rtt min/avg/max/mdev 0.000/0.000/0.000/0.000 ms
 Desktop1->Alexa: 1/1, rtt min/avg/max/mdev 13.245/13.245/13.245/0.000 ms
 Desktop1->Desktop2: 1/1, rtt min/avg/max/mdev 11.498/11.498/11.498/0.000 ms
 Desktop1->Desktop3: 1/1, rtt min/avg/max/mdev 49.239/49.239/49.239/0.000 ms
 Desktop1->Laptop: 1/1, rtt min/avg/max/mdev 73.147/73.147/73.147/0.000 ms
 Desktop1->SmartTV: 1/1, rtt min/avg/max/mdev 24.442/24.442/24.442/0.000 ms
 Desktop1->server1: 1/1, rtt min/avg/max/mdev 24.034/24.034/24.034/0.000 ms
 Desktop1->server2: 1/0, rtt min/avg/max/mdev 0.000/0.000/0.000/0.000 ms
 Desktop2->Alexa: 1/1, rtt min/avg/max/mdev 13.820/13.820/13.820/0.000 ms
 Desktop2->Desktop1: 1/1, rtt min/avg/max/mdev 5.098/5.098/5.098/0.000 ms
 Desktop2->Desktop3: 1/1, rtt min/avg/max/mdev 6.531/6.531/6.531/0.000 ms
 Desktop2->Laptop: 1/1, rtt min/avg/max/mdev 24.097/24.097/24.097/0.000 ms
 Desktop2->SmartTV: 1/1, rtt min/avg/max/mdev 24.510/24.510/24.510/0.000 ms
 Desktop2->server1: 1/1, rtt min/avg/max/mdev 21.380/21.380/21.380/0.000 ms
 Desktop2->server2: 1/0, rtt min/avg/max/mdev 0.000/0.000/0.000/0.000 ms
 Desktop3->Alexa: 1/1, rtt min/avg/max/mdev 10.853/10.853/10.853/0.000 ms
 Desktop3->Desktop1: 1/1, rtt min/avg/max/mdev 7.001/7.001/7.001/0.000 ms
 Desktop3->Desktop2: 1/1, rtt min/avg/max/mdev 6.238/6.238/6.238/0.000 ms
 Desktop3->Laptop: 1/1, rtt min/avg/max/mdev 108.080/108.080/108.080/0.000 ms
 Desktop3->SmartTV: 1/1, rtt min/avg/max/mdev 52.063/52.063/52.063/0.000 ms
 Desktop3->server1: 1/1, rtt min/avg/max/mdev 23.769/23.769/23.769/0.000 ms
 Desktop3->server2: 1/0, rtt min/avg/max/mdev 0.000/0.000/0.000/0.000 ms
 Laptop->Alexa: 1/1, rtt min/avg/max/mdev 3.276/3.276/3.276/0.000 ms
 Laptop->Desktop1: 1/1, rtt min/avg/max/mdev 10.034/10.034/10.034/0.000 ms
 Laptop->Desktop2: 1/1, rtt min/avg/max/mdev 9.101/9.101/9.101/0.000 ms
 Laptop->Desktop3: 1/1, rtt min/avg/max/mdev 13.979/13.979/13.979/0.000 ms
```

7. **[10 points]** If you wanted to allow Traceroute to run between SmartTV and server1, would your firewall currently support that operation? Why or why not? State any assumptions you are making, such as the protocol used for the probes etc. Explain your answer.

The firewall would support a connection between SmartTv and server1 since traceroute commonly uses ICMP echo packets with TTL values. The firewall allows any icmp protocol connection from rule #1, so traceroute would work.

8. **[10 points]** Would your firewall currently support DNS queries? Why or why not? Explain your answer.

The firewall would support DNS queries because DNS uses UDP packets, and the firewall allows any UDP connections between IPs. We established this rule in the lab.