Lab 1 100 points

Remember your Timestamps for the screenshots!

Part A: Installing and Using Mininet

Resources for this lab:

- Computer Networking: A Top-down Approach Chapter 1 Section 1.4: Delay, Traceroute and Ping
- 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

Parts 1 and 2: Software Installation

Part 1: Installing Virtualbox

The first thing required to use Mininet is a Virtual Machine (VM) manager to run our Mininet VM. VirtualBox is free and open source and can be downloaded here. You are free to use kvm, vmware, or xen if you are familiar with those hypervisors - however, the TAs will not support them, so if you choose this route you are on your own for support.
Note : Special instructions for those with an M1 chip can be found here.

Are you having trouble installing VirtualBox? Make sure to consult the VirtualBox manual, canvas discussions, or the TAs. It is recommended that you **start the lab early** in case you encounter any problems setting up either VirtualBox or Mininet. If you do not have a computer, please contact the TAs and we can point you to some resources.

Common Problems Installing VirtualBox:
- Did you download the correct version? (32-bit vs 64-bit)
- Is your computer really old? It might not be able to use virtualization -- talk to the TA. - Make sure that virtualization is enabled in your BIOS.
- Do you have a M1 chip? -- check out the special instructions pdf and talk to a TA.

Part 2: Installing Mininet

Once VirtualBox is installed, you can install the Mininet VM. For this class, please use the Mininet VM available at this link. (You must use your UCSC Google account to get access.)

Using a GUI:
- Once the download is complete, open VirtualBox and select File>Import Appliance… Navigate to the

OVA file you downloaded.

- When the mininet VM has been successfully imported, start the VM.
- You should be presented with a GUI. Helpful tips:
    a. Chromium is a web browser. You can use this to go to the class webpage and copy/paste example code from the PDFs.
    b. You can use your Google Drive or VirtualBox shared folders to copy files to/from the VM. The TAs or student tutor can help you with this in lab.

Part 3: Experimenting with Mininet

A walkthrough can be found on the Mininet page. In this lab you are expected to spend some time familiarizing yourself with Mininet. First and foremost, here's some background and information on what Mininet is and why we are using it (we used to have the labs using actual routers and switches; since then we have migrated to virtualized labs).

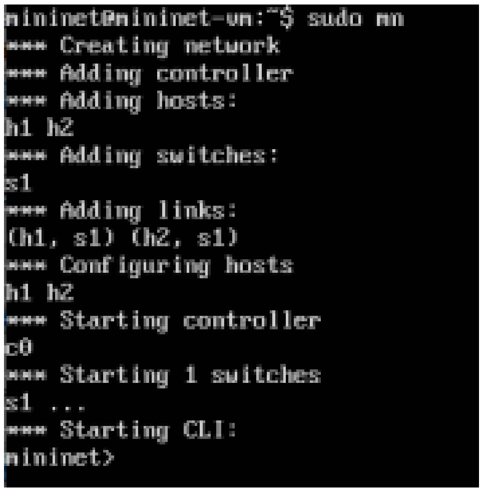
What is Mininet? Mininet is a "network in a box" tool developed at Stanford in 2010. It is designed to allow large scale networks to be emulated in software on a laptop. Its rise has also been dictated by the use of OpenFlow (which will be the subject of Lab 3 and the Final Project). If you are interested in reading more, here is the original Mininet paper.

Why Mininet? Our previous "physical" network lab had been aging from the wear and tear of years of usage (it was donated by Cisco in 2004). Mininet is probably one of the simplest forms of network emulators, is *free*, is open source, and is widely used by the research community, as well as by universities for teaching computer networks. It allows for more interesting topologies than what can be achieved in the physical lab. Most importantly, it has enabled us to accommodate CE 150's enrollment growth (only five years ago enrollment was around 40 students, while today it is peaking at 150). The physical lab only had 10 workstations, but over the years only fewer were operational. So in light of the growing student base we started looking towards a solution which scales with the number of students. Almost all students have access to a computer, so instead of buying hundreds of thousands of dollars worth of specialized equipment (e.g., routers, switches), we use general-purpose computers.

How does Mininet Work? Mininet works simply by creating a virtual network on your computer/laptop. It accomplishes this task by creating host namespaces (h1, h2, etc) and connecting them through virtual interfaces. So when we run the command *ping* between the linux namespaces h1 and h2, the ping will run from h1's namespace through a virtual interface pair created for h1 and h2, before it reaches h2. If h1 and h2 are connected through a switch as shown in the Python code in the Mininet walkthrough, the ping will transit multiple virtual interface pairs. The switches that we will be using are running OpenVSwitch (OVS). Mininet will connect additional virtual interfaces between each virtual port on the switch with each connected host. The host name space allows each host to see the same file system, but operates as its own process that will run separately from each of the other host processes. The OVS version running on the Ubuntu image supports OpenFlow.

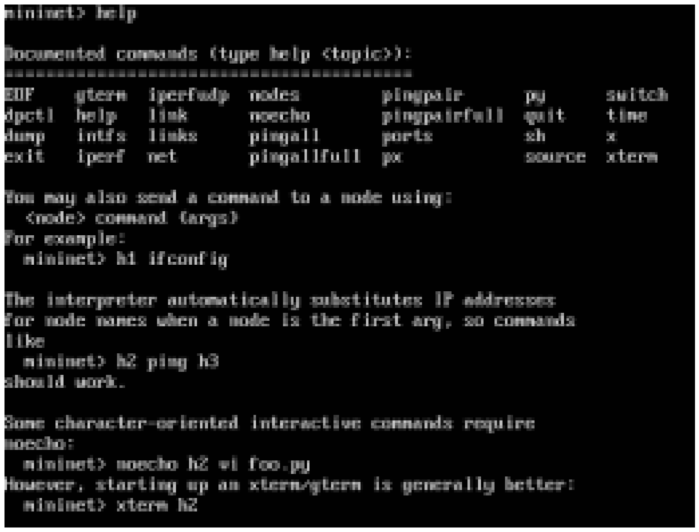
Understanding some Mininet commands: We recommend practicing with the following Mininet commands before you start the lab exercises:

1. ***sudo mn***: will start Mininet



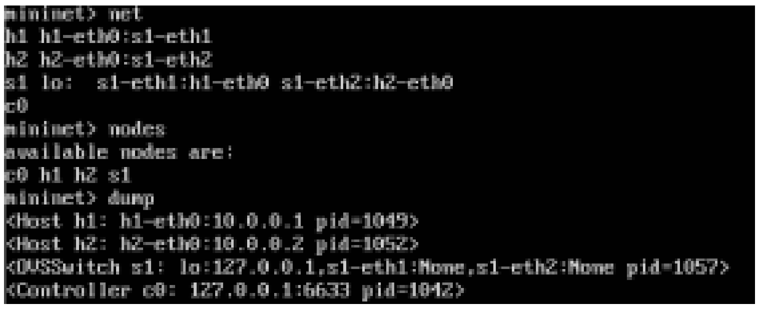
```
mininet@mininet-vm:~$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

2. You can use ***mn -h*** or type ***help*** after you have run Mininet.



```
mininet> help

Documented commands (type help <topic>):
========================================
EOF    gterm  iperfudp  nodes        pingpair      py      switch
dpctl  help   link      noecho       pingpairfull  quit    time
dump   intfs  links     pingall      ports         sh      x
exit   iperf  net       pingallfull  px            source  xterm

You may also send a command to a node using:
  <node> command {args}
For example:
  mininet> h1 ifconfig

The interpreter automatically substitutes IP addresses
for node names when a node is the first arg, so commands
like
  mininet> h2 ping h3
should work.

Some character-oriented interactive commands require
noecho:
  mininet> noecho h2 vi foo.py
However, starting up an xterm/gterm is generally better:
  mininet> xterm h2
```

3. We can see that when we launched Mininet above, a mini-network was created. a. The ***net*** command shows that *h1* indicates host 1 has one network interface *eth0,* which is connected to the switch on interface *eth1* (on the output line: *h1 h1-eth0:s1-eth1)*. There is also host 2 (*h2*), switch 1 (*s1*)*,* and controller 0 (*c0*).



```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=1049>
<Host h2: h2-eth0:10.0.0.2 pid=1052>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=1057>
<Controller c0: 127.0.0.1:6633 pid=1042>
```

b. From **dump** we can also see what IP addresses have been assigned to h1 and h2. We also are given the pids of each process. Processes in Mininet are used for hosts, switches, and controllers. Mininet is composed of processes using Interprocess Communication (IPC) to emulate a network environment.

4. ***sudo mn -c***: cleans the Mininet system. Use this if you are having errors to start Mininet, in case some problem from a previous execution has persisted.

[50 pts] Part 4: Building Your Own Network in Mininet

For all the questions below that require a screenshot, make sure that a timestamp with the date is visible next to your results (you can have a portion of another terminal open with the date command). No credit if a requested timestamp is not provided.

Running Mininet as a Python script:

To make custom topologies, it is useful to be able to refine a topology in a script. The following is an example of using a Python script to launch Mininet with a custom topology. You should use this as a skeleton for getting started on the lab exercises that follow. 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.

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.link import TCLink

class MyTopology(Topo):
    """
    A basic topology
    """
    def __init__(self):
        Topo.__init__(self)

        # Set Up Topology Here
        switch = self.addSwitch('s1') ## Adds a Switch

        host1 = self.addHost('h1') ## Adds a Host

        self.addLink(host1, switch) ## Add a link

if __name__ == '__main__':
    """
    If this script is run as an executable (by chmod +x), this is
    what it will do
    """

    topo = MyTopology() ## Creates the topology net =
    Mininet(topo=topo, link=TCLink) ## Loads the topology
    net.start() ## Starts Mininet

    # Commands here will run on the simulated topology
    CLI(net)

    net.stop() ## Stops Mininet
```

Mininet Exercises:

1. [4 pts] **Mininet Commands**

   a) What command do you use to find the interfaces of a host?

   Use the *net command.*

   b) What are the commands used to test connectivity between hosts and what is the difference between them?

   You can use *mininet> h1 ping -c 1 h2*
   Or *mininet> pingall*

   ***Pingall* does an all pairs ping while the first command requires you to manually input the hosts.**

   c) What command do you use to close Mininet?

   Use the *exit* command.

   d) What command would you use to run Wireshark?

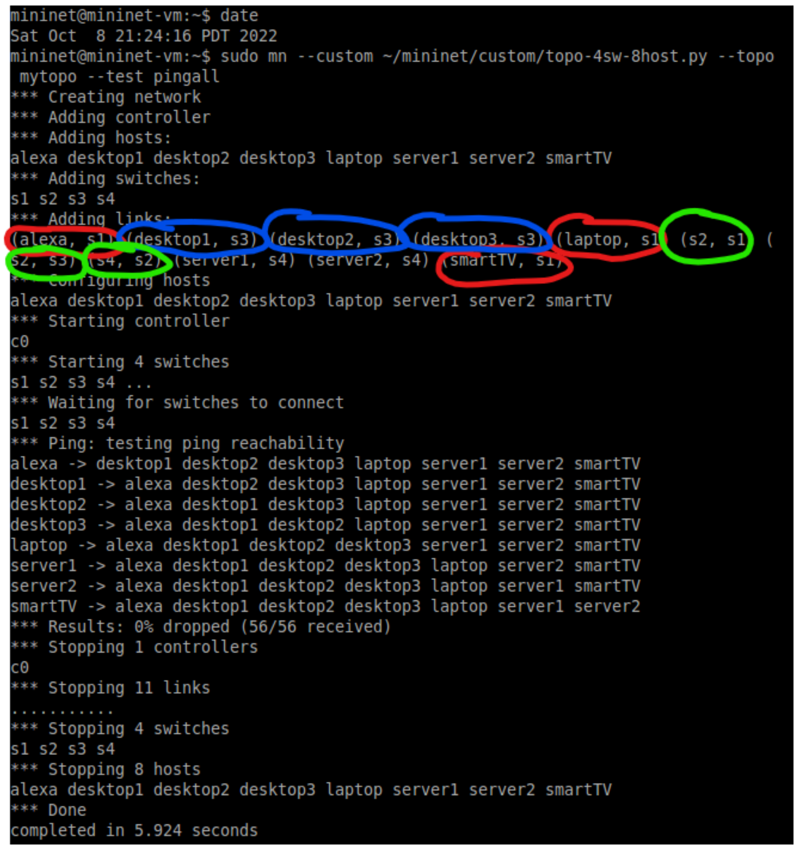
   Use the command *sudo wireshark &*

2. [8 pts] **Writing your own topology script**

   In Mininet change the default configuration through a Python script to create the topology in Figure 1. Provide a single screenshot with markup showing these connections. i. [2 pts] Server1 and Server2 are connected to Switch 4

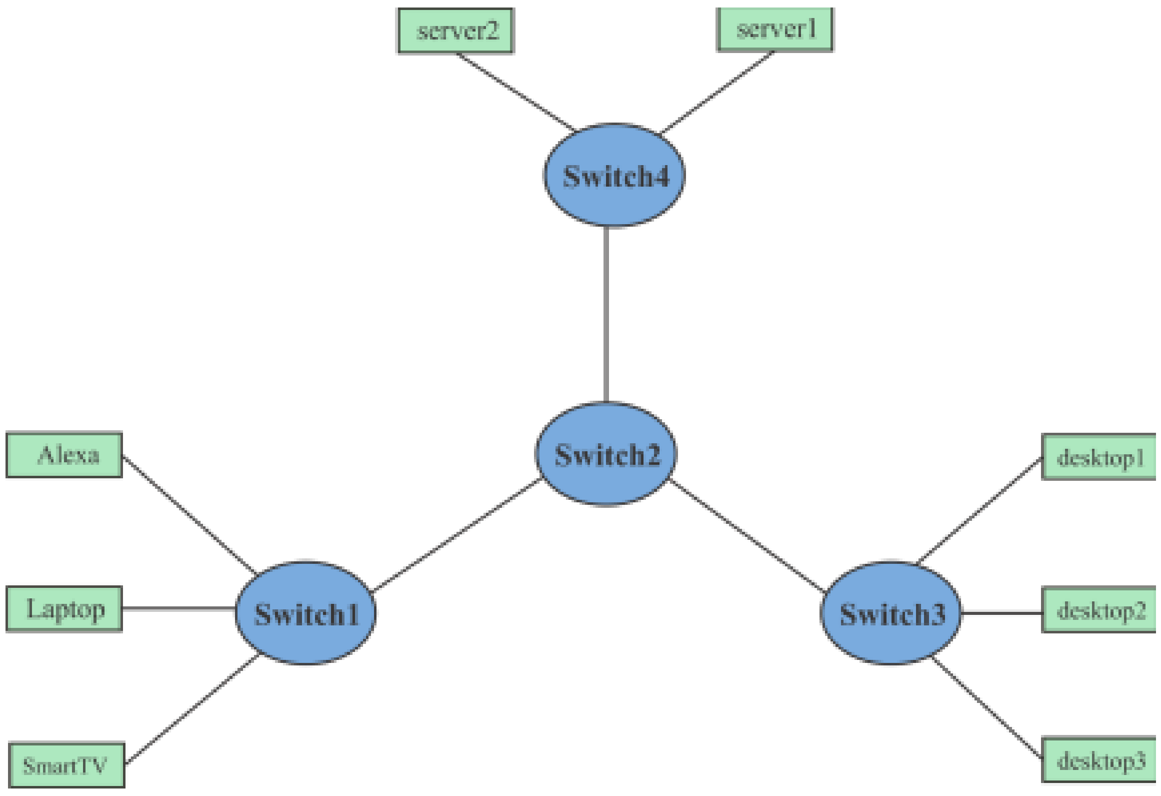
   ii. [2 pts] Alexa, Laptop and SmartTV are connected to Switch 1
   iii. [2 pts] desktop1, desktop2, and desktop3 are connected to Switch 3
   iv. [2 pts] Switch1, Switch3 and Switch4 are connected to Switch2

   SCREENSHOT BELOW



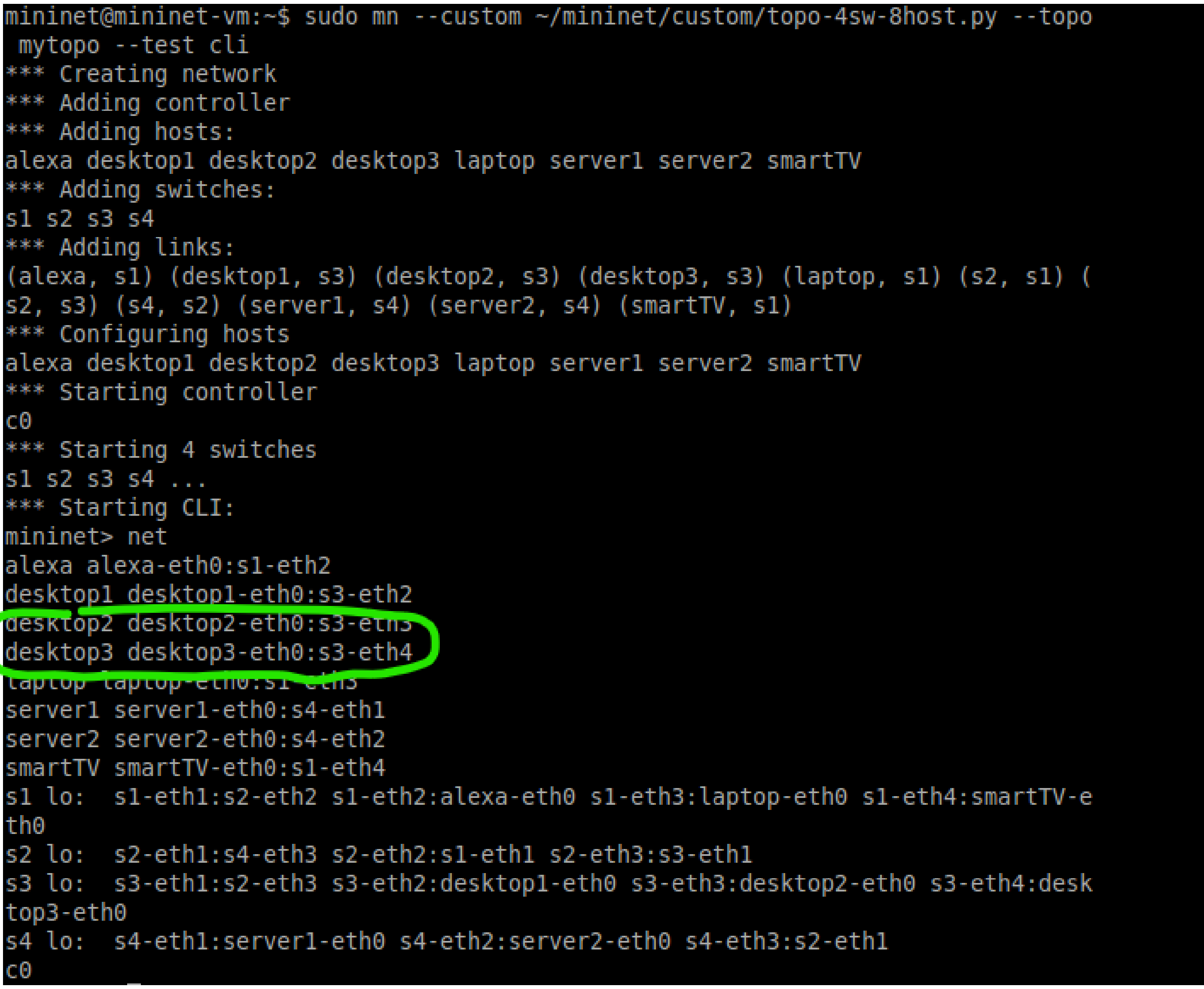
```
mininet@mininet-vm:~$ date
Sat Oct  8 21:24:16 PDT 2022
mininet@mininet-vm:~$ sudo mn --custom ~/mininet/custom/topo-4sw-8host.py --topo
 mytopo --test pingall
*** Creating network
*** Adding controller
*** Adding hosts:
alexa desktop1 desktop2 desktop3 laptop server1 server2 smartTV
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(alexa, s1) (desktop1, s3) (desktop2, s3) (desktop3, s3) (laptop, s1) (s2, s1) (
s2, s3) (s4, s2) (server1, s4) (server2, s4) (smartTV, s1)
*** Configuring hosts
alexa desktop1 desktop2 desktop3 laptop server1 server2 smartTV
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Waiting for switches to connect
s1 s2 s3 s4
*** Ping: testing ping reachability
alexa -> desktop1 desktop2 desktop3 laptop server1 server2 smartTV
desktop1 -> alexa desktop2 desktop3 laptop server1 server2 smartTV
desktop2 -> alexa desktop1 desktop3 laptop server1 server2 smartTV
desktop3 -> alexa desktop1 desktop2 laptop server1 server2 smartTV
laptop -> alexa desktop1 desktop2 desktop3 server1 server2 smartTV
server1 -> alexa desktop1 desktop2 desktop3 laptop server2 smartTV
server2 -> alexa desktop1 desktop2 desktop3 laptop server1 smartTV
smartTV -> alexa desktop1 desktop2 desktop3 laptop server1 server2
*** Results: 0% dropped (56/56 received)
*** Stopping 1 controllers
c0
*** Stopping 11 links
...........
*** Stopping 4 switches
s1 s2 s3 s4
*** Stopping 8 hosts
alexa desktop1 desktop2 desktop3 laptop server1 server2 smartTV
*** Done
completed in 5.924 seconds
```

Figure1:



3. [2 pts] **Verify the topology**
Run a command that will display the different devices and connections in Figure 1. (Hint: refer to the description given above in the introduction to Mininet) Save a screenshot of the output after running the command and highlight the connections of desktop2 and desktop3.

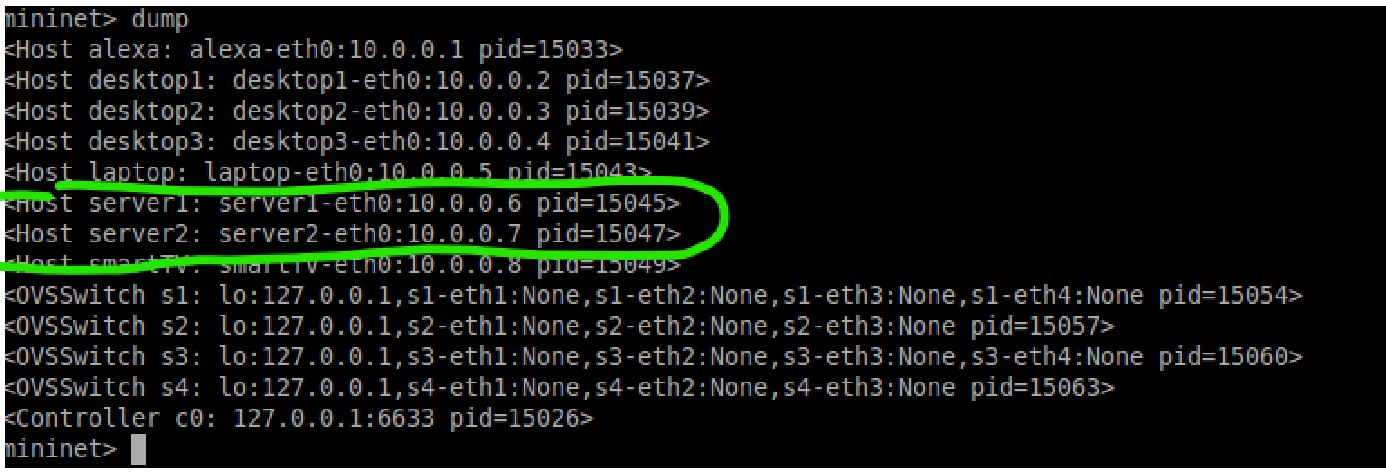
```
Sun Oct  9 00:11:09 PDT 2022
```



```
mininet@mininet-vm:~$ sudo mn --custom ~/mininet/custom/topo-4sw-8host.py --topo
 mytopo --test cli
*** Creating network
*** Adding controller
*** Adding hosts:
alexa desktop1 desktop2 desktop3 laptop server1 server2 smartTV
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(alexa, s1) (desktop1, s3) (desktop2, s3) (desktop3, s3) (laptop, s1) (s2, s1) (
s2, s3) (s4, s2) (server1, s4) (server2, s4) (smartTV, s1)
*** Configuring hosts
alexa desktop1 desktop2 desktop3 laptop server1 server2 smartTV
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet> net
alexa alexa-eth0:s1-eth2
desktop1 desktop1-eth0:s3-eth2
desktop2 desktop2-eth0:s3-eth3
desktop3 desktop3-eth0:s3-eth4
laptop laptop-eth0:s1-eth3
server1 server1-eth0:s4-eth1
server2 server2-eth0:s4-eth2
smartTV smartTV-eth0:s1-eth4
s1 lo:  s1-eth1:s2-eth2 s1-eth2:alexa-eth0 s1-eth3:laptop-eth0 s1-eth4:smartTV-e
th0
s2 lo:  s2-eth1:s4-eth3 s2-eth2:s1-eth1 s2-eth3:s3-eth1
s3 lo:  s3-eth1:s2-eth3 s3-eth2:desktop1-eth0 s3-eth3:desktop2-eth0 s3-eth4:desk
top3-eth0
s4 lo:  s4-eth1:server1-eth0 s4-eth2:server2-eth0 s4-eth3:s2-eth1
c0
```

4. [2 pts] **Check the IP Addresses**
Run a command that will display the IP address of ALL of the devices in Figure 1. Save a screenshot of the output after running the command and highlight the IP addresses of Server1 and Server2 in the screenshot.

```
Sun Oct  9 00:11:09 PDT 2022
```

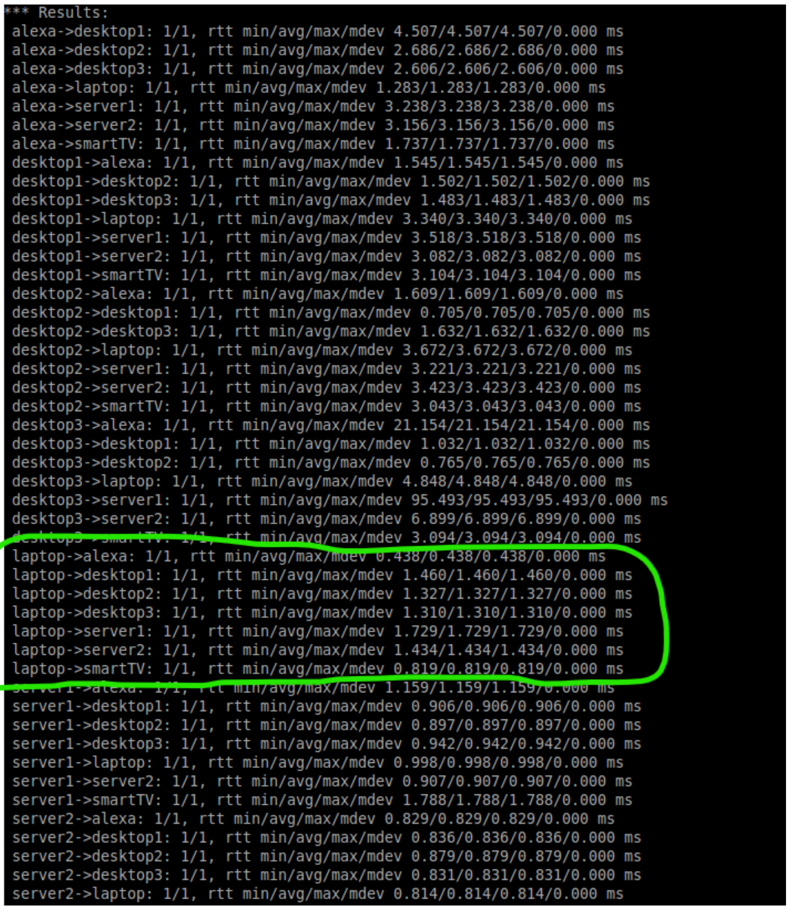


```
mininet> dump
<Host alexa: alexa-eth0:10.0.0.1 pid=15033>
<Host desktop1: desktop1-eth0:10.0.0.2 pid=15037>
<Host desktop2: desktop2-eth0:10.0.0.3 pid=15039>
<Host desktop3: desktop3-eth0:10.0.0.4 pid=15041>
<Host laptop: laptop-eth0:10.0.0.5 pid=15043>
<Host server1: server1-eth0:10.0.0.6 pid=15045>
<Host server2: server2-eth0:10.0.0.7 pid=15047>
<Host smartTV: smartTV-eth0:10.0.0.8 pid=15049>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None,s1-eth4:None pid=15054>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=15057>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None,s3-eth4:None pid=15060>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None pid=15063>
<Controller c0: 127.0.0.1:6633 pid=15026>
mininet> 
```

5. [4 pts] **Verifying connectivity**
Run the ***pingallfull*** command in the Mininet command line, and explain what you understand from the output of pingall. Save a screenshot of the output and highlight the row that represents the Laptop pinging all other devices in the network.

***Pingallfull* is testing the connectivity of all hosts by running a traceroute from each host to host. It then gives the RTT times of each connectivity.**

```
Sun Oct  9 00:11:09 PDT 2022
```

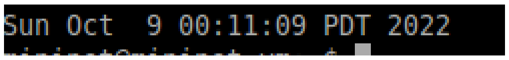


```
*** Results:
 alexa->desktop1: 1/1, rtt min/avg/max/mdev 4.507/4.507/4.507/0.000 ms
 alexa->desktop2: 1/1, rtt min/avg/max/mdev 2.686/2.686/2.686/0.000 ms
 alexa->desktop3: 1/1, rtt min/avg/max/mdev 2.606/2.606/2.606/0.000 ms
 alexa->laptop: 1/1, rtt min/avg/max/mdev 1.283/1.283/1.283/0.000 ms
 alexa->server1: 1/1, rtt min/avg/max/mdev 3.238/3.238/3.238/0.000 ms
 alexa->server2: 1/1, rtt min/avg/max/mdev 3.156/3.156/3.156/0.000 ms
 alexa->smartTV: 1/1, rtt min/avg/max/mdev 1.737/1.737/1.737/0.000 ms
 desktop1->alexa: 1/1, rtt min/avg/max/mdev 1.545/1.545/1.545/0.000 ms
 desktop1->desktop2: 1/1, rtt min/avg/max/mdev 1.502/1.502/1.502/0.000 ms
 desktop1->desktop3: 1/1, rtt min/avg/max/mdev 1.483/1.483/1.483/0.000 ms
 desktop1->laptop: 1/1, rtt min/avg/max/mdev 3.340/3.340/3.340/0.000 ms
 desktop1->server1: 1/1, rtt min/avg/max/mdev 3.518/3.518/3.518/0.000 ms
 desktop1->server2: 1/1, rtt min/avg/max/mdev 3.082/3.082/3.082/0.000 ms
 desktop1->smartTV: 1/1, rtt min/avg/max/mdev 3.104/3.104/3.104/0.000 ms
 desktop2->alexa: 1/1, rtt min/avg/max/mdev 1.609/1.609/1.609/0.000 ms
 desktop2->desktop1: 1/1, rtt min/avg/max/mdev 0.705/0.705/0.705/0.000 ms
 desktop2->desktop3: 1/1, rtt min/avg/max/mdev 1.632/1.632/1.632/0.000 ms
 desktop2->laptop: 1/1, rtt min/avg/max/mdev 3.672/3.672/3.672/0.000 ms
 desktop2->server1: 1/1, rtt min/avg/max/mdev 3.221/3.221/3.221/0.000 ms
 desktop2->server2: 1/1, rtt min/avg/max/mdev 3.423/3.423/3.423/0.000 ms
 desktop2->smartTV: 1/1, rtt min/avg/max/mdev 3.043/3.043/3.043/0.000 ms
 desktop3->alexa: 1/1, rtt min/avg/max/mdev 21.154/21.154/21.154/0.000 ms
 desktop3->desktop1: 1/1, rtt min/avg/max/mdev 1.032/1.032/1.032/0.000 ms
 desktop3->desktop2: 1/1, rtt min/avg/max/mdev 0.765/0.765/0.765/0.000 ms
 desktop3->laptop: 1/1, rtt min/avg/max/mdev 4.848/4.848/4.848/0.000 ms
 desktop3->server1: 1/1, rtt min/avg/max/mdev 95.493/95.493/95.493/0.000 ms
 desktop3->server2: 1/1, rtt min/avg/max/mdev 6.899/6.899/6.899/0.000 ms
 desktop3->smartTV: 1/1, rtt min/avg/max/mdev 3.094/3.094/3.094/0.000 ms
 laptop->alexa: 1/1, rtt min/avg/max/mdev 0.438/0.438/0.438/0.000 ms
 laptop->desktop1: 1/1, rtt min/avg/max/mdev 1.460/1.460/1.460/0.000 ms
 laptop->desktop2: 1/1, rtt min/avg/max/mdev 1.327/1.327/1.327/0.000 ms
 laptop->desktop3: 1/1, rtt min/avg/max/mdev 1.310/1.310/1.310/0.000 ms
 laptop->server1: 1/1, rtt min/avg/max/mdev 1.729/1.729/1.729/0.000 ms
 laptop->server2: 1/1, rtt min/avg/max/mdev 1.434/1.434/1.434/0.000 ms
 laptop->smartTV: 1/1, rtt min/avg/max/mdev 0.819/0.819/0.819/0.000 ms
 server1->alexa: 1/1, rtt min/avg/max/mdev 1.159/1.159/1.159/0.000 ms
 server1->desktop1: 1/1, rtt min/avg/max/mdev 0.906/0.906/0.906/0.000 ms
 server1->desktop2: 1/1, rtt min/avg/max/mdev 0.897/0.897/0.897/0.000 ms
 server1->desktop3: 1/1, rtt min/avg/max/mdev 0.942/0.942/0.942/0.000 ms
 server1->laptop: 1/1, rtt min/avg/max/mdev 0.998/0.998/0.998/0.000 ms
 server1->server2: 1/1, rtt min/avg/max/mdev 0.907/0.907/0.907/0.000 ms
 server1->smartTV: 1/1, rtt min/avg/max/mdev 1.788/1.788/1.788/0.000 ms
 server2->alexa: 1/1, rtt min/avg/max/mdev 0.829/0.829/0.829/0.000 ms
 server2->desktop1: 1/1, rtt min/avg/max/mdev 0.836/0.836/0.836/0.000 ms
 server2->desktop2: 1/1, rtt min/avg/max/mdev 0.879/0.879/0.879/0.000 ms
 server2->desktop3: 1/1, rtt min/avg/max/mdev 0.831/0.831/0.831/0.000 ms
 server2->laptop: 1/1, rtt min/avg/max/mdev 0.814/0.814/0.814/0.000 ms
```

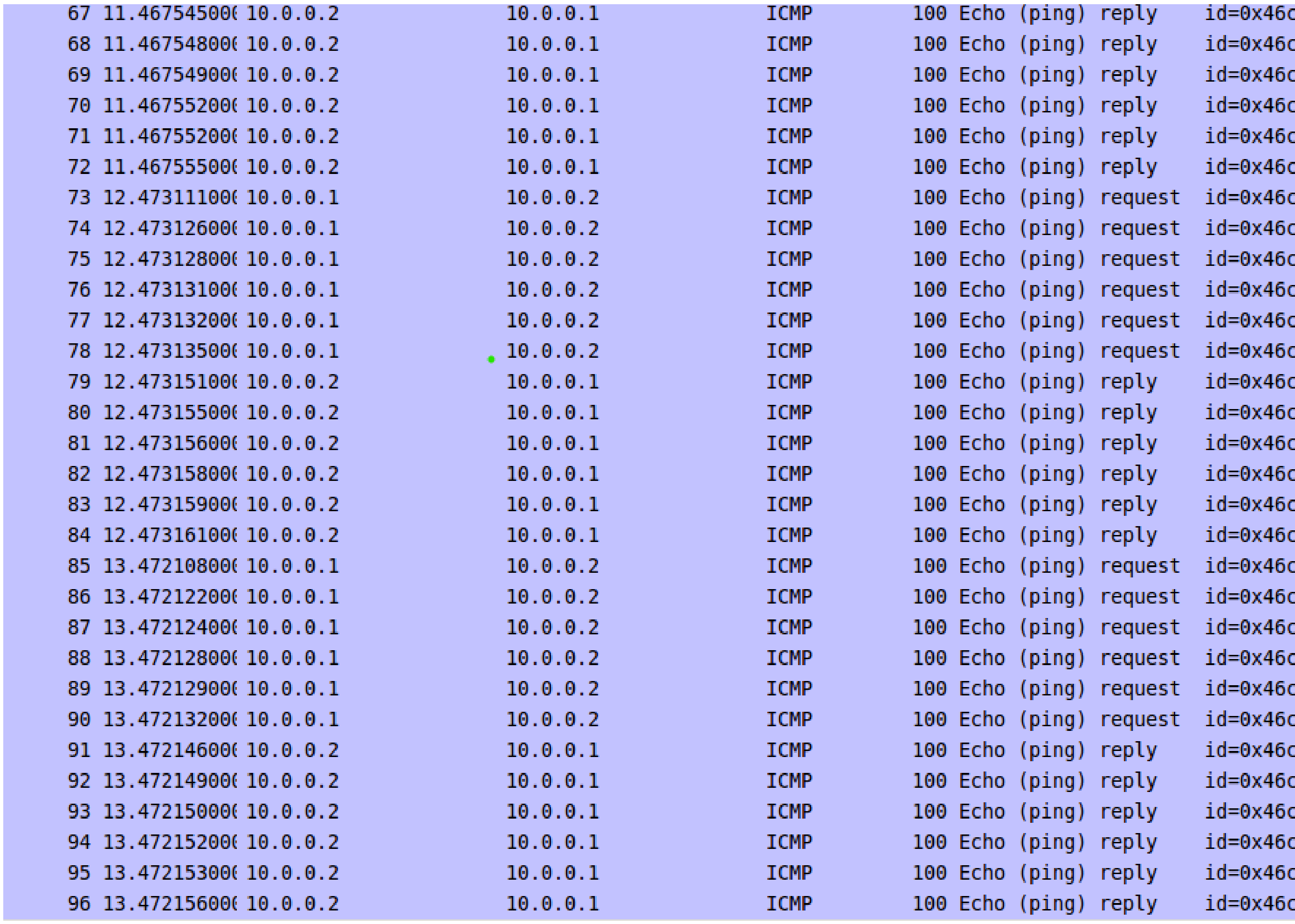
6. [16 pts] **Ping: Wireshark Analysis for topology in Figure 1**
Note: Run Wireshark as "sudo wireshark" and choose the "any" interface to capture on. a. [4 pts] What is ICMP? Use your own words - no copy/paste for credit. (Your textbook has a good small section on ICMP.)

ICMP stands for Internet Control Message Protocol and it is used by network devices to report on connectivity issues. It is used in routers to send error messages such as if a a service is unavailable or a host could not be reached.

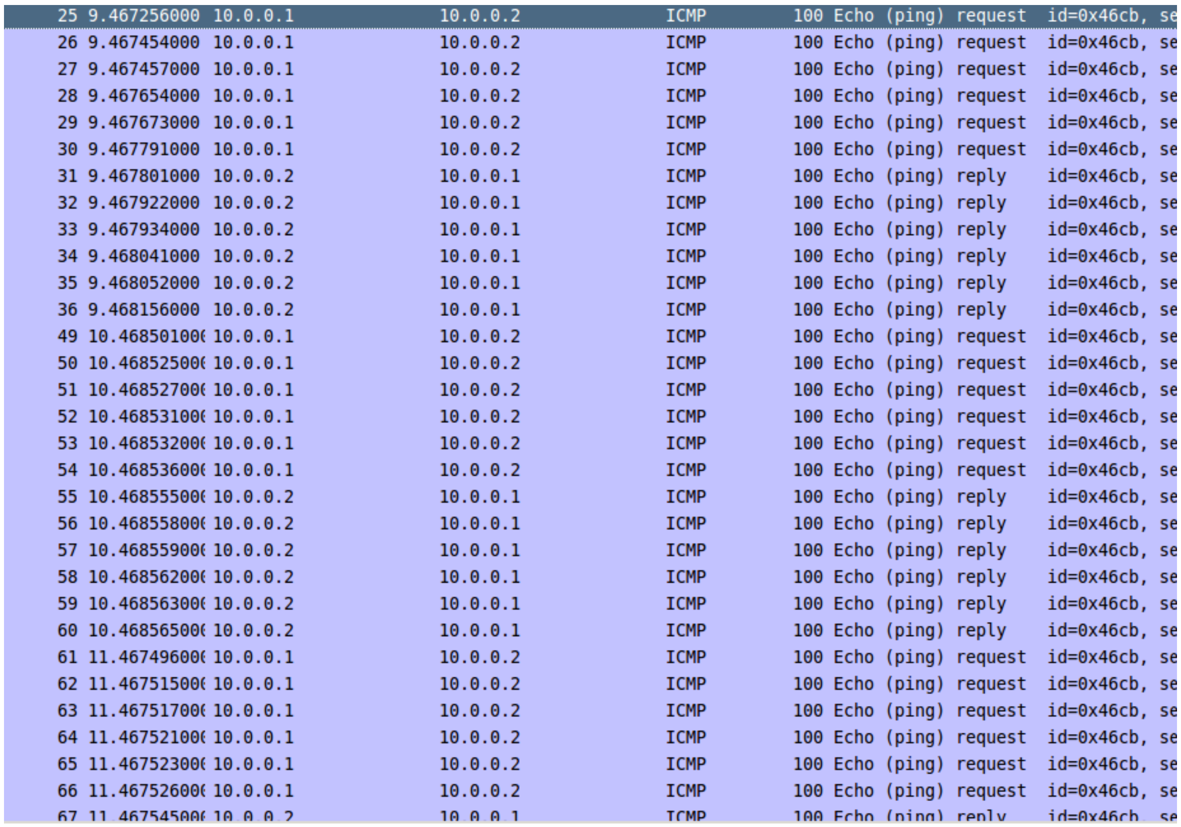
b. [4 pts] Run Ping from a host to any other host using `X ping -c 5 Y` (replace X and Y with the names of the source host and destination host respectively). With the help of the display filter, filter to show only ICMP packets, find the packets which were sent and received by Ping and highlight them in your screenshot.



```
Sun Oct  9 00:11:09 PDT 2022
```



```
   67 11.46754500( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   68 11.46754800( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   69 11.46754900( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   70 11.46755200( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   71 11.46755200( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   72 11.46755500( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   73 12.47311100( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46c
   74 12.47312600( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46c
   75 12.47312800( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46c
   76 12.47313100( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46c
   77 12.47313200( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46c
   78 12.47313500( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46c
   79 12.47315100( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   80 12.47315500( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   81 12.47315600( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   82 12.47315800( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   83 12.47315900( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   84 12.47316100( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   85 13.47210800( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46c
   86 13.47212200( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46c
   87 13.47212400( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46c
   88 13.47212800( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46c
   89 13.47212900( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46c
   90 13.47213200( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46c
   91 13.47214600( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   92 13.47214900( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   93 13.47215000( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   94 13.47215200( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   95 13.47215300( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
   96 13.47215600( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46c
```



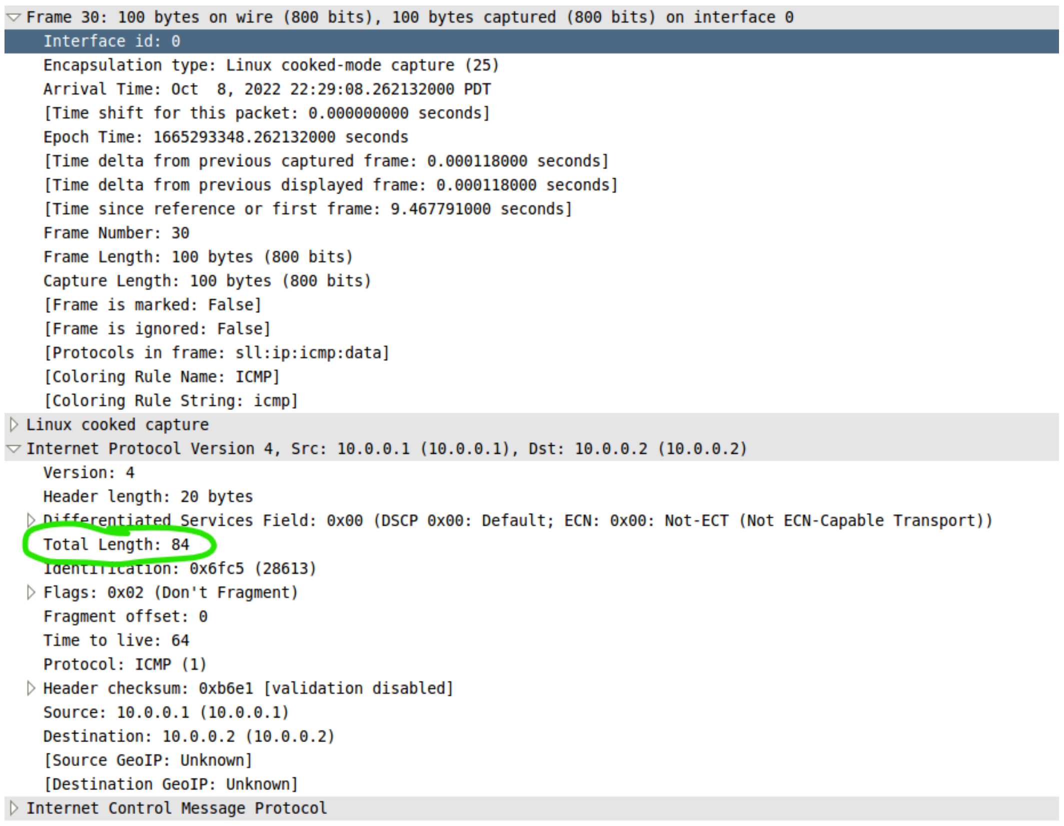
```
   25 9.467256000  10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   26 9.467454000  10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   27 9.467457000  10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   28 9.467654000  10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   29 9.467673000  10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   30 9.467791000  10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   31 9.467801000  10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46cb, se
   32 9.467922000  10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46cb, se
   33 9.467934000  10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46cb, se
   34 9.468041000  10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46cb, se
   35 9.468052000  10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46cb, se
   36 9.468156000  10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46cb, se
   49 10.46850100( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   50 10.46852500( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   51 10.46852700( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   52 10.46853100( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   53 10.46853200( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   54 10.46853600( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   55 10.46855500( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46cb, se
   56 10.46855800( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46cb, se
   57 10.46855900( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46cb, se
   58 10.46856200( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46cb, se
   59 10.46856300( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46cb, se
   60 10.46856500( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46cb, se
   61 11.46749600( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   62 11.46751500( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   63 11.46751700( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   64 11.46752100( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   65 11.46752300( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   66 11.46752600( 10.0.0.1            10.0.0.2            ICMP        100 Echo (ping) request  id=0x46cb, se
   67 11.46754500( 10.0.0.2            10.0.0.1            ICMP        100 Echo (ping) reply    id=0x46cb, se
```

c. [4 pts] Discuss the 2 types of ICMP messages that Ping is using (as highlighted in (b)) and briefly describe how the Ping application operates using these 2 messages.

The 2 types of ICMP messages that Ping is using are request and reply. The ping operation uses "request" to request a connectivity test to the source destination. It then replies to the request with a response time that measures if there is a healthy connection between the hosts.

d. [4 pts] In Wireshark, drill into a single ICMP echo request. Mark the request you are examining in your screenshot. What is the total length of the transmission frame? Include a screenshot of the drilldown that shows the details of this packet, particularly its total length.

```
Sun Oct  9 00:11:09 PDT 2022
```



```
▽ Frame 30: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface 0
    Interface id: 0
    Encapsulation type: Linux cooked-mode capture (25)
    Arrival Time: Oct  8, 2022 22:29:08.262132000 PDT
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1665293348.262132000 seconds
    [Time delta from previous captured frame: 0.000118000 seconds]
    [Time delta from previous displayed frame: 0.000118000 seconds]
    [Time since reference or first frame: 9.467791000 seconds]
    Frame Number: 30
    Frame Length: 100 bytes (800 bits)
    Capture Length: 100 bytes (800 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: sll:ip:icmp:data]
    [Coloring Rule Name: ICMP]
    [Coloring Rule String: icmp]
▷ Linux cooked capture
▽ Internet Protocol Version 4, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.2 (10.0.0.2)
    Version: 4
    Header length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 84
    Identification: 0x6fc5 (28613)
  ▷ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
  ▷ Header checksum: 0xb6e1 [validation disabled]
    Source: 10.0.0.1 (10.0.0.1)
    Destination: 10.0.0.2 (10.0.0.2)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
▷ Internet Control Message Protocol
```

7. [14pts] Exploring Propagation Delay

Make a table highlighting the <u>average</u> RTT reported by `ping` with respect to the link delay set in Mininet as described in (a)(b)(c) below. A sample table:

Node pair	Average RTT with delays = 20ms
Laptop to server1	184.895 millseconds

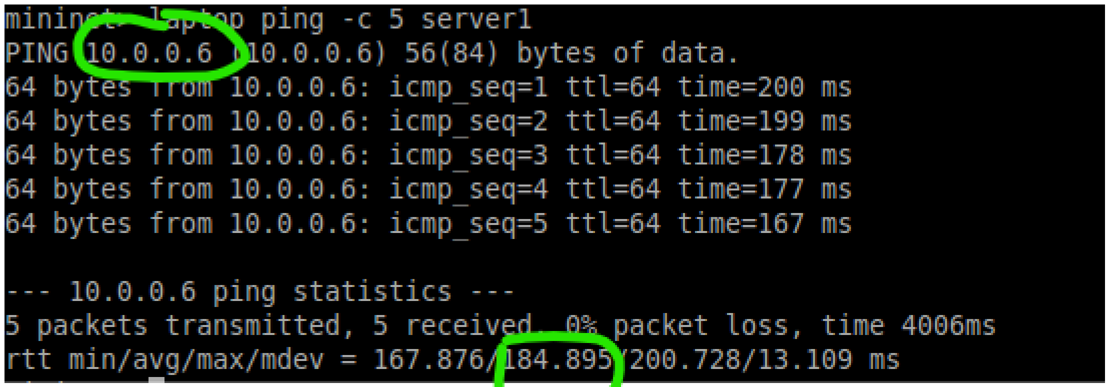
SmartTV to Laptop	89.96 milliseconds

Make the following changes to the topology in Figure 1:

Set all links to have a link delay of 20ms.

a. [3pts] Run `Laptop ping -c 5 server1`. What is the IP address of server1? Take a screenshot of your results and highlight the average delay and IP addresses of the Laptop and Server.

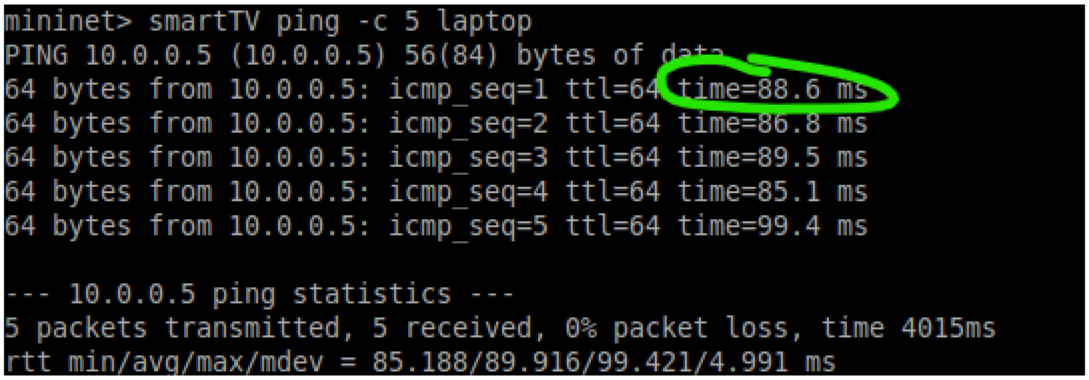
The IP address of server1 is 10.0.0.6. The IP address of the laptop is 10.0.0.5



```
mininet> laptop ping -c 5 server1
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data.
64 bytes from 10.0.0.6: icmp_seq=1 ttl=64 time=200 ms
64 bytes from 10.0.0.6: icmp_seq=2 ttl=64 time=199 ms
64 bytes from 10.0.0.6: icmp_seq=3 ttl=64 time=178 ms
64 bytes from 10.0.0.6: icmp_seq=4 ttl=64 time=177 ms
64 bytes from 10.0.0.6: icmp_seq=5 ttl=64 time=167 ms

--- 10.0.0.6 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 167.876/184.895/200.728/13.109 ms
```

b. [3pts] Run `SmartTV ping -c 5 Laptop`. What could be some reasons that cause a variation in the different ICMP messages sent out in *ping*? Take a screenshot of your results and highlight the RTT for the first ICMP message sent.

The reason the ICMP messages are different is because SmartTv and Laptop are connected to the same switch, while laptop has to go through two other switches to connect with server1.

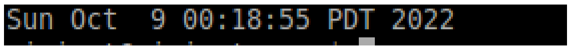
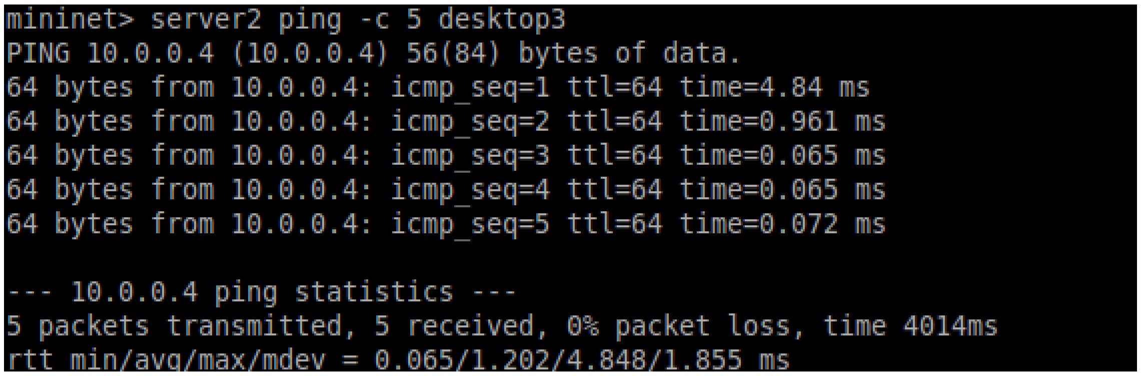


```
mininet> smartTV ping -c 5 laptop
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=88.6 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=86.8 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=89.5 ms
64 bytes from 10.0.0.5: icmp_seq=4 ttl=64 time=85.1 ms
64 bytes from 10.0.0.5: icmp_seq=5 ttl=64 time=99.4 ms

--- 10.0.0.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4015ms
rtt min/avg/max/mdev = 85.188/89.916/99.421/4.991 ms
```

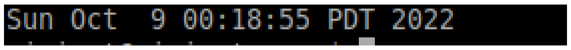
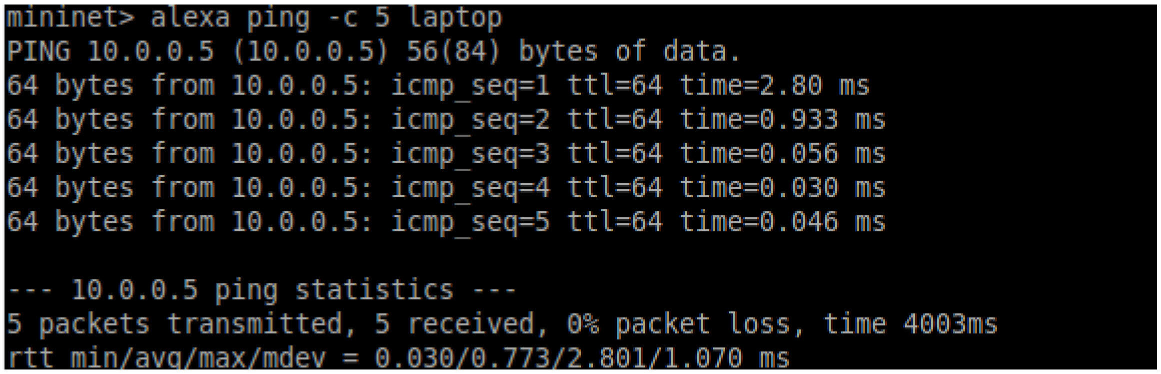
Change the link delay between desktop 3 and switch3 to 50ms.

Change the link delay between server1 and switch4 to 35ms.

c. [4pts] Run `server2 ping -c 5 desktop3`. Do you notice any changes? Take a screenshot of your results and highlight the changes if any, and explain in support of your answer.


```
Sun Oct  9 00:18:55 PDT 2022
```

```
mininet> server2 ping -c 5 desktop3
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=4.84 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.961 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.065 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.065 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.072 ms

--- 10.0.0.4 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4014ms
rtt min/avg/max/mdev = 0.065/1.202/4.848/1.855 ms
```

d. [4pts] Run `Alexa ping -c 5 Laptop`. Do you notice any changes? Explain in support of your answer. Take a screenshot of your results and mark references in your screenshot.


```
Sun Oct  9 00:18:55 PDT 2022
```

```
mininet> alexa ping -c 5 laptop
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=2.80 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=0.933 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=0.056 ms
64 bytes from 10.0.0.5: icmp_seq=4 ttl=64 time=0.030 ms
64 bytes from 10.0.0.5: icmp_seq=5 ttl=64 time=0.046 ms

--- 10.0.0.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4003ms
rtt min/avg/max/mdev = 0.030/0.773/2.801/1.070 ms
```

Part B: Deep Dive into Traceroute Using Wireshark over the Internet

Traceroute Background:

Recall that traceroute operates by sending a series of probes along the path from a source to a destination in this way:

- It first sends one or more (usually 3) datagrams with the time-to-live (TTL) field in the IP header set to 1
- It then sends one or more datagrams towards the same destination with a TTL value of 2 ● It then sends one or more datagrams datagrams towards the same destination with a TTL value of 3 ● and so on… until the final destination is reached

Recall that every router on the path decrements the TTL in each received datagram by 1 (actually, RFC 791 says that the router must decrement the TTL by *at least* one). Upon receiving the packet with a TTL=0, the router returns an ICMP message (type 11 – **TTL-exceeded**) to the sending host. (You are going to see these messages in Wireshark!)

As a result of this behavior, a datagram with a TTL of 1 (sent by the host executing traceroute) only travel one hop and the router one hop away from the sender will send an ICMP TTL-exceeded message back to the sender; the datagram sent with a TTL of 2 will cause the router two hops away to send an ICMP message back to the sender; the datagram sent with a TTL of 3 will cause the router three hops away to send an ICMP message back to the sender; and so on... In this manner, the host executing traceroute can learn the IP addresses of the routers between itself and the destination by looking at the source IP addresses in the datagrams containing the ICMP TTL-exceeded messages. The Traceroute program nicely displays that information for you in its output.

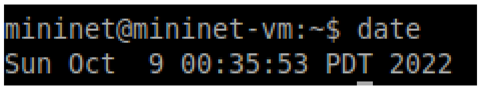
In this lab you will look at these Traceroute probes sent by the sender and the TTL-Expired messages returned by the Receiver as they are displayed in Wireshark. You will identify the probes with the incrementing TTL values and the corresponding TTL-exceeded ICMP messages that are returned in response.

[50 pts] Traceroute Exercises - **All exercises are to be run on your Virtual Machine**

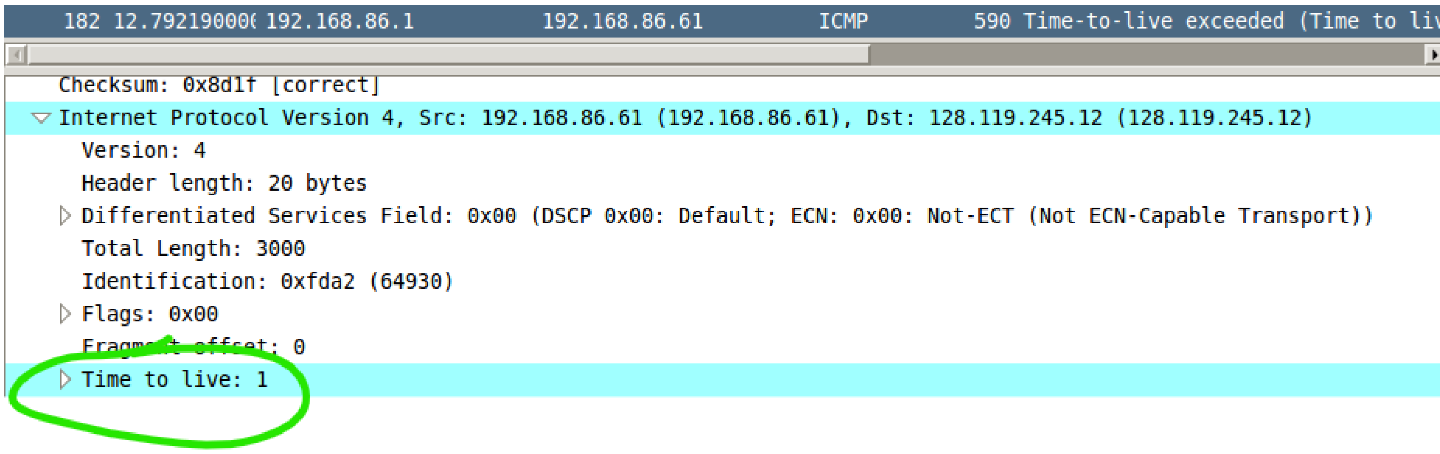
8. [15 pts] **Warmup: Search for Traceroute Probes**
Start up a Wireshark capture using the interface *eth0*. Then run the following command in a terminal:
`traceroute gaia.cs.umass.edu`

a) Find **one** of the exchanges where the ICMP Message **TTL-Exceeded** is received by the client (the traceroute process running on your VM). Then find the probe sent from your VM that immediately preceded TTL exceeded, *i.e.*, the probe that caused TTL-Exceeded. .

What was the value for TTL in the probe packet (sent from client→server)? (Hint: you can find the TTL field if you drill into the IP header). Include a screenshot with the 2 packets and the TTL of the probe packet highlighted.

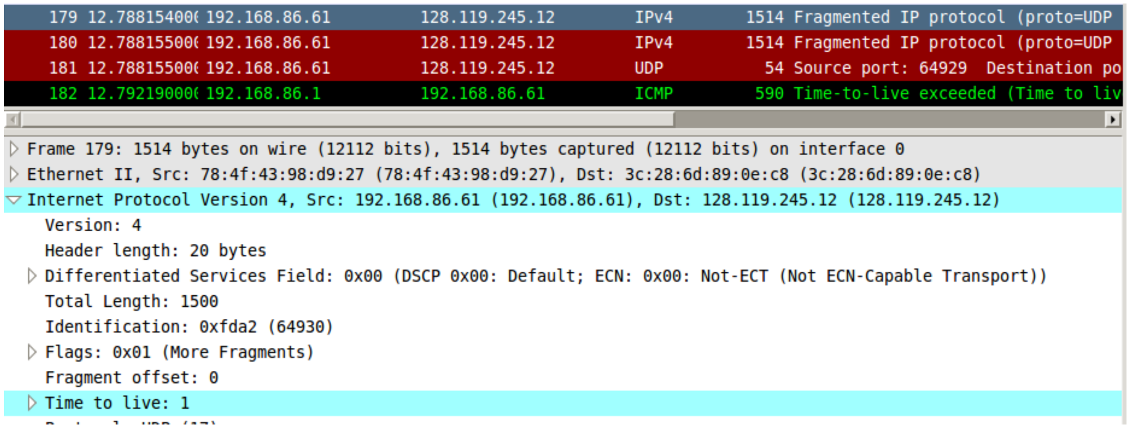


```
mininet@mininet-vm:~$ date
Sun Oct  9 00:35:53 PDT 2022
```



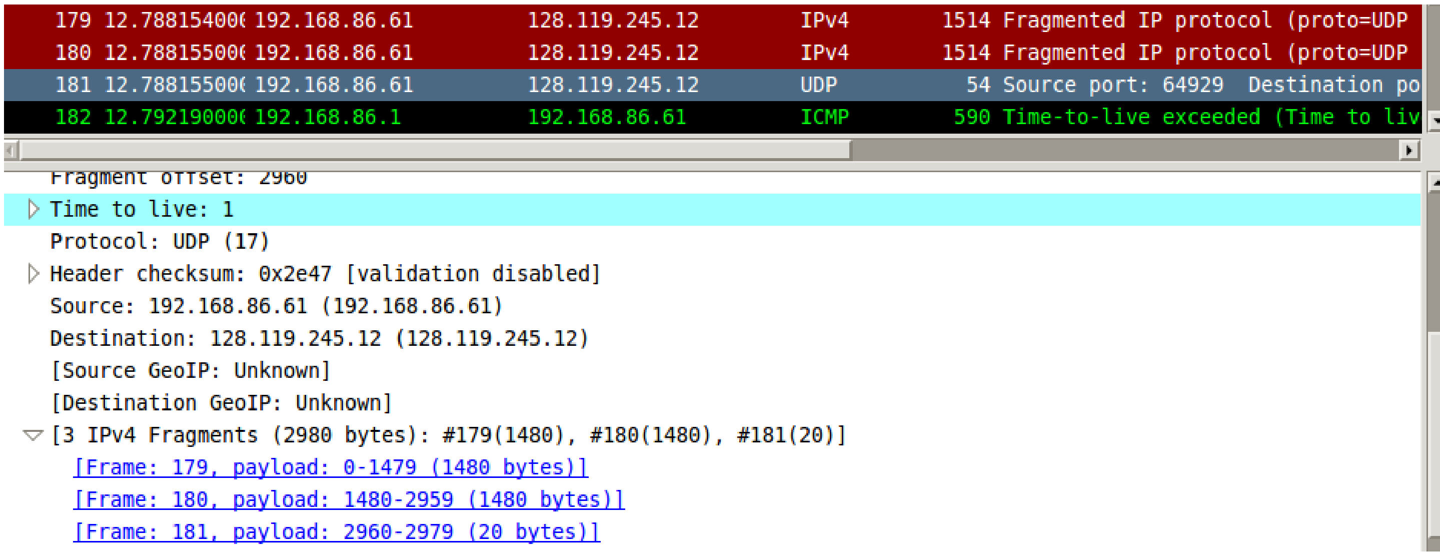
```
182 12.79219000( 192.168.86.1          192.168.86.61         ICMP        590 Time-to-live exceeded (Time to liv
```

```
Checksum: 0x8d1f [correct]
▽ Internet Protocol Version 4, Src: 192.168.86.61 (192.168.86.61), Dst: 128.119.245.12 (128.119.245.12)
    Version: 4
    Header length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 3000
    Identification: 0xfda2 (64930)
  ▷ Flags: 0x00
    Fragment offset: 0
  ▷ Time to live: 1
```



```
179 12.78815400( 192.168.86.61    128.119.245.12    IPv4    1514 Fragmented IP protocol (proto=UDP
180 12.78815500( 192.168.86.61    128.119.245.12    IPv4    1514 Fragmented IP protocol (proto=UDP
181 12.78815500( 192.168.86.61    128.119.245.12    UDP       54 Source port: 64929  Destination po
182 12.79219000( 192.168.86.1     192.168.86.61     ICMP     590 Time-to-live exceeded (Time to liv
```

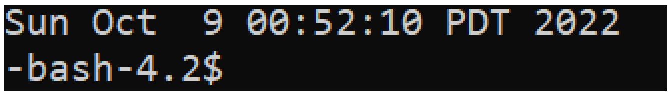
```
▷ Frame 179: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
▷ Ethernet II, Src: 78:4f:43:98:d9:27 (78:4f:43:98:d9:27), Dst: 3c:28:6d:89:0e:c8 (3c:28:6d:89:0e:c8)
▽ Internet Protocol Version 4, Src: 192.168.86.61 (192.168.86.61), Dst: 128.119.245.12 (128.119.245.12)
    Version: 4
    Header length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 1500
    Identification: 0xfda2 (64930)
  ▷ Flags: 0x01 (More Fragments)
    Fragment offset: 0
  ▷ Time to live: 1
```



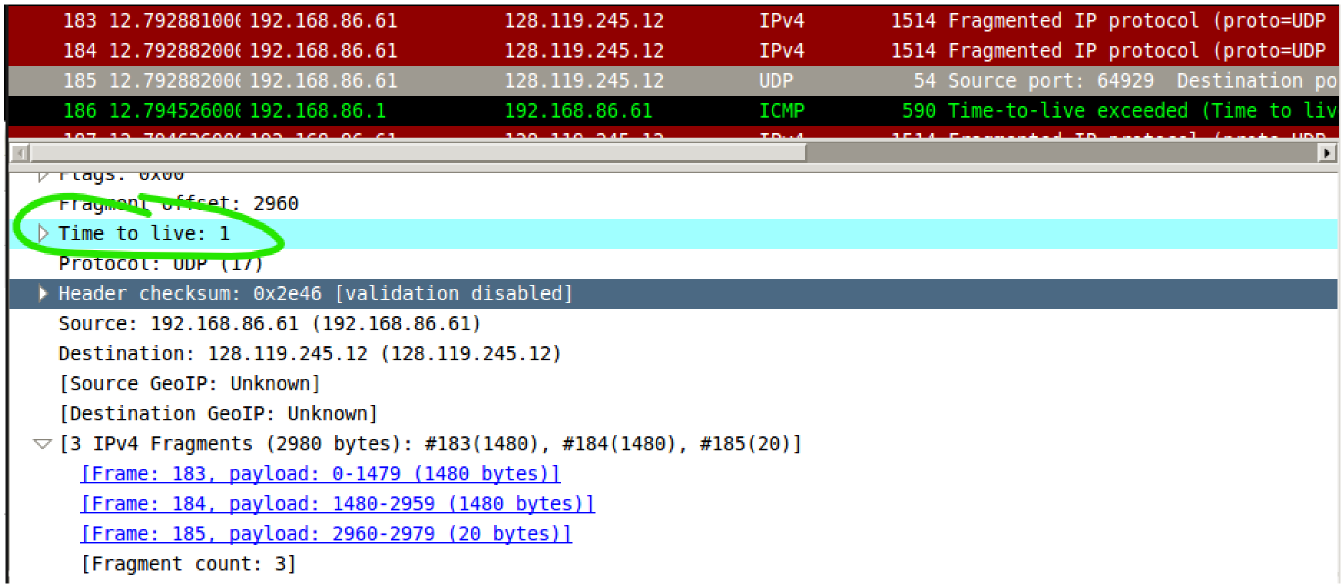
```
179 12.78815400( 192.168.86.61    128.119.245.12    IPv4    1514 Fragmented IP protocol (proto=UDP
180 12.78815500( 192.168.86.61    128.119.245.12    IPv4    1514 Fragmented IP protocol (proto=UDP
181 12.78815500( 192.168.86.61    128.119.245.12    UDP       54 Source port: 64929  Destination po
182 12.79219000( 192.168.86.1     192.168.86.61     ICMP     590 Time-to-live exceeded (Time to liv
```

```
    Fragment offset: 2960
  ▷ Time to live: 1
    Protocol: UDP (17)
  ▷ Header checksum: 0x2e47 [validation disabled]
    Source: 192.168.86.61 (192.168.86.61)
    Destination: 128.119.245.12 (128.119.245.12)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
  ▽ [3 IPv4 Fragments (2980 bytes): #179(1480), #180(1480), #181(20)]
      [Frame: 179, payload: 0-1479 (1480 bytes)]
      [Frame: 180, payload: 1480-2959 (1480 bytes)]
      [Frame: 181, payload: 2960-2979 (20 bytes)]
```

b) Now find the very NEXT Traceroute probe sent by the Client.
What is the value for TTL in the corresponding IP packet?
How many probes do you see with this TTL in total?
Include a screenshot with the TTL value highlighted and discuss your observations.



```
Sun Oct  9 00:52:10 PDT 2022
-bash-4.2$
```



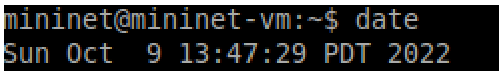
```
183 12.79288100( 192.168.86.61      128.119.245.12      IPv4      1514 Fragmented IP protocol (proto=UDP
184 12.79288200( 192.168.86.61      128.119.245.12      IPv4      1514 Fragmented IP protocol (proto=UDP
185 12.79288200( 192.168.86.61      128.119.245.12      UDP         54 Source port: 64929  Destination po
186 12.79452600( 192.168.86.1       192.168.86.61       ICMP       590 Time-to-live exceeded (Time to liv
```

```
  Flags: 0x00
  Fragment offset: 2960
▷ Time to live: 1
  Protocol: UDP (17)
▶ Header checksum: 0x2e46 [validation disabled]
  Source: 192.168.86.61 (192.168.86.61)
  Destination: 128.119.245.12 (128.119.245.12)
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
▽ [3 IPv4 Fragments (2980 bytes): #183(1480), #184(1480), #185(20)]
    [Frame: 183, payload: 0-1479 (1480 bytes)]
    [Frame: 184, payload: 1480-2959 (1480 bytes)]
    [Frame: 185, payload: 2960-2979 (20 bytes)]
    [Fragment count: 3]
```

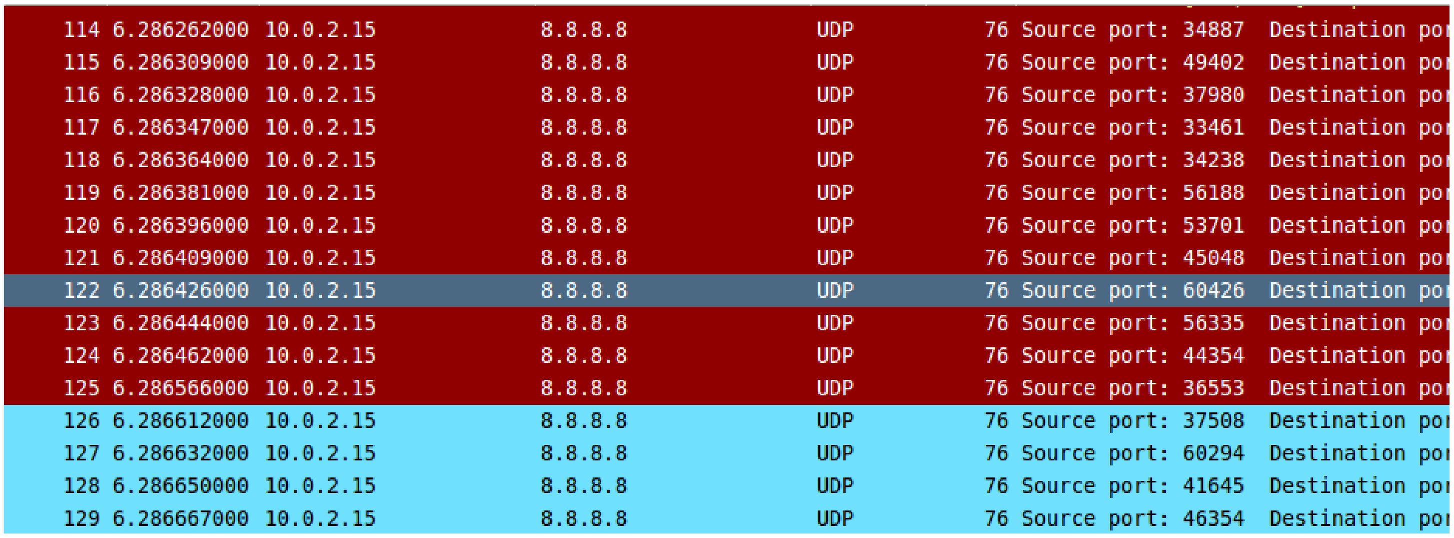
9. [10 pts] **Discover the protocol for Traceroute Probes**

On your Mininet VM, run Traceroute to a destination of your choice that completes with just a few *** in the output lines (or none if possible - not necessary). Do NOT use any Traceroute flags on the command line. Use Wireshark to determine if Traceroute is sending its probes using UDP or ICMP Echo Requests. Which protocol is Traceroute using for its probe packets? Include a screenshot of the packets you have captured that lead to your conclusion.

Tracerpute is sending its probes using UDP



```
mininet@mininet-vm:~$ date
Sun Oct  9 13:47:29 PDT 2022
```



```
114 6.286262000  10.0.2.15      8.8.8.8       UDP      76 Source port: 34887  Destination po
115 6.286309000  10.0.2.15      8.8.8.8       UDP      76 Source port: 49402  Destination po
116 6.286328000  10.0.2.15      8.8.8.8       UDP      76 Source port: 37980  Destination po
117 6.286347000  10.0.2.15      8.8.8.8       UDP      76 Source port: 33461  Destination po
118 6.286364000  10.0.2.15      8.8.8.8       UDP      76 Source port: 34238  Destination po
119 6.286381000  10.0.2.15      8.8.8.8       UDP      76 Source port: 56188  Destination po
120 6.286396000  10.0.2.15      8.8.8.8       UDP      76 Source port: 53701  Destination po
121 6.286409000  10.0.2.15      8.8.8.8       UDP      76 Source port: 45048  Destination po
122 6.286426000  10.0.2.15      8.8.8.8       UDP      76 Source port: 60426  Destination po
123 6.286444000  10.0.2.15      8.8.8.8       UDP      76 Source port: 56335  Destination po
124 6.286462000  10.0.2.15      8.8.8.8       UDP      76 Source port: 44354  Destination po
125 6.286566000  10.0.2.15      8.8.8.8       UDP      76 Source port: 36553  Destination po
126 6.286612000  10.0.2.15      8.8.8.8       UDP      76 Source port: 37508  Destination po
127 6.286632000  10.0.2.15      8.8.8.8       UDP      76 Source port: 60294  Destination po
128 6.286650000  10.0.2.15      8.8.8.8       UDP      76 Source port: 41645  Destination po
129 6.286667000  10.0.2.15      8.8.8.8       UDP      76 Source port: 46354  Destination po
```

10. [10 pts] **Switch the Protocol**

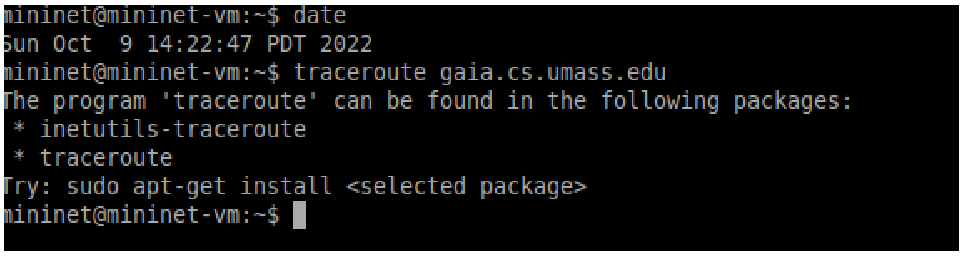
Now run Traceroute to the same destination used in (9), but force Traceroute to change the protocol it uses for its probes, e.g., if in (9) Traceroute was using UDP, force it to use ICMP and vice versa. (Hint: there is a flag for Traceroute to specify the desired protocol. Check the man page for the flags).

What command did you use to change the protocol?

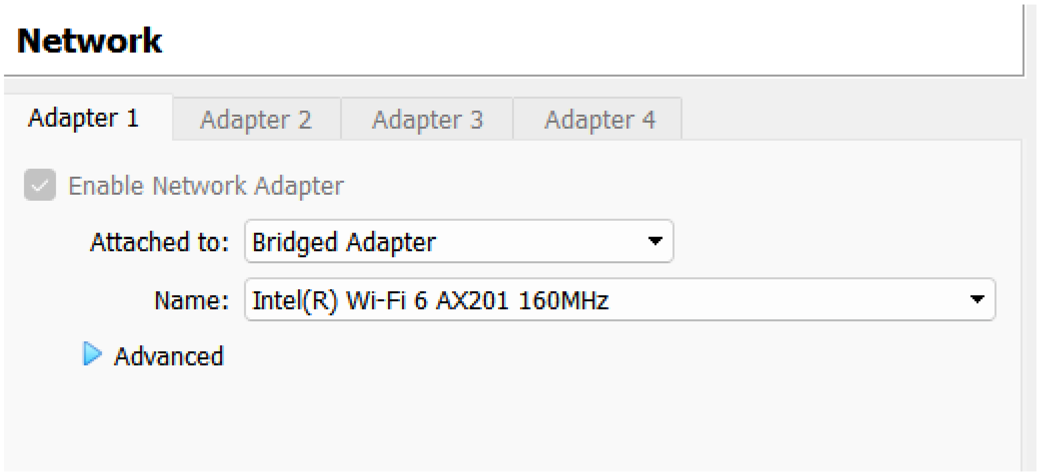
Use traceroute -I for ICMP Echo probes

Use Wireshark to show that Traceroute is using the forced protocol. Include screenshots with packets circled or highlighted that verify Traceroute is using a different protocol for its probes.

The *traceroute* command does not work for me. I tried changing the network to bridged and that still didn't work. Not sure what the workaround for this problem is.



```
mininet@mininet-vm:~$ date
Sun Oct  9 14:22:47 PDT 2022
mininet@mininet-vm:~$ traceroute gaia.cs.umass.edu
The program 'traceroute' can be found in the following packages:
 * inetutils-traceroute
 * traceroute
Try: sudo apt-get install <selected package>
mininet@mininet-vm:~$
```



Network

Enable Network Adapter: ✓

Attached to: Bridged Adapter

Name: Intel(R) Wi-Fi 6 AX201 160MHz

11. [10 pts] **Checking out the IP packets (Layer 3 in protocol stack)**
Reference: Section 4.3 (The Internet Protocol (IPv4) of text book, Figure 4.16 shows the IP Packet Header

Run Traceroute to a destination of your choice that completes with just a few *** in the output lines (or none if possible - not necessary). Use the display filter “udp||icmp” so that only UDP and/or ICMP protocol packets are displayed.

Find one of the probes sent by your VM and drill down into the IP Packet Header. Expand the Internet Protocol part of the packet in the packet details window.

I am using the Ip-wireshark-trace-1.pcapng file for this problem as traceroute does not work for me.

1. What is the IP address of your VM?
   10.0.2.15
2. What is the IP address of the destination?
   128.119.245.12
3. What is the value in the time-to-live (TTL) field in this IPv4 datagram’s header?
   TTL value: 1

4. How many bytes are in the IP header?
   20 bytes
5. How many bytes are in the payload of the IP datagram? Explain how you determined the number of payload bytes. Ask for help in lab!
   2,980 bytes. This is found under the payload description under “3 IPv4 Fragments”

12. [5 pts] **Protocols and Apps**
    a) What is a protocol?
    Network protocols define the format, order of messages sent and received among network entities, and actions taken on message transmission, receipt
    b) What is an application?
    An application is a layer at the top of the protocol stack that contains protocols that facilitate the data transfer process with lower layers
    c) Based on your definitions, is Traceroute an application or a protocol? Explain your answer.
    Traceroute is an application since it uses mostly ICMP echo packets, UDP packets, and TCP packets to measure RTT values.

[5 pts] <u>**Extra Credit:**</u>
Think about the probes you observed in (8). Find the response from the Server that causes Traceroute to terminate. If you do not see a final response in Wireshark, try another site that does send a response. Remember, we have no control over how hosts are configured – some respond to UDP probes and final ICMP probes, others do not. Screenshot your results in Wireshark and describe what type of packet you have found that causes Traceroute to finish.

Submission:

You will submit 2 files for this assignment (student id is the part before ‘@’ in your email: id@ucsc.edu):

1. <your student id>-lab1.pdf

The PDF with all of your solutions to the questions.

2. <your student id>-topo.py

    The Mininet topology you created.

[1] References to figures and sections are for the 8th edition of our text, *Computer Networks, A Top-down Approach, 8h ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2020.* Our website for this book is http://gaia.cs.umass.edu/kurose_ross You'll find lots of interesting open material there.