Luis Valdivia - 1462385
levaldiv@ucsc.edu
PreLab & Lab 1 - 10/16/19

Pre-Lab Questions:

1) **What command will show you which groups you are a member of?**

Using the command “groups<username>” will show you what groups a user belongs to.
https://www.2daygeek.com/how-to-check-which-groups-a-user-belongs-to-on-linux/

2) **What does the environment variable “$?” hold? (echo $)**

The environment variable “$?” is used to find the return value of the last executed command.
https://stackoverflow.com/questions/6834487/what-is-the-dollar-question-mark-variable-in-shell-scripting

3) **What key combination will suspend a currently running process and place it as a background process?**

To suspend a currently running process and place it as a background process we use the following key combination: CTRL+Z (which will suspend the process) and then use “bg” command (will send it to the background).
https://serverfault.com/questions/409698/how-to-send-running-process-to-background

4) **With what command (and arguments) can you find out your kernel version and the “nodename”? [Output should not include any other information]**

Using the following commands: “uname-s” will print out the kernel name, “uname-v” will print out the kernel version, and “uname-n” will print out the network node hostname.
https://www.cyberciti.biz/faq/find-print-linux-unix-kernel-version/

5) **What is the difference between the paths “.”, “..” and “~”? What does the path “/” refer to when not preceded by anything?**

The path “.” means the current directory; the path “..” means the parent of the current directory; and “~” refers to starting from the home directory. The path “/” refers to the root directory.
https://www.cs.jhu.edu/~joanne/unix.html

6) **What is a pid? Which command would you use to find the “pid” for a running process?**

Pid (or Process ID) is an identification number that is assigned to each process when it is created. Using the command “ps” will show a list of current running processes and their IDs. To see all running process use the command “ps-e.”
https://www.2daygeek.com/9-methods-to-check-find-the-process-id-pid-ppid-of-a-running-program-in-linux/

Luis Valdivia - 1462385
levaldiv@ucsc.edu
PreLab & Lab 1 - 10/16/19

7) Write a single command that will return every user's default shell.

We know that the shell for every user is stored in /etc/password. We can access this file by doing the following command, "cat/etc/password." We also know that for every user, the file contains 7 columns: *username:password:UID:GID:UserInfo:Home:ShellPath*. We need to be able to get the username and shell path. We can do that with the following command, "cat/etc/password | cut -d: -f1,7" which will give us "username:ShellPath."
However, if we just want to get the name of the shell and the path we could run the following command, "cat/etc/password | cut -d: -f1,7 | cut -d "/" -f1,3--output-delimiter=""""" which will give us "username:shell"
https://unix.stackexchange.com/questions/313928/return-every-user-s-default-shell

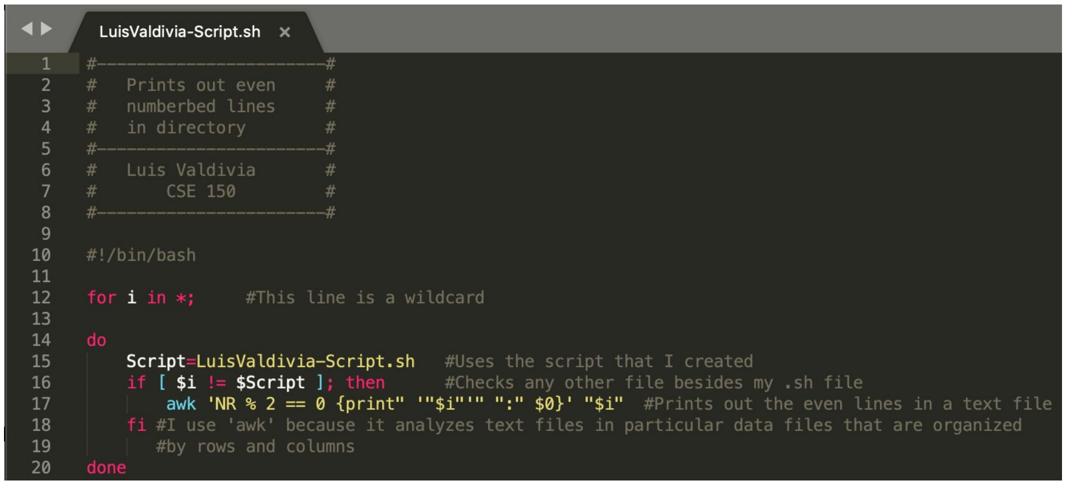
8) What is the difference between "sudo" and "su root"?

The "sudo" command allows a permitted user to execute a command as another user. While the command "su root" will open a new terminal with *root* privileges for every command. "Su" stands for "substitute" and can be used to change another user. So when we use the command "su root" you are changing the user.
https://superuser.com/questions/1226082/difference-between-sudo-su-root-vs-sudo-su

9) How would you tell your computer to run a program or script on a schedule or set interval on Linux? [E.g. Run this program once every 30 min]

To run a program/script on a schedule, we can use the "watch" command. You will be able to see the program output in time. By default watch reruns the command/program every 2 seconds but we can alter it to our needs. Ex. watch -n file, where n specifies the interval. We can also use "cron" to run periodically.
https://www.tecmint.com/run-repeat-linux-command-every-x-seconds/

10) Write a shell script that only prints out the even numbered lines of each file in the current directory.



```
#-----------------------#
#   Prints out even     #
#   numberbed lines     #
#   in directory        #
#-----------------------#
#   Luis Valdivia       #
#       CSE 150         #
#-----------------------#

#!/bin/bash

for i in *;     #This line is a wildcard

do
    Script=LuisValdivia-Script.sh   #Uses the script that I created
    if [ $i != $Script ]; then      #Checks any other file besides my .sh file
        awk 'NR % 2 == 0 {print" '"$i"'" ":" $0}' "$i"  #Prints out the even lines in a text file
    fi #I use 'awk' because it analyzes text files in particular data files that are organized
       #by rows and columns
done
```

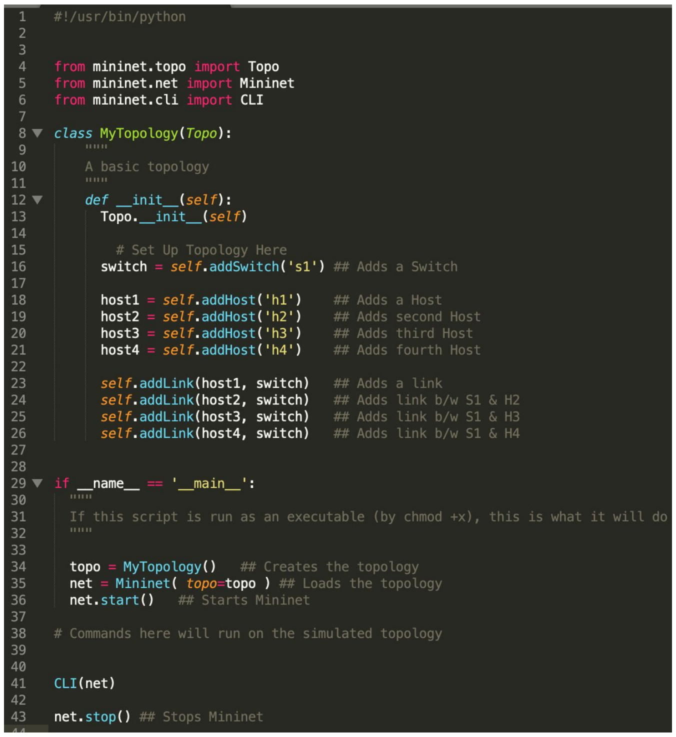
Luis Valdivia - 1462385
levaldiv@ucsc.edu
PreLab & Lab 1 - 10/16/19

The code above reads every document in the directory and prints out the even numbered lines. Line 15 is using the script that I created. Line 16 is an if statement saying that it will check any other file besides my .sh file. Awk analyzes the text files in a particular data that are organized by columns and rows. The "NR % 2 == 0" analyzes the file to see if it is divisible by 2 and if it is it prints out the line.

Luis Valdivia - 1462385
levaldiv@ucsc.edu
PreLab & Lab 1 - 10/16/19

Lab Questions:

1) **In Mininet, change the default configuration to have 4 hosts connected to a switch**

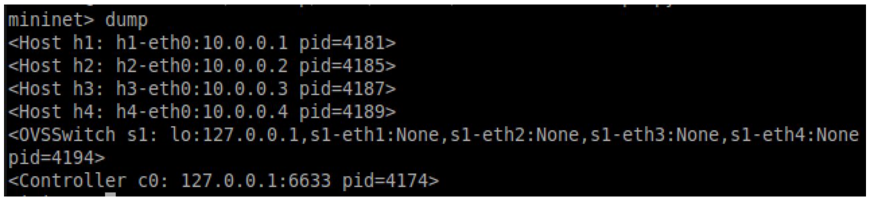


```
1  #!/usr/bin/python
2
3
4  from mininet.topo import Topo
5  from mininet.net import Mininet
6  from mininet.cli import CLI
7
8  class MyTopology(Topo):
9      """
10     A basic topology
11     """
12     def __init__(self):
13       Topo.__init__(self)
14
15         # Set Up Topology Here
16       switch = self.addSwitch('s1') ## Adds a Switch
17
18       host1 = self.addHost('h1')    ## Adds a Host
19       host2 = self.addHost('h2')    ## Adds second Host
20       host3 = self.addHost('h3')    ## Adds third Host
21       host4 = self.addHost('h4')    ## Adds fourth Host
22
23       self.addLink(host1, switch)   ## Adds a link
24       self.addLink(host2, switch)   ## Adds link b/w S1 & H2
25       self.addLink(host3, switch)   ## Adds link b/w S1 & H3
26       self.addLink(host4, switch)   ## Adds link b/w S1 & H4
27
28
29 if __name__ == '__main__':
30   """
31   If this script is run as an executable (by chmod +x), this is what it will do
32   """
33
34   topo = MyTopology()   ## Creates the topology
35   net = Mininet( topo=topo ) ## Loads the topology
36   net.start()   ## Starts Mininet
37
38 # Commands here will run on the simulated topology
39
40
41 CLI(net)
42
43 net.stop() ## Stops Mininet
```

Lines 19-21 add 3 more hosts to make a total of 4 hosts. Lines 23-26 adds a link between all the hosts and switch1 and connects them.

2) **Save a screenshot of *dump* and *pingall* output**

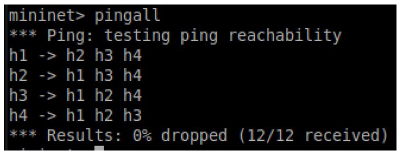
Dump:



```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=4181>
<Host h2: h2-eth0:10.0.0.2 pid=4185>
<Host h3: h3-eth0:10.0.0.3 pid=4187>
<Host h4: h4-eth0:10.0.0.4 pid=4189>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None,s1-eth4:None
pid=4194>
<Controller c0: 127.0.0.1:6633 pid=4174>
```

The "dump" command shows the information of each node of the network. We can see each host with its connection to "eth0" and their IP [as well as their pid], the switch with its connection to the controller and the controller with the interface IP.

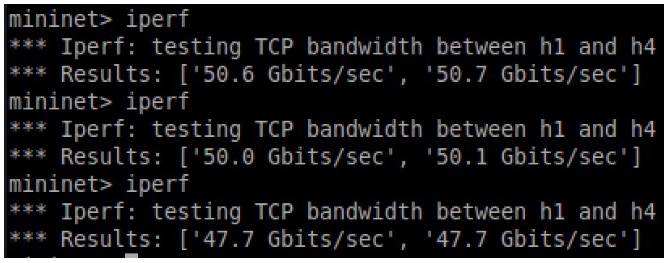
Pingall:



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Pingall sends a ping from every host to each one of the other connected hosts. In the screenshot above we can see what pingall does and that there were no packets lost.

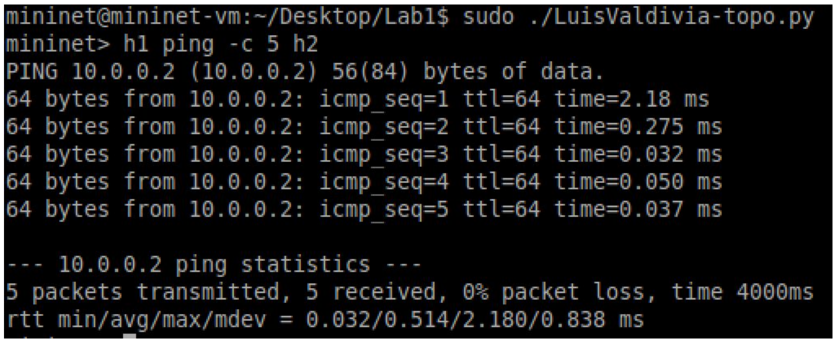
3) **Run the *iperf* command as well, and screenshot the output, how fast was the connect?**



```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['50.6 Gbits/sec', '50.7 Gbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['50.0 Gbits/sec', '50.1 Gbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['47.7 Gbits/sec', '47.7 Gbits/sec']
```

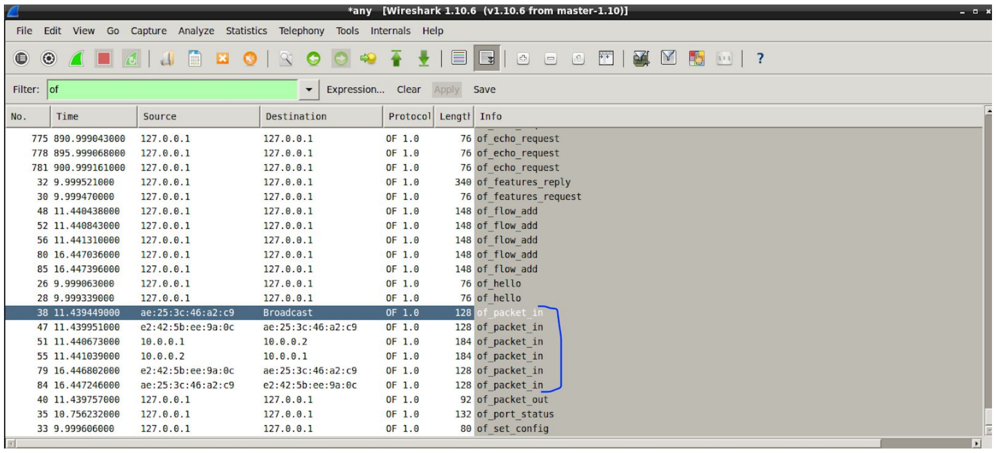
I ran the "iperf" command 3 times to see if the connection would be consistent. In doing so, I found that the connection speed is around 47-50 Gbits/sec (symmetric). This tells us the amount of data transmitted in Gbits/sec between host 1 and host 4.

4) Run wireshark
   a) **Run ping from a host to any other host using hX ping -c 5 hY. How many of_packet_in messages show up? Screenshot the results**



```
mininet@mininet-vm:~/Desktop/Lab1$ sudo ./LuisValdivia-topo.py
mininet> h1 ping -c 5 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=2.18 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.275 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.032 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.050 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.037 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.032/0.514/2.180/0.838 ms
```


*any [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: of
Expression... Clear Apply Save

```
No.   Time          Source             Destination        Protocol Length Info
  775 890.999043000 127.0.0.1          127.0.0.1          OF 1.0       76 of_echo_request
  778 895.999068000 127.0.0.1          127.0.0.1          OF 1.0       76 of_echo_request
  781 900.999161000 127.0.0.1          127.0.0.1          OF 1.0       76 of_echo_request
   32 9.999521000   127.0.0.1          127.0.0.1          OF 1.0      340 of_features_reply
   30 9.999470000   127.0.0.1          127.0.0.1          OF 1.0       76 of_features_request
   48 11.440438000  127.0.0.1          127.0.0.1          OF 1.0      148 of_flow_add
   52 11.440843000  127.0.0.1          127.0.0.1          OF 1.0      148 of_flow_add
   56 11.441310000  127.0.0.1          127.0.0.1          OF 1.0      148 of_flow_add
   80 16.447036000  127.0.0.1          127.0.0.1          OF 1.0      148 of_flow_add
   85 16.447396000  127.0.0.1          127.0.0.1          OF 1.0      148 of_flow_add
   26 9.999063000   127.0.0.1          127.0.0.1          OF 1.0       76 of_hello
   28 9.999339000   127.0.0.1          127.0.0.1          OF 1.0       76 of_hello
   38 11.439449000  ae:25:3c:46:a2:c9  Broadcast          OF 1.0      128 of_packet_in
   47 11.439951000  e2:42:5b:ee:9a:0c  ae:25:3c:46:a2:c9  OF 1.0      128 of_packet_in
   51 11.440673000  10.0.0.1           10.0.0.2           OF 1.0      184 of_packet_in
   55 11.441039000  10.0.0.2           10.0.0.1           OF 1.0      184 of_packet_in
   79 16.446802000  e2:42:5b:ee:9a:0c  ae:25:3c:46:a2:c9  OF 1.0      128 of_packet_in
   84 16.447246000  ae:25:3c:46:a2:c9  e2:42:5b:ee:9a:0c  OF 1.0      128 of_packet_in
   40 11.439757000  127.0.0.1          127.0.0.1          OF 1.0       92 of_packet_out
   35 10.756232000  127.0.0.1          127.0.0.1          OF 1.0      132 of_port_status
   33 9.999606000   127.0.0.1          127.0.0.1          OF 1.0       80 of_set_config
```

Using WireShark and the filler "of," I was able to capture **6** *of_packet_in* messages.

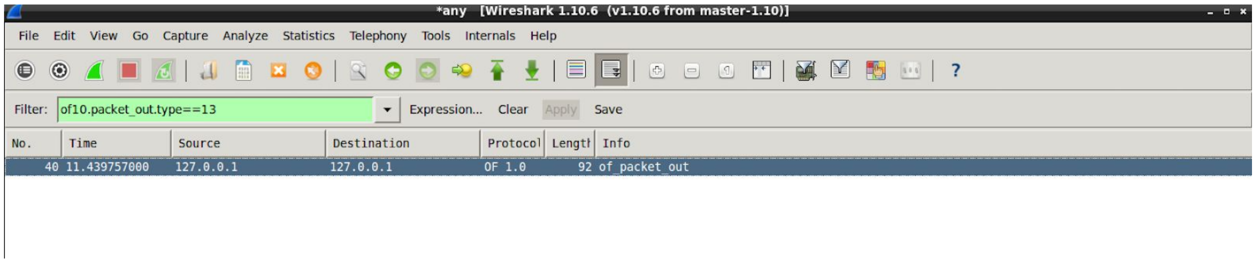
b) What is the source and destination IP addresses for these entries? Find another packet that matches the "of" filler with the OpenFlow typefield set to OFPT_PACKET_OUT. What is the source and destination IP address for this entry? Screen shot results

Number	Source IP	Destination IP
38	ae:25:3c:46:a2:c9	Broadcast
47	e2:42:5b:ee:9a:0c	ae:25:3c:46:a2:c9
51	10.0.0.1	10.0.0.2
55	10.0.0.2	10.0.0.1
79	e2:42:5b:ee:9a:0c	ae:25:3c:46:a2:c9
84	ae:25:3c:46:a2:c9	e2:42:5b:ee:9a:0c

The first one is from host1 to the switch [the controller] to make it broadcast to all the other hosts. The rest of the IPs are between host1 and host2.

Luis Valdivia - 1462385
levaldiv@ucsc.edu
PreLab & Lab 1 - 10/16/19

Using the OpenFlow typefield set to OFPT_PACKET_OUT, I was able to find another packet with the filter "of." Below is the Source IP and Destination IP.



*any [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

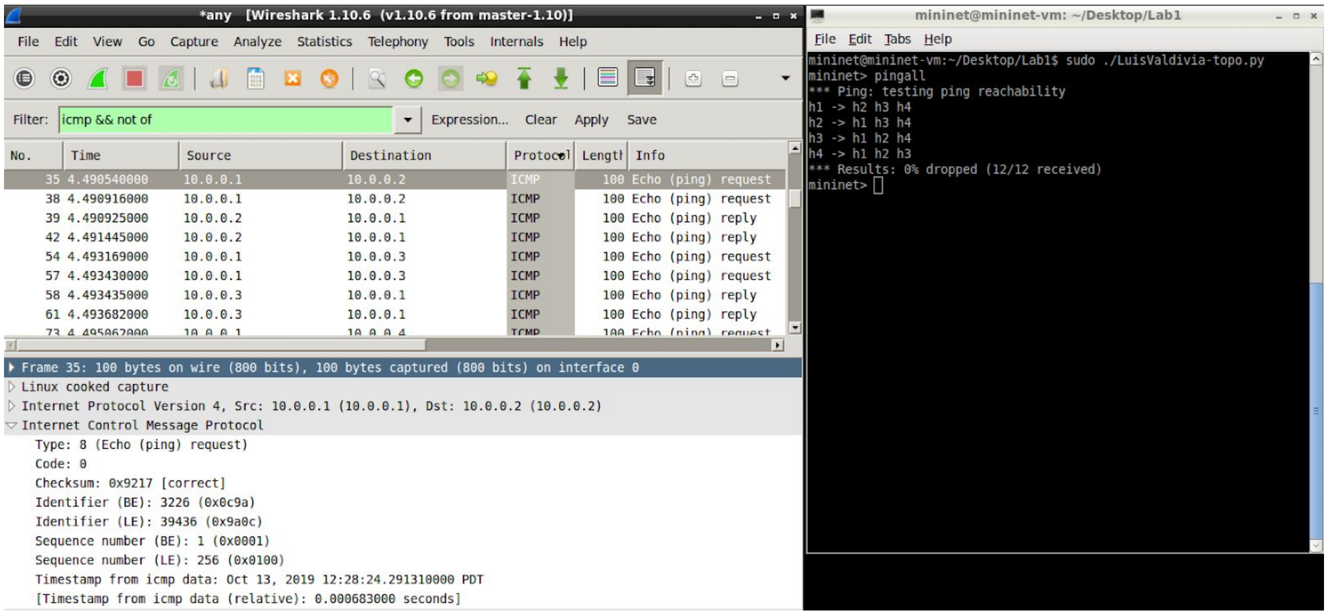
Filter: of10.packet_out.type==13

```
No.   Time          Source      Destination   Protocol  Length  Info
   40 11.439757000  127.0.0.1   127.0.0.1     OF 1.0        92  of_packet_out
```

Number	Source IP	Destination IP
40	127.0.0.1	127.0.0.1

c) **Replace display filter for "of" to "icmp && not of." Run *pingall* again, how many entries are generated in WireShark? What types of icmp entries show up? Screenshot the results.**

Wireshark showed 48 packets using the command **"icmp && not of."**



*any [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save
mininet@mininet-vm: ~/Desktop/Lab1
File Edit Tabs Help

Filter: icmp && not of

```
No.   Time          Source      Destination   Protocol  Length  Info
   35 4.490540000   10.0.0.1    10.0.0.2      ICMP       100  Echo (ping) request
   38 4.490916000   10.0.0.1    10.0.0.2      ICMP       100  Echo (ping) request
   39 4.490925000   10.0.0.2    10.0.0.1      ICMP       100  Echo (ping) reply
   42 4.491445000   10.0.0.2    10.0.0.1      ICMP       100  Echo (ping) reply
   54 4.493169000   10.0.0.1    10.0.0.3      ICMP       100  Echo (ping) request
   57 4.493430000   10.0.0.1    10.0.0.3      ICMP       100  Echo (ping) request
   58 4.493435000   10.0.0.3    10.0.0.1      ICMP       100  Echo (ping) reply
   61 4.493682000   10.0.0.3    10.0.0.1      ICMP       100  Echo (ping) reply
   73 4.495062000   10.0.0.1    10.0.0.4      ICMP       100  Echo (ping) request
```

```
Frame 35: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface 0
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)
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x9217 [correct]
    Identifier (BE): 3226 (0x0c9a)
    Identifier (LE): 39436 (0x9a0c)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    Timestamp from icmp data: Oct 13, 2019 12:28:24.291310000 PDT
    [Timestamp from icmp data (relative): 0.000683000 seconds]
```

```
mininet@mininet-vm:~/Desktop/Lab1$ sudo ./LuisValdivia-topo.py
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet> 
```

There are two different ICMP entries: echo reply and echo request.