Assignment 5:

Made by:

Elad Sezanayev - 211909940
Shaked Levi - 318985165

Table of Contents:

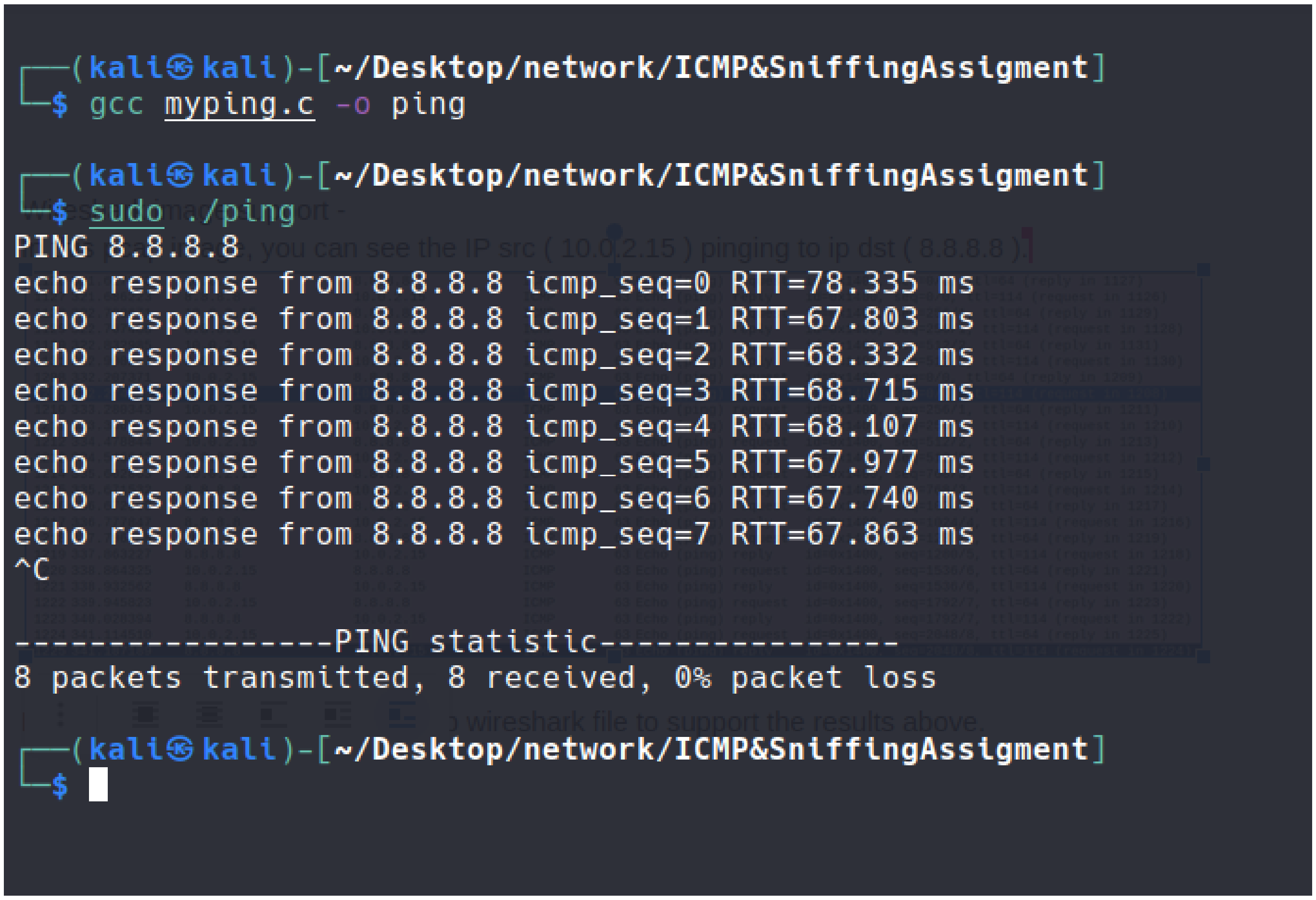
Extras -
There are pcap files & myping.c & sniffer.c code added in the zip file to review the results of each simulation ran with ping & sniffing the ping.

Prologue-

This assignment was divided into two parts.
Part A - creating a 'ping' simulator to send a ping to another dest ip and receive echo response from it.
Part B - creating a packet sniffer which will have the ability to sniff ICMP packets and display the data of those. (IP src, IP dst, type, code, sequence…)

Part A- my ping:

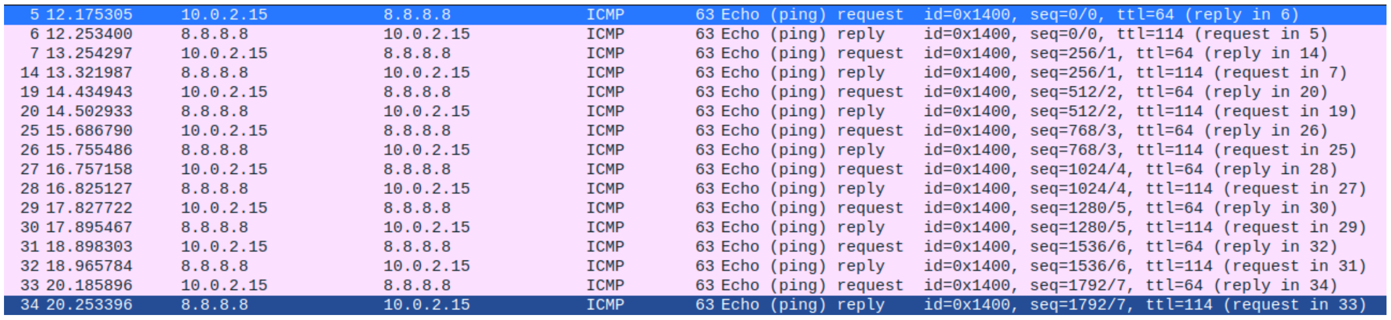
In this part we needed to simulate sending a ping to an IP Address and to receive echo responses from it.
We took it a step ahead and made it to work like a real ping (i'e in the terminal -ping).
What we did was to define a source ip (which can be spoofed, because we used raw socket) and define a destination ip (the ip we want to ping), by doing so, we let the program re ping the dest ip until the user press CTRL-C, which will then output the statistics.
Given below is a picture representing a simulation of sending ping to 8.8.8.8 (google) -



```
┌──(kali㉿kali)-[~/Desktop/network/ICMP&SniffingAssigment]
└─$ gcc myping.c -o ping

┌──(kali㉿kali)-[~/Desktop/network/ICMP&SniffingAssigment]
└─$ sudo ./ping
PING 8.8.8.8
echo response from 8.8.8.8 icmp_seq=0 RTT=78.335 ms
echo response from 8.8.8.8 icmp_seq=1 RTT=67.803 ms
echo response from 8.8.8.8 icmp_seq=2 RTT=68.332 ms
echo response from 8.8.8.8 icmp_seq=3 RTT=68.715 ms
echo response from 8.8.8.8 icmp_seq=4 RTT=68.107 ms
echo response from 8.8.8.8 icmp_seq=5 RTT=67.977 ms
echo response from 8.8.8.8 icmp_seq=6 RTT=67.740 ms
echo response from 8.8.8.8 icmp_seq=7 RTT=67.863 ms
^C

-----------------PING statistic----------------
8 packets transmitted, 8 received, 0% packet loss

┌──(kali㉿kali)-[~/Desktop/network/ICMP&SniffingAssigment]
└─$
```

Wireshark image support -

In this pcap image, you can see the IP src (10.0.2.15) pinging to ip dst (8.8.8.8) and an echo reply from the dst.



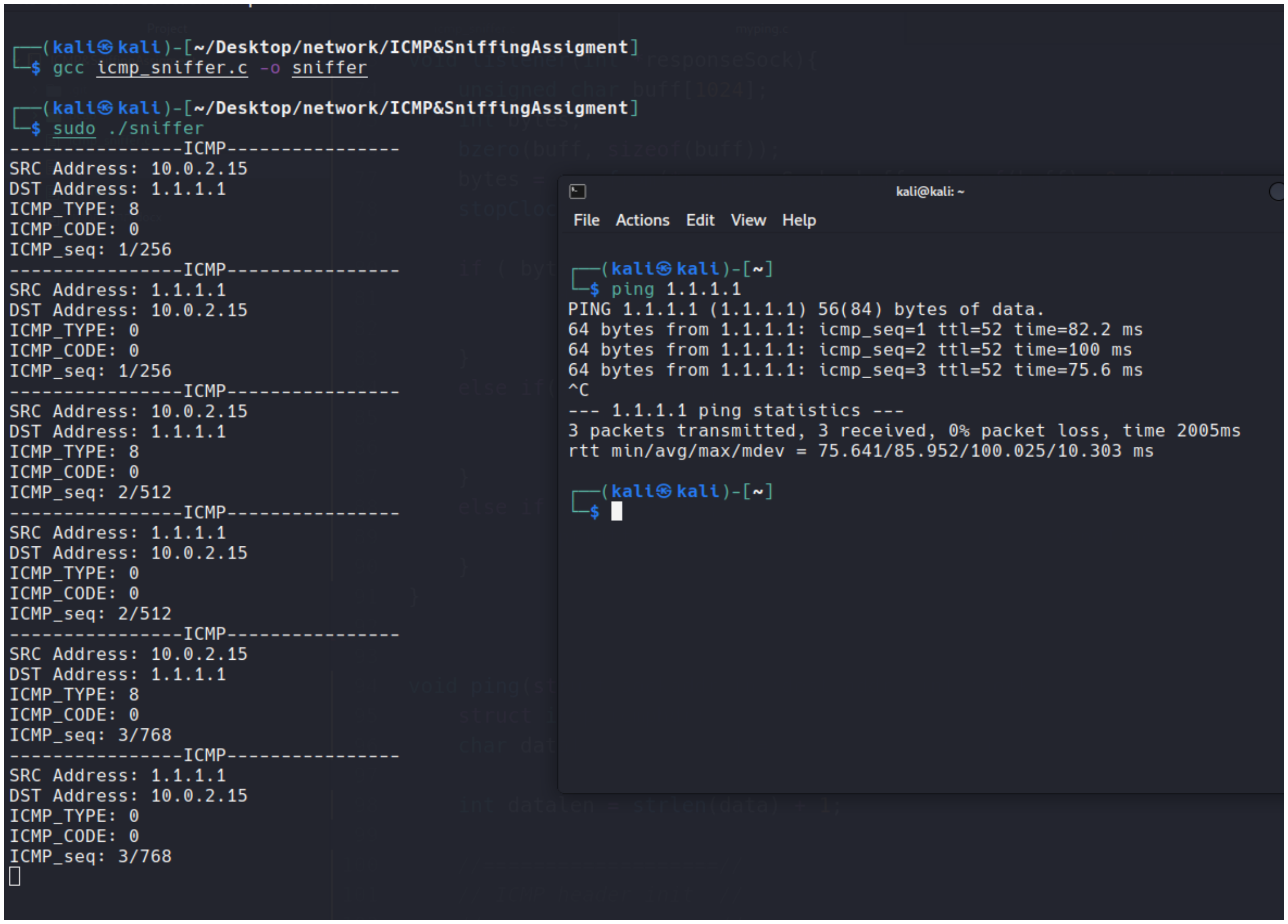
```
 5 12.175305   10.0.2.15   8.8.8.8     ICMP   63 Echo (ping) request  id=0x1400, seq=0/0, ttl=64 (reply in 6)
 6 12.253400   8.8.8.8     10.0.2.15   ICMP   63 Echo (ping) reply    id=0x1400, seq=0/0, ttl=114 (request in 5)
 7 13.254297   10.0.2.15   8.8.8.8     ICMP   63 Echo (ping) request  id=0x1400, seq=256/1, ttl=64 (reply in 14)
14 13.321987   8.8.8.8     10.0.2.15   ICMP   63 Echo (ping) reply    id=0x1400, seq=256/1, ttl=114 (request in 7)
19 14.434943   10.0.2.15   8.8.8.8     ICMP   63 Echo (ping) request  id=0x1400, seq=512/2, ttl=64 (reply in 20)
20 14.502933   8.8.8.8     10.0.2.15   ICMP   63 Echo (ping) reply    id=0x1400, seq=512/2, ttl=114 (request in 19)
25 15.686790   10.0.2.15   8.8.8.8     ICMP   63 Echo (ping) request  id=0x1400, seq=768/3, ttl=64 (reply in 26)
26 15.755486   8.8.8.8     10.0.2.15   ICMP   63 Echo (ping) reply    id=0x1400, seq=768/3, ttl=114 (request in 25)
27 16.757158   10.0.2.15   8.8.8.8     ICMP   63 Echo (ping) request  id=0x1400, seq=1024/4, ttl=64 (reply in 28)
28 16.825127   8.8.8.8     10.0.2.15   ICMP   63 Echo (ping) reply    id=0x1400, seq=1024/4, ttl=114 (request in 27)
29 17.827722   10.0.2.15   8.8.8.8     ICMP   63 Echo (ping) request  id=0x1400, seq=1280/5, ttl=64 (reply in 30)
30 17.895467   8.8.8.8     10.0.2.15   ICMP   63 Echo (ping) reply    id=0x1400, seq=1280/5, ttl=114 (request in 29)
31 18.898303   10.0.2.15   8.8.8.8     ICMP   63 Echo (ping) request  id=0x1400, seq=1536/6, ttl=64 (reply in 32)
32 18.965784   8.8.8.8     10.0.2.15   ICMP   63 Echo (ping) reply    id=0x1400, seq=1536/6, ttl=114 (request in 31)
33 20.185896   10.0.2.15   8.8.8.8     ICMP   63 Echo (ping) request  id=0x1400, seq=1792/7, ttl=64 (reply in 34)
34 20.253396   8.8.8.8     10.0.2.15   ICMP   63 Echo (ping) reply    id=0x1400, seq=1792/7, ttl=114 (request in 33)
```

In the zip folder, we added the entire pcap wireshark file to support the results above.

Part B - Sniffing:

In this part we needed to simulate an ICMP sniffer which will sniff all packets in the network and only display those of ICMP protocol.
What we did at first was to open a raw socket under the availability to read all the received packet data.
After that we analyzed the data piece by piece. At first we accessed the Ethernet header to see if we got an IP header. Moving on we checked if the IP header contained an ICMP protocol header, If so, we unpacked the data in the ICMP header and displayed it on screen to the user.

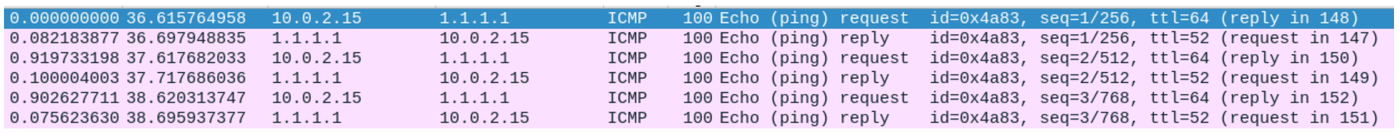
Given below is a picture representing a simulation sniffing packets that go through our local ip address and 1.1.1.1 (cloudflare) -
To explain it further, the wireshark ran simultaneously to our sniffer in order to backup that the results are the same.



```
┌──(kali㉿kali)-[~/Desktop/network/ICMP&SniffingAssigment]
└─$ gcc icmp_sniffer.c -o sniffer

┌──(kali㉿kali)-[~/Desktop/network/ICMP&SniffingAssigment]
└─$ sudo ./sniffer
----------------ICMP----------------
SRC Address: 10.0.2.15
DST Address: 1.1.1.1
ICMP_TYPE: 8
ICMP_CODE: 0
ICMP_seq: 1/256
----------------ICMP----------------
SRC Address: 1.1.1.1
DST Address: 10.0.2.15
ICMP_TYPE: 0
ICMP_CODE: 0
ICMP_seq: 1/256
----------------ICMP----------------
SRC Address: 10.0.2.15
DST Address: 1.1.1.1
ICMP_TYPE: 8
ICMP_CODE: 0
ICMP_seq: 2/512
----------------ICMP----------------
SRC Address: 1.1.1.1
DST Address: 10.0.2.15
ICMP_TYPE: 0
ICMP_CODE: 0
ICMP_seq: 2/512
----------------ICMP----------------
SRC Address: 10.0.2.15
DST Address: 1.1.1.1
ICMP_TYPE: 8
ICMP_CODE: 0
ICMP_seq: 3/768
----------------ICMP----------------
SRC Address: 1.1.1.1
DST Address: 10.0.2.15
ICMP_TYPE: 0
ICMP_CODE: 0
ICMP_seq: 3/768
```

```
┌──(kali㉿kali)-[~]
└─$ ping 1.1.1.1
PING 1.1.1.1 (1.1.1.1) 56(84) bytes of data.
64 bytes from 1.1.1.1: icmp_seq=1 ttl=52 time=82.2 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=52 time=100 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=52 time=75.6 ms
^C
--- 1.1.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 75.641/85.952/100.025/10.303 ms

┌──(kali㉿kali)-[~]
└─$
```

Extras:
ICMP Type : 8 relates to Echo request, 0 relates to Echo reply.
Whole packets wireshark captured:



```
0.000000000 36.615764958  10.0.2.15       1.1.1.1        ICMP    100 Echo (ping) request  id=0x4a83, seq=1/256, ttl=64 (reply in 148)
0.082183877 36.697948835  1.1.1.1         10.0.2.15      ICMP    100 Echo (ping) reply    id=0x4a83, seq=1/256, ttl=52 (request in 147)
0.919733198 37.617682033  10.0.2.15       1.1.1.1        ICMP    100 Echo (ping) request  id=0x4a83, seq=2/512, ttl=64 (reply in 150)
0.100004003 37.717686036  1.1.1.1         10.0.2.15      ICMP    100 Echo (ping) reply    id=0x4a83, seq=2/512, ttl=52 (request in 149)
0.902627711 38.620313747  10.0.2.15       1.1.1.1        ICMP    100 Echo (ping) request  id=0x4a83, seq=3/768, ttl=64 (reply in 152)
0.075623630 38.695937377  1.1.1.1         10.0.2.15      ICMP    100 Echo (ping) reply    id=0x4a83, seq=3/768, ttl=52 (request in 151)
```

In the zip folder, we added the entire pcap wireshark file to support the results above.