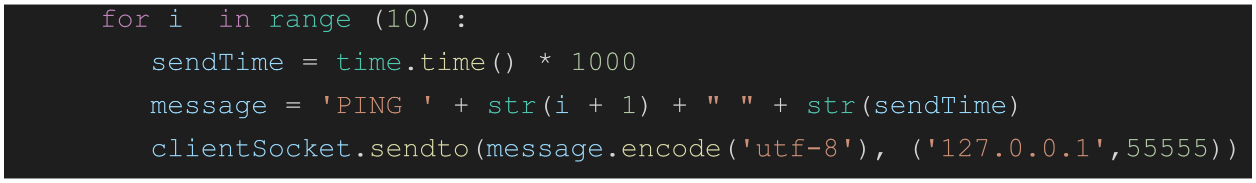
Assignment 1

Tuguldur Tserenbaljir (鐵特德) 1090062711

Part 1 :

After examining the “UDPPingerServer.py” file, I created a “UDPPingerClient.py” file to send and receive packets using UDP sockets with a timeout. Firstly, I create a client socket that is binded to my local host with the port of 12000 and set its timeout to 1 second. To calculate average round-trip time and packet loss I created variables (avg_rtt, total_packets) with a value of 0. Since we are sending 10 messages to the server, a for loop with the range of 10 is created and the messages send the strings PING, sequence number, and sending time in milliseconds. We increment the sequence number by one because the sequence number starts at 1, not 0. When the message variable is ready to be sent by the client, I use the “sendto” module of the socket library with the encoding of “utf-8” (because the message should be byte-like). Thus, we record the time of sending the message to calculate the round-trip time for future purposes.



```
    for i  in range (10) :
        sendTime = time.time() * 1000
        message = 'PING ' + str(i + 1) + " " + str(sendTime)
        clientSocket.sendto(message.encode('utf-8'), ('127.0.0.1',55555))
```

In contrast, I use the try and except the method of python. In the “try”, my code tries to receive the data from the server within the time frame but if it doesn’t it goes to the exception “timeout”. If the following ping messages are successfully received by the client end, it will record the accepted time and print out in the following format. Also, get the round-trip time by minusing sending time (which we recorded when sending the message) from the receiving time (which we recorded when receiving the message). Thus, for every successful message received, I add all of the values to the variable “avg_rtt” to calculate the average round-trip time in the future. Since the time is a little bit long and tedious to read, I only printed out the first three digits after the decimal point.

```
try:
    data, server = clientSocket.recvfrom(1024)
    rec_Time = time.time() * 1000
    rtt = rec_Time - sendTime
    avg_rrt = avg_rrt + rtt
    print("PING " + str(i+1) + ' ' + f"{rtt:.3f}")
```

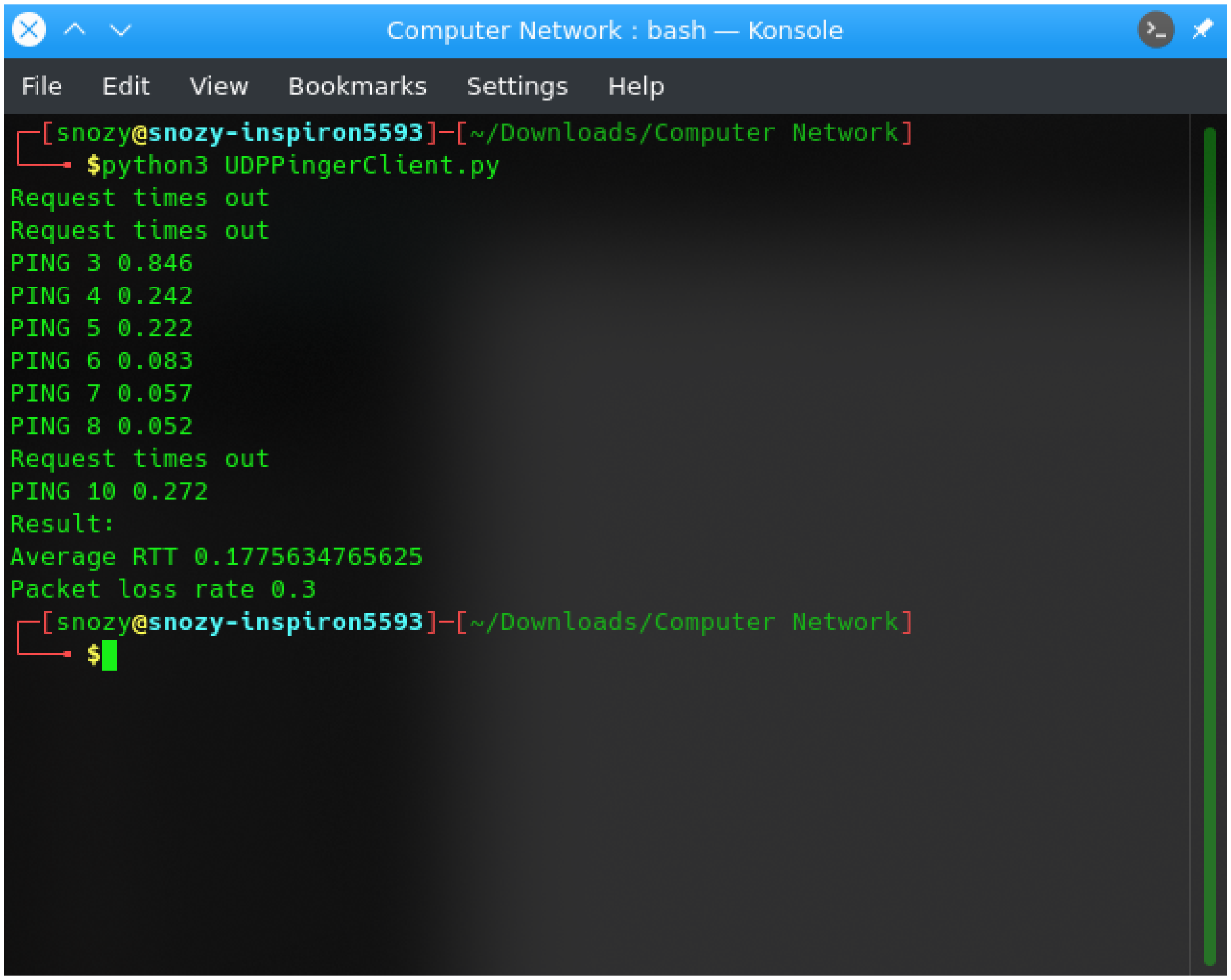
On the other hand, if there is a timeout exception, it prints out “Request times out” and increments the total number of dropped packets by 1 which is named “packets”.

After every loop, total packets are incremented by 1 because for every message timed out or not, it is still considered as one packet. So, for the purpose to find the total amount of packets it is incremented after every loop.

Finally, I print out the results of the average round-trip and packet and loss rate. The average round-trip is calculated by adding all of the round-trip time from all of the messages and dividing by 10. Whereas the Packet loss rate is the total number of packets dropped (variable names packets) divided by the total number of packets which is 10.

The hardest problem I faced in part 1 was pinging the server from the client's perspective since it is my first time doing so. I kept getting “OSError: Errno 99 ”, but after watching various youtube videos and articles from the internet I was able to find my bug and successfully ping and finish part 1.

Result:


Computer Network : bash — Konsole
File Edit View Bookmarks Settings Help

```
┌─[snozy@snozy-inspiron5593]─[~/Downloads/Computer Network]
└──╼ $python3 UDPPingerClient.py
Request times out
Request times out
PING 3 0.846
PING 4 0.242
PING 5 0.222
PING 6 0.083
PING 7 0.057
PING 8 0.052
Request times out
PING 10 0.272
Result:
Average RTT 0.1775634765625
Packet loss rate 0.3
┌─[snozy@snozy-inspiron5593]─[~/Downloads/Computer Network]
└──╼ $
```

Part 2 :

To reuse Part 1, I started out to delete the extra variables and exceptions that I do not need. After that, I created imported struct and threading from the python library. Then, I created another socket called “my_socket” to receive the “ICMP” for the assignment.

```
from socket import *
import struct
import time
import threading
```

```
clientSocket = socket(AF_INET, SOCK_DGRAM)
my_socket = socket(AF_INET, SOCK_RAW, getprotobyname('icmp'))
```

The first steps are pretty similar to Part 1 as I send a message using UDP in a “for” loop 10 times. As soon as the client sends a message to the server using UDP, it uses multithreading to receive the following information. Before the threading starts, there is a “listen” function where purpose is to listen to the received message from the server to the client in a timeout of 1 second.

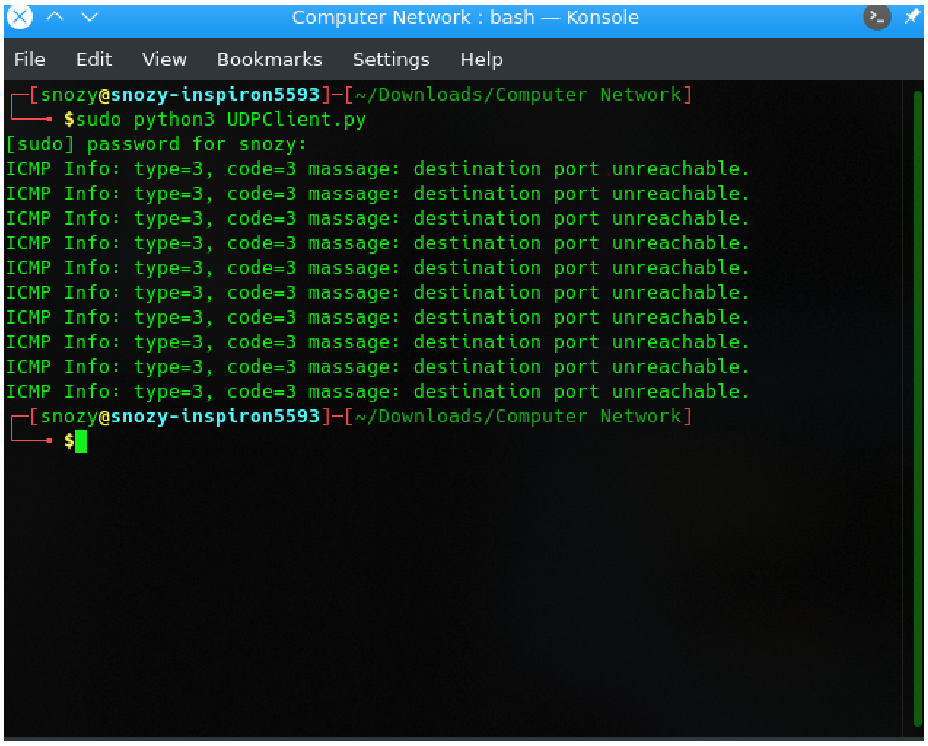
```
    def listen():
            rec_packet, addr = my_socket.recvfrom(1024)
            icmp_header = rec_packet[20:28]
            type, code, checksum, p_id, sequence = struct.unpack('bbHHh',
icmp_header)
            print("ICMP Info: type=" + str(type) + ", code=" + str(code)
+ " massage: destination port unreachable.")
```

After receiving the information, the variable “rec_packet” has all of the “ICMP” information we need. Thus, created a header to save and extract the ICMP information where collects from 20 to 28. However, we only need the first two which are the ICMP type and code. After successfully attaining the information on the ICMP, the function prints out the information we need for this Part 2.

Hardest problem:

The hardest problem I faced was learning to multithread as easy as it is, it is hard to get confused very easily. Another problem I faced was when I forgot to turn off my firewall in my Linux system and the receiving end kept going into timeout where it took me extra two hours of sitting to figure it out.

Result:


Computer Network : bash — Konsole
File Edit View Bookmarks Settings Help

```
┌─[snozy@snozy-inspiron5593]─[~/Downloads/Computer Network]
└──╼ $sudo python3 UDPClient.py
[sudo] password for snozy:
ICMP Info: type=3, code=3 massage: destination port unreachable.
ICMP Info: type=3, code=3 massage: destination port unreachable.
ICMP Info: type=3, code=3 massage: destination port unreachable.
ICMP Info: type=3, code=3 massage: destination port unreachable.
ICMP Info: type=3, code=3 massage: destination port unreachable.
ICMP Info: type=3, code=3 massage: destination port unreachable.
ICMP Info: type=3, code=3 massage: destination port unreachable.
ICMP Info: type=3, code=3 massage: destination port unreachable.
ICMP Info: type=3, code=3 massage: destination port unreachable.
ICMP Info: type=3, code=3 massage: destination port unreachable.
┌─[snozy@snozy-inspiron5593]─[~/Downloads/Computer Network]
└──╼ $
```

Part 3:

First, I created a function "traceroute" called by the main. Then, set the destination address, port, and max hops. After setting the values of the variables, I created and loop which loops from TTL to max hops. Inside the for loop, it configures the receiving(ICMP) and sending (UDP) sockets where both of its timeouts are to 1 second. In addition to binding the receive, it system writes out the TTL and declares the values of the current address, name, and tries(In our case it is 3). If it does not successfully send a message it prints out "Request timed out". Otherwise, it continues to try (3) to receive the ICMP message whereas it calculates the round-trip time and tries to get the current address name if there is any. After every try,

```
        while tries > 0:
            try:
                _, curr_addr = receive.recvfrom(512)
                send_time = time.time() * 1000
                rec_time = time.time() * 1000
                rtt = rec_time - send_time
                curr_addr = curr_addr[0]
                try:
                    curr_name = socket.gethostbyaddr(curr_addr)[0]
                except socket.error:
                    curr_name = curr_addr
            except socket.error as errno:
                tries = tries - 1
                pass
        send.close()
        receive.close()
```

It closes the sending and receiving socket and continues the for loop if it has not reached the final destination. If it has the loop breaks. The system writes the current address with its name and round-trip time.

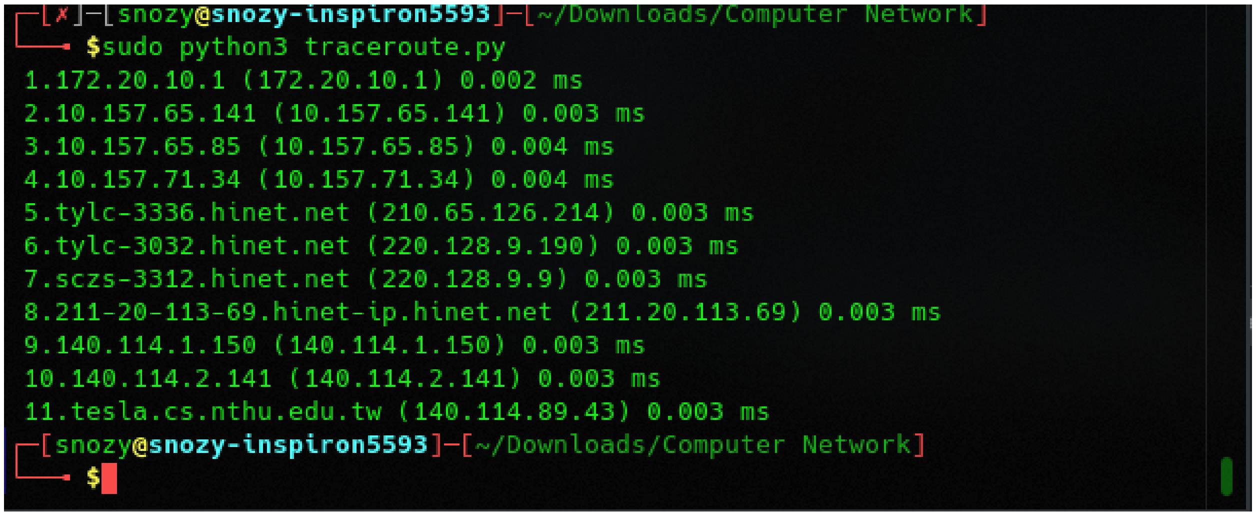
```
            sys.stdout.write("%s (%s) %0.3f ms\n" %
(curr_name,curr_addr,rtt))

            ttl += 1
            if curr_addr == dest_addr or ttl > max_hops:
                break
```

The hardest problem I faced:

The hardest problem I faced was calculating each if statement.

Result:



```
┌─[✗]─[snozy@snozy-inspiron5593]─[~/Downloads/Computer Network]
└──╼ $sudo python3 traceroute.py
1.172.20.10.1 (172.20.10.1) 0.002 ms
2.10.157.65.141 (10.157.65.141) 0.003 ms
3.10.157.65.85 (10.157.65.85) 0.004 ms
4.10.157.71.34 (10.157.71.34) 0.004 ms
5.tylc-3336.hinet.net (210.65.126.214) 0.003 ms
6.tylc-3032.hinet.net (220.128.9.190) 0.003 ms
7.sczs-3312.hinet.net (220.128.9.9) 0.003 ms
8.211-20-113-69.hinet-ip.hinet.net (211.20.113.69) 0.003 ms
9.140.114.1.150 (140.114.1.150) 0.003 ms
10.140.114.2.141 (140.114.2.141) 0.003 ms
11.tesla.cs.nthu.edu.tw (140.114.89.43) 0.003 ms
┌─[snozy@snozy-inspiron5593]─[~/Downloads/Computer Network]
└──╼ $
```