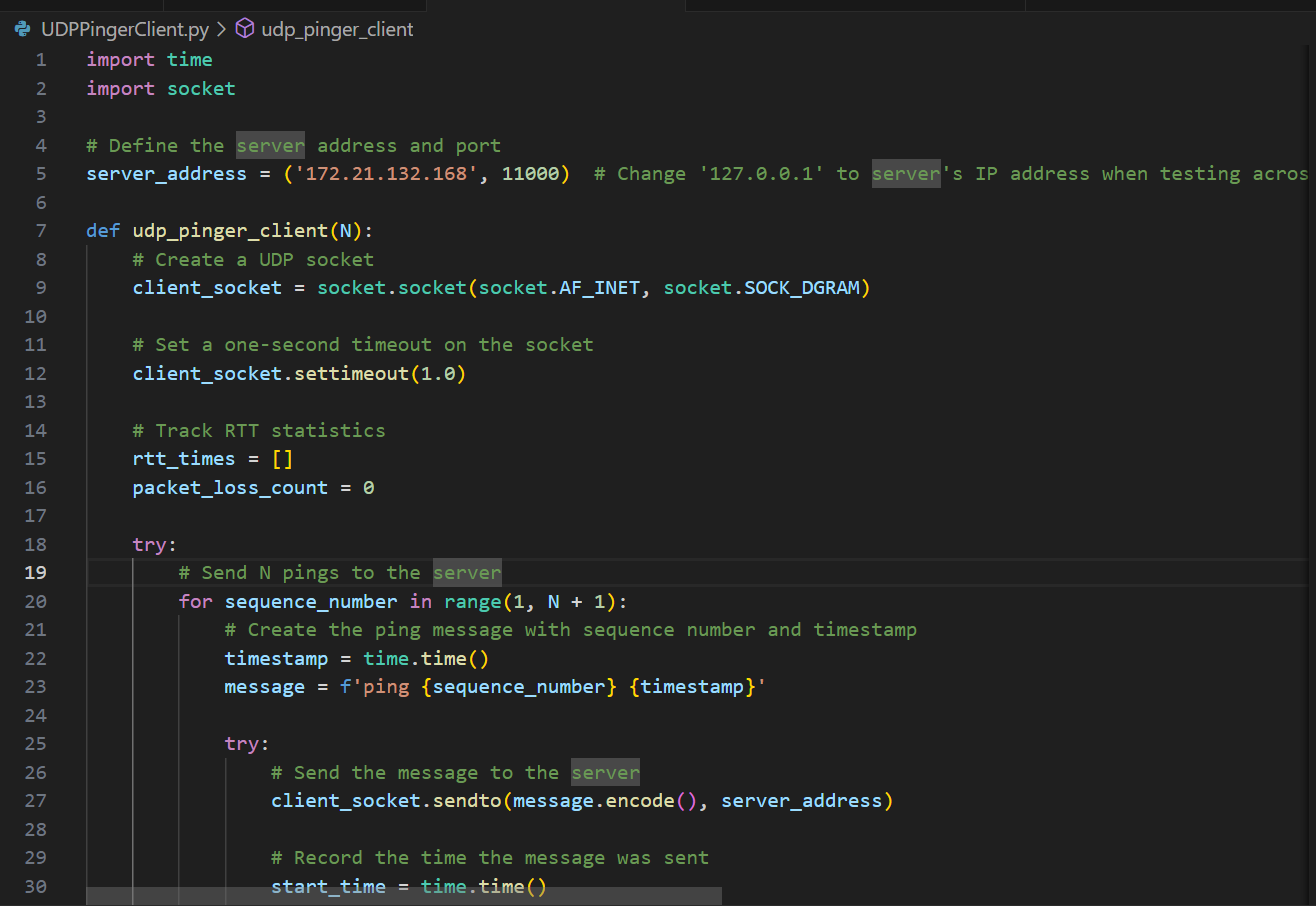
**Part-1: UDP Pinger**

**UDPPingerClient.py:**

Initially, we developed a basic UDP client that successfully communicated with the provided UDP server code in the assignment. After establishing successful communication, we enhanced the client with additional features, including allowing the user to input the number of pings, calculating Round Trip Time (RTT), and adding error handling mechanisms.



**RTT Calculations:**

For RTT calculations, we utilized the time module to record timestamps when sending and receiving messages. A list was created to store these times, and the min and max functions were used to determine the minimum and maximum RTT values.



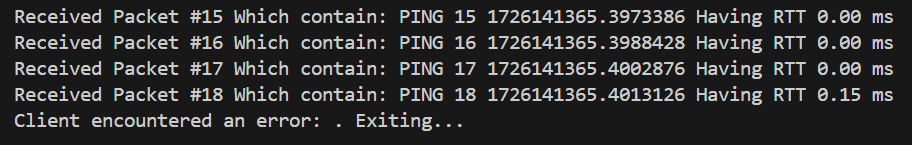


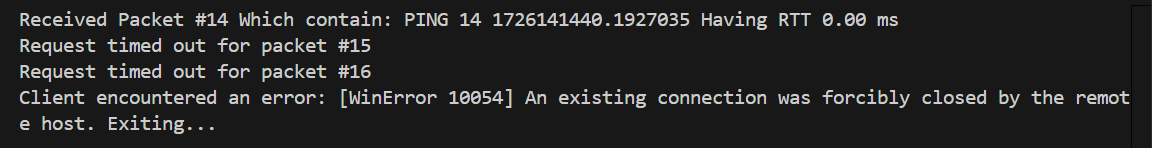
**Result:**

The result of the communication displays RTT statistics in the terminal, as shown in the example below.

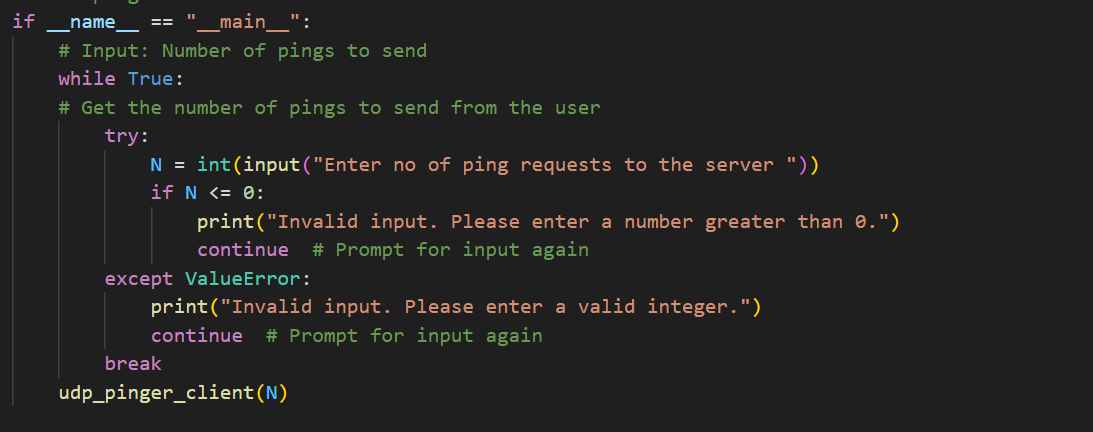
**Error Handling:**

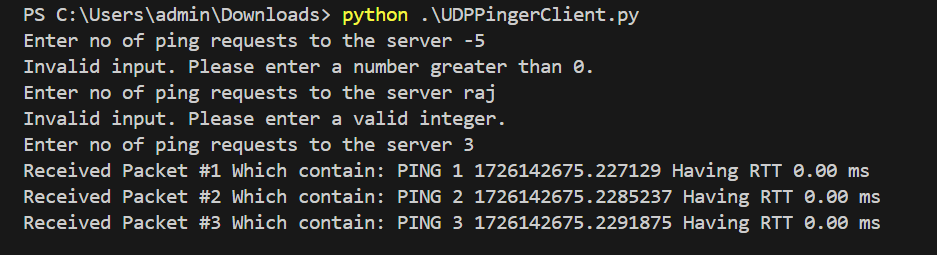
We introduced error-handling logic that prints error messages to the client’s terminal and gracefully closes the socket in case of errors, ensuring that the socket is freed for reuse. This includes handling scenarios -

1. where the client is closed by pressing Ctrl+C or when the server is shut down.
2. where the server is closed by pressing Ctrl+C or when the server is shut down.



During testing, we discovered that if a user entered an invalid input, such as a negative number or random characters for the number of pings, the client would display an error and exit. To address this, we implemented code that prompts the user to re-enter a valid input without closing the socket. We utilized assistance from ChatGPT to generate this error-handling code





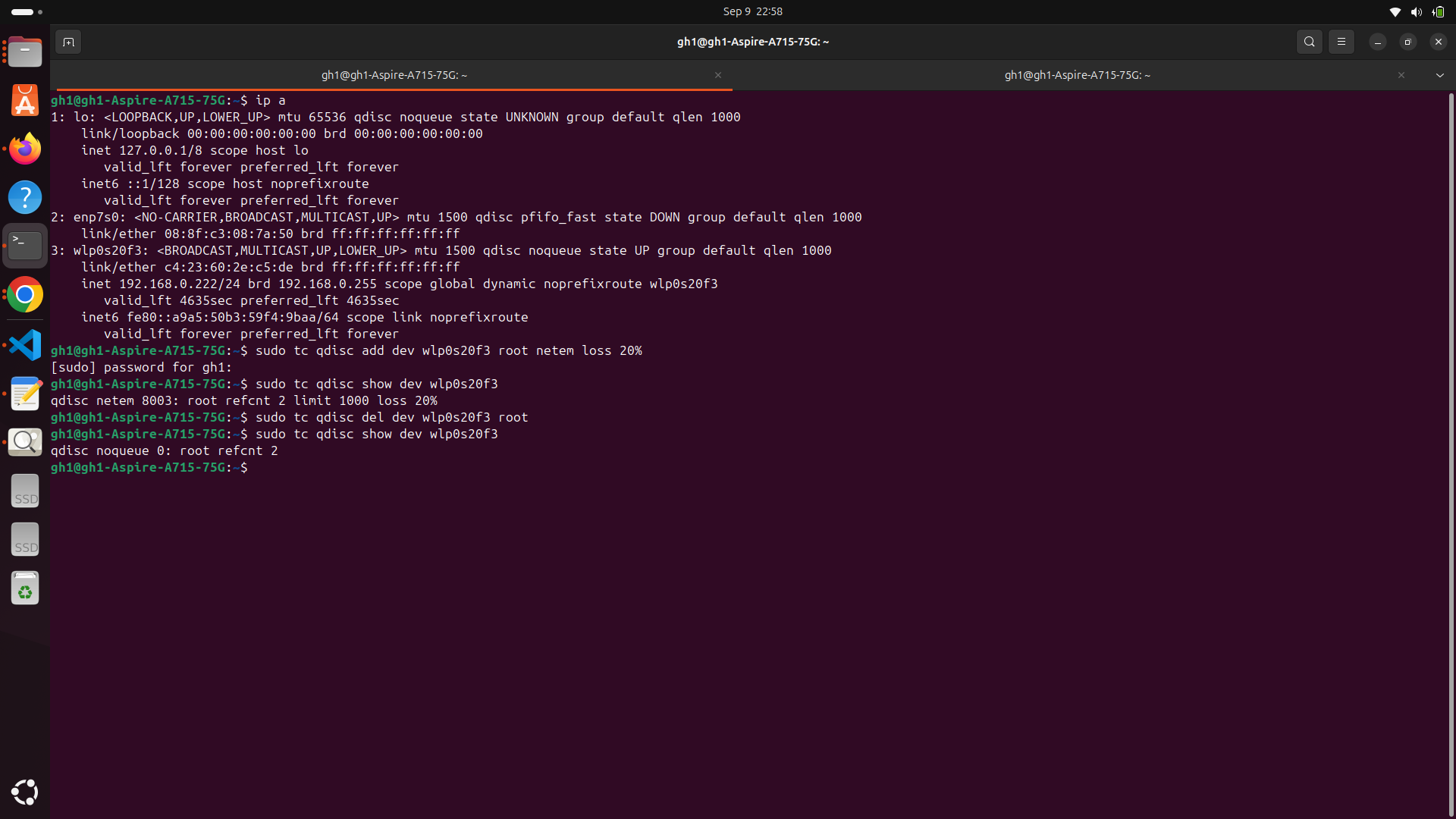
**UDPPingerModified Server:**

As per the assignment requirements, we modified the server code by removing the random packet loss simulation and using the Network Interface Card (NIC) to simulate 20% packet loss. This was achieved using the following commands:

sudo tc qdisc add dev wlp020f3 - This command adds packet loss simulation.

sudo tc qdisc show dev wlp020f3 - This verifies that packet loss simulation has started, showing qdisc netem 8003: root refcnt 2 limit 1000 loss 20%.

sudo tc qdisc del dev wlp020f3 - This command stops the packet loss simulation.



**Part 2- TCP Pinger**

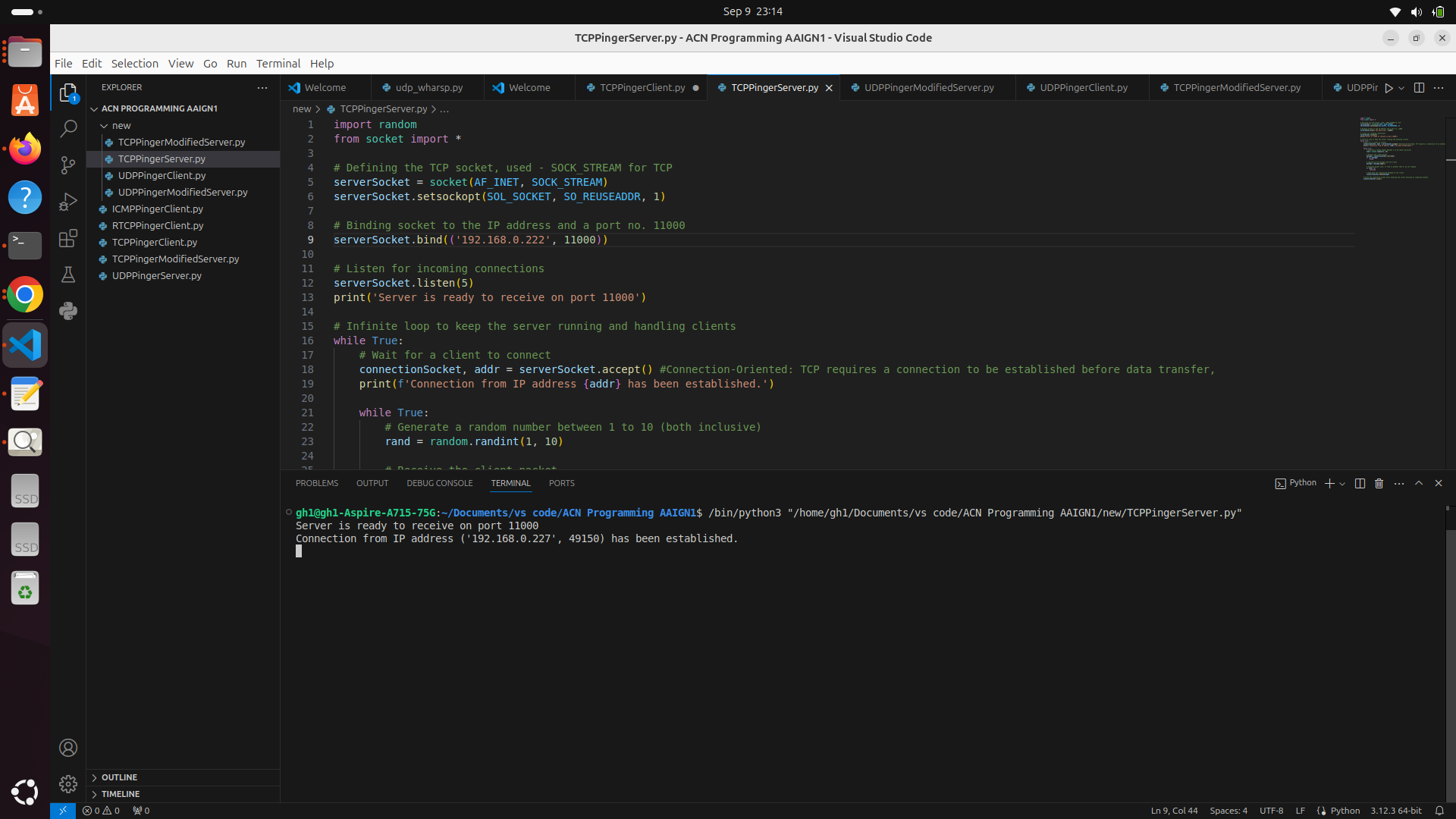
1. TCPPingerServer.py

UDP server code is modified to work for TCP server. The SOCK\_DGRAM in UDP server code which is required for creating a socket is replaced with SOCK\_STREAM. There is no any connection establishment in UDP but TCP has connection establishment so only bind is required in UDP server code but in TCP we need to bind , listening for incoming connections and then accepting the connections. In TCP connection closing is required which is not required in UDP.

This code generates a integer which results the packet loss with probability of 20%.

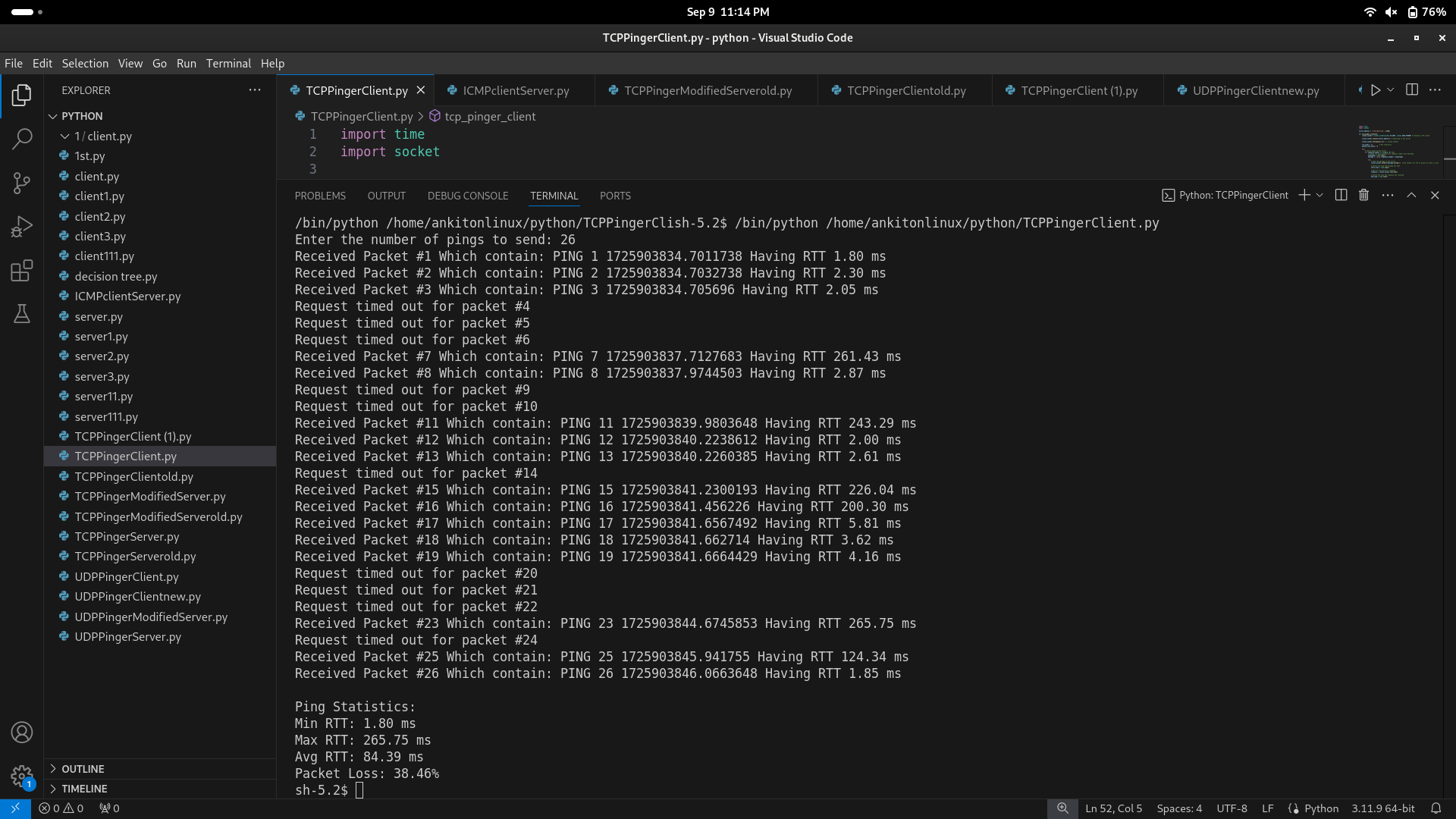
During development, we executed on same machine by send pings to 127.0.0.1 and done this procedure till point we get correct code.

After that we run server and client of TCP on different machines and added partners computer IP while binding IP to port address 11,000 .



1. TCPPingerClient.py

For TCP client code we have used SOCK\_STREAM and testing and debugging done on same machine at initial phase on IP address 127.0.0.1. We maintained the time when TCP ping packet is send to server and subtracted that time from the time of when response reached to client for Round Trip Time calculations.



Minimum Round Trip Time (RTT) is 1.80 mili-seconds.

Maximum Round Trip Time (RTT) is 265.75 mili-seconds.

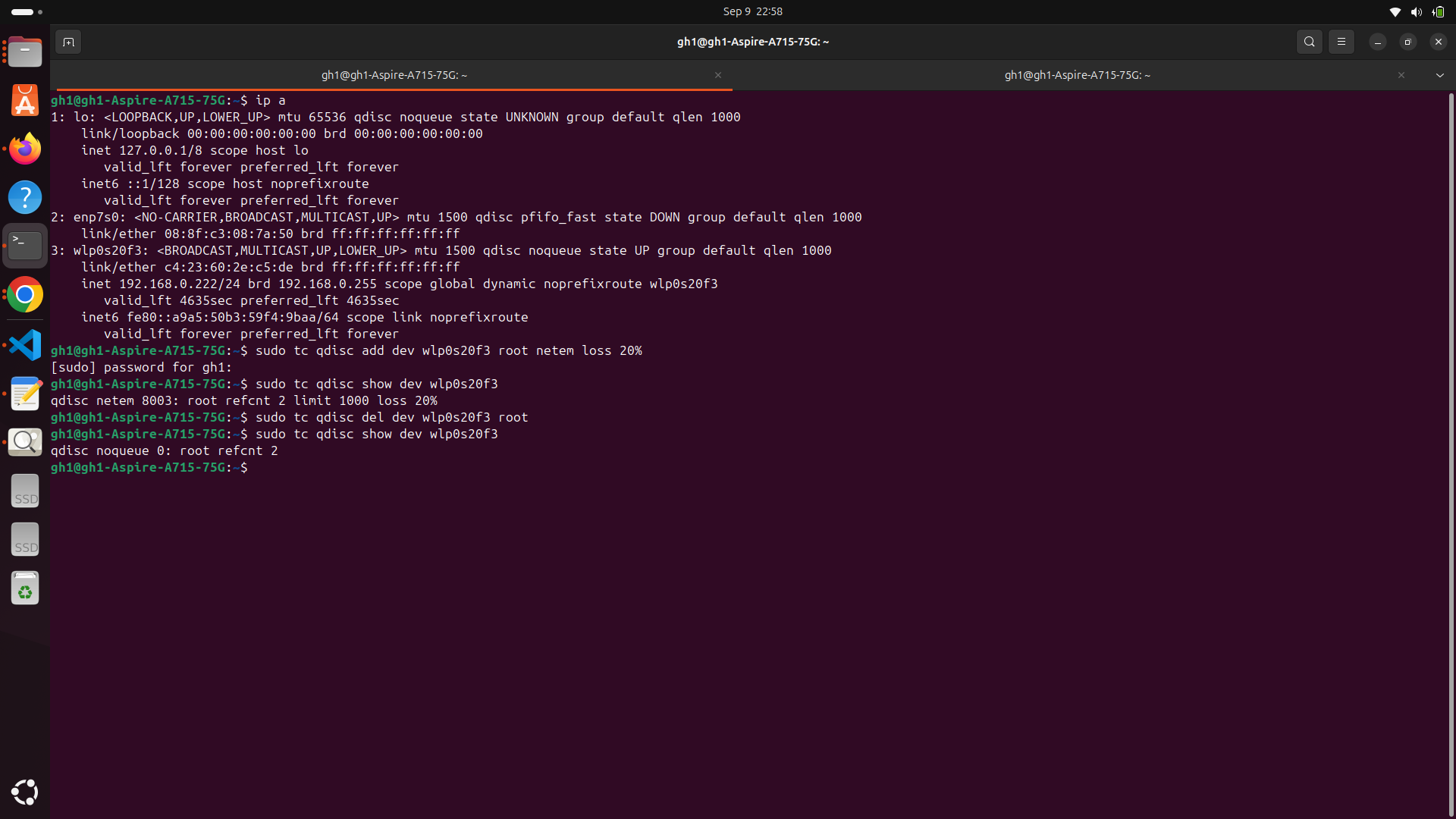
Average Round Trip Time (RTT) is 84.39 mili-seconds.

The rate of packet loss is 38.46 %

From the results we obtained by testing the TCP client and server on different machines, UDP is faster protocol than TCP as Average Round Trip Time (RTT) for TCP is 84.39 mili-seconds and that for UDP the average round trip time is 39.68 mili-seconds. This is due to the fact that TCP requires connection establishment and acknowledgment. For very simple packets like pings the big overhead of connection establishment handshaking, acknowledgements and retransmission of packets makes TCP less efficient in this scenario.

1. TCPPingerModifiedserver.py

Introducing the packet loss of 20% by traffic control (tc) netem utility in our LInux system, this causes the losses at NIC level and hence hence no need of code to simulate ping paket loss using randint function.

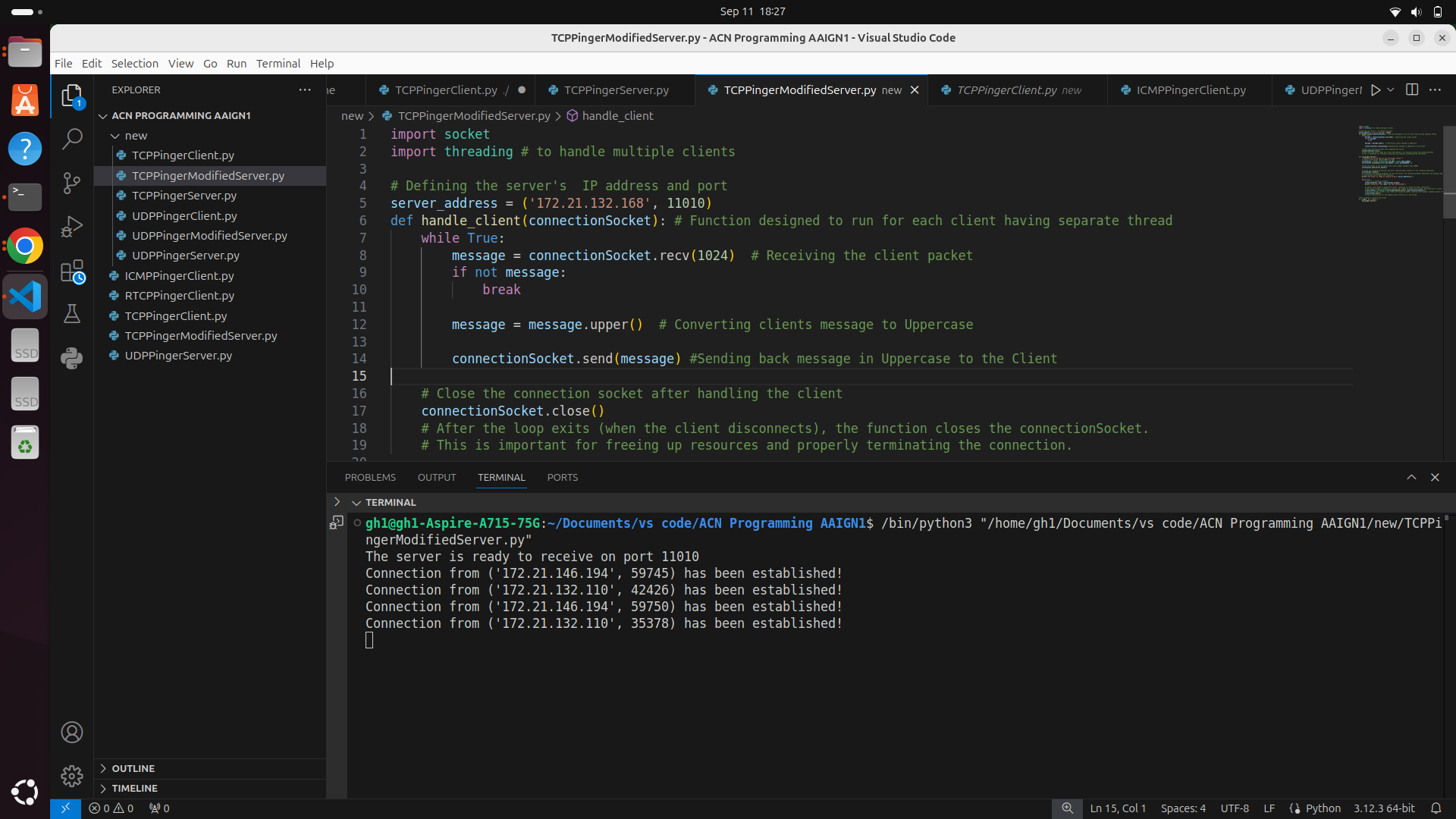


After running that commands we checked for the same by “sudo tc qdisc show dev wlp020f3” and the output is

qdisc netem 8003: root refcnt 2 limit 1000 loss 20%

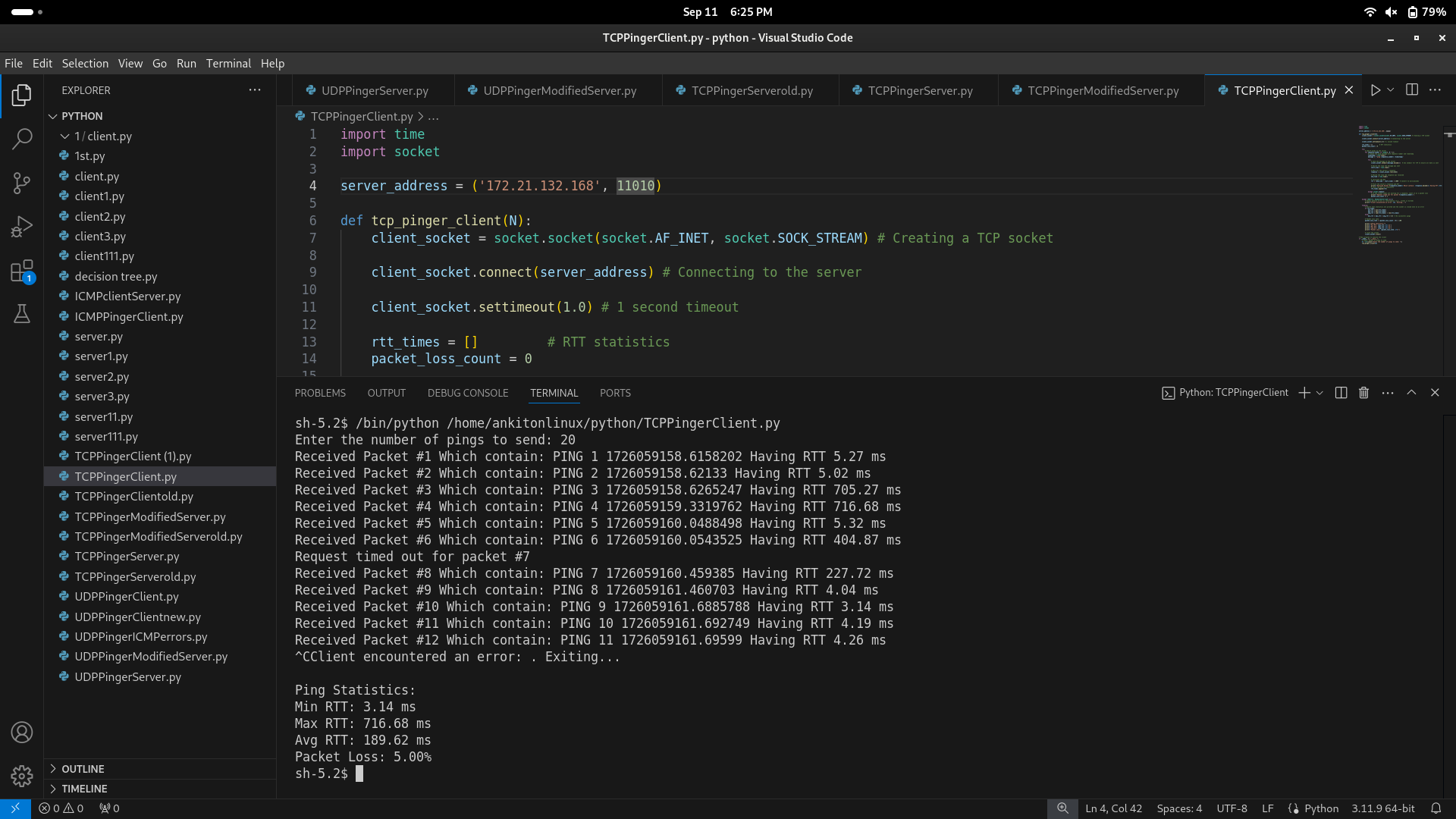
TCP modified code is able to handle multiple incoming client requests. If waits for client to connect and when a connection is made it returns new connectionSocket for communicating with client. Here we implemented such that each client gets its own thread and makes sure that connecting multiple clients and interacting with server is done properly. In that case server will remain available for accepting new client requests while handling previous threads which justifies that this allows multiple clients handled concurrently.

TCP multiple clients connected to server

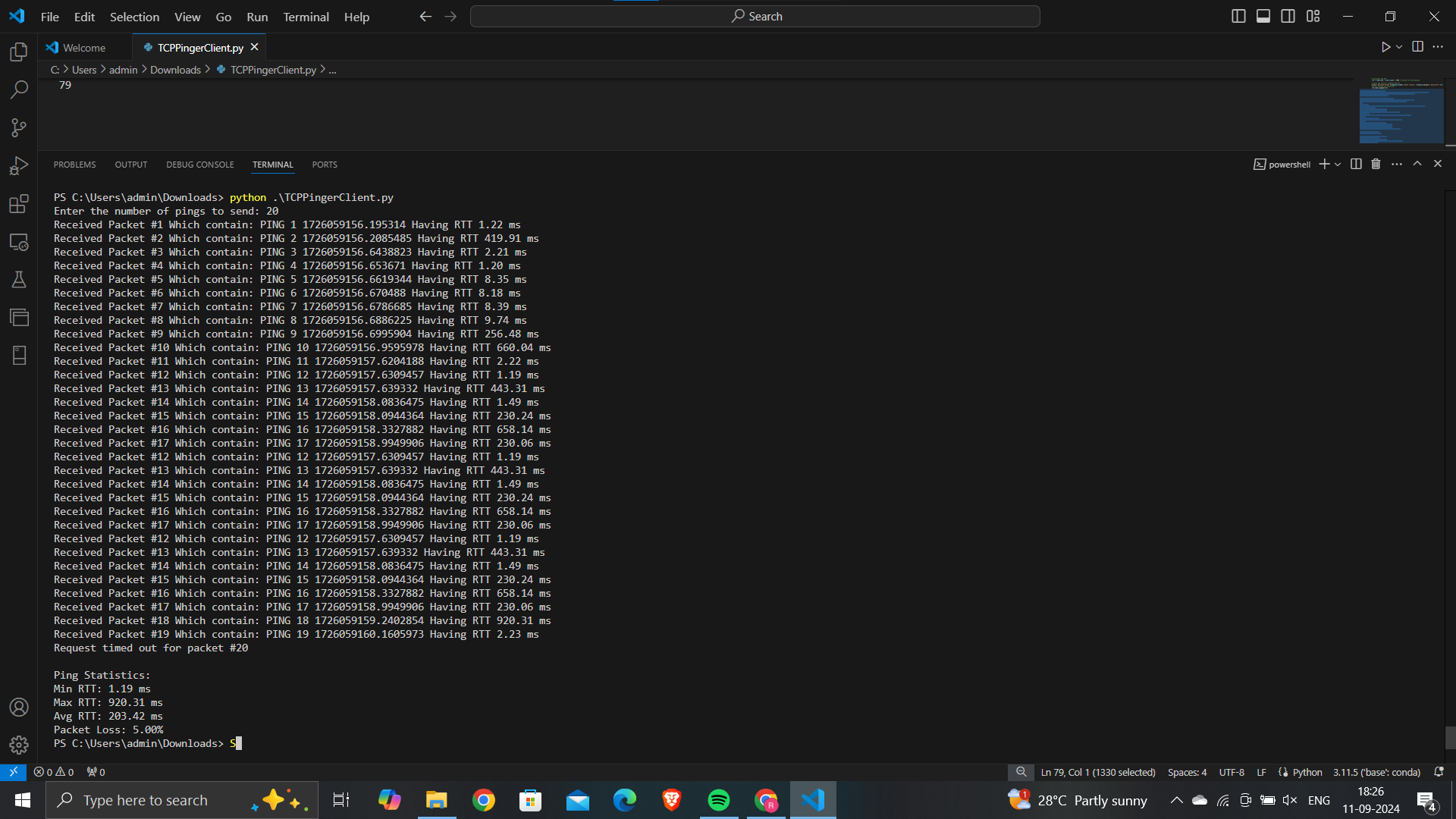


We started client 1 and client 2 at approximately same time and and killed client 1 after sending packet number 12 out of 20 packets in which ping number 11 sent by client at time 1726859161.69599 .

Client 1 (killed client 1 in between)



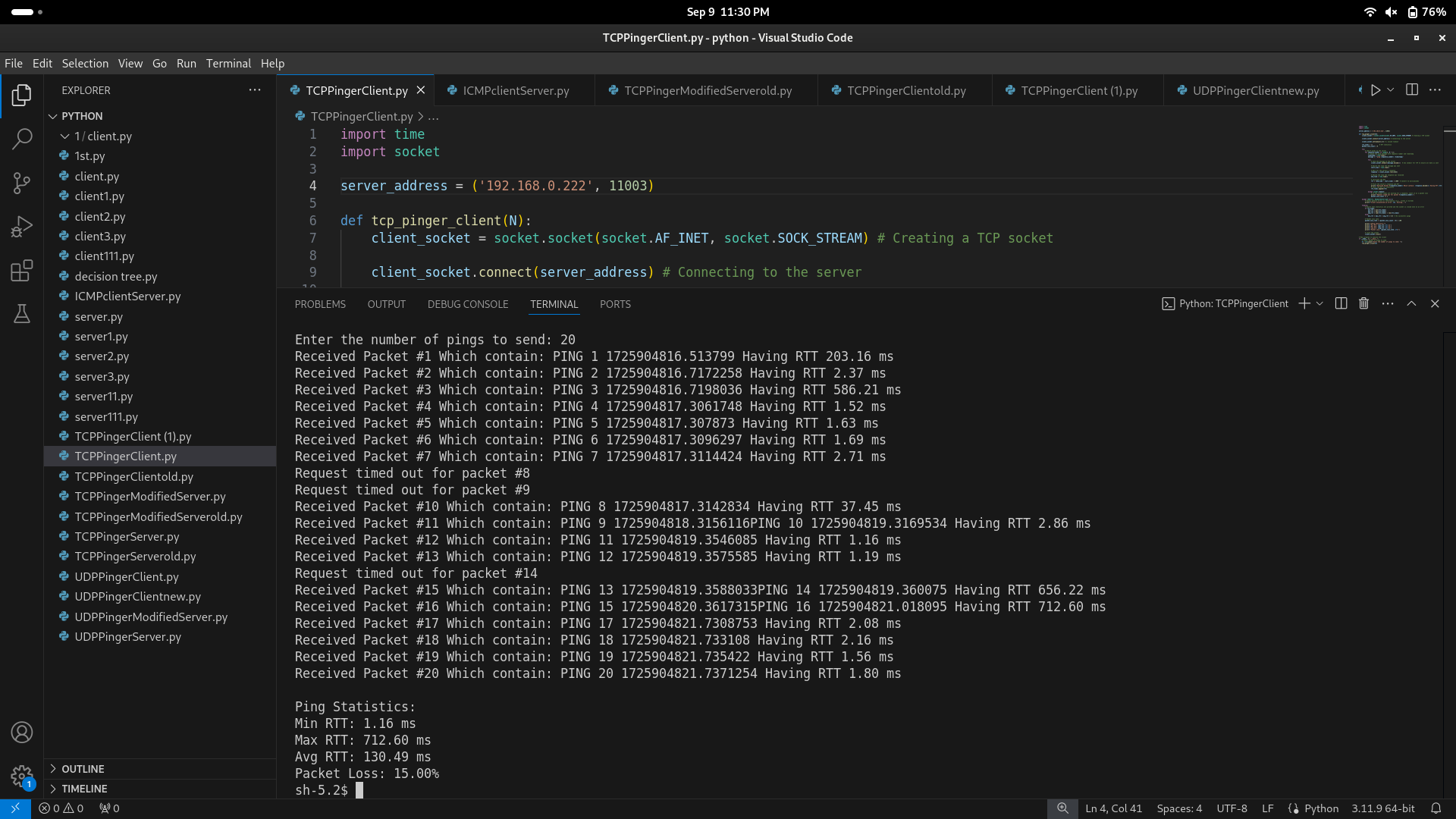
Client 2



Client 2 had started sending 20 pings to the server where first ping is send at time 1726059156.195341, after sending all pings it results in 20% packet loss with minimum RTT 1.19 milliseconds , maximum RTT 920 miliseconds, average RTT 203.42 milisecond.

1. TCP Pinger Client for serverTCPPingerServerModified.py

We send 20 pings to the server out of that ping no. 8,9 and 14 time out occurred. The shortest time recorded for ping number 12 which represents min delay between client request and server’s response. THe longest time seen for ping number 16 which shows significant delay during that instance of time.



Minimum Round Trip Time (RTT) is 1.16 mili-seconds.

Maximum Round Trip Time (RTT) is 712.60 mili-seconds.

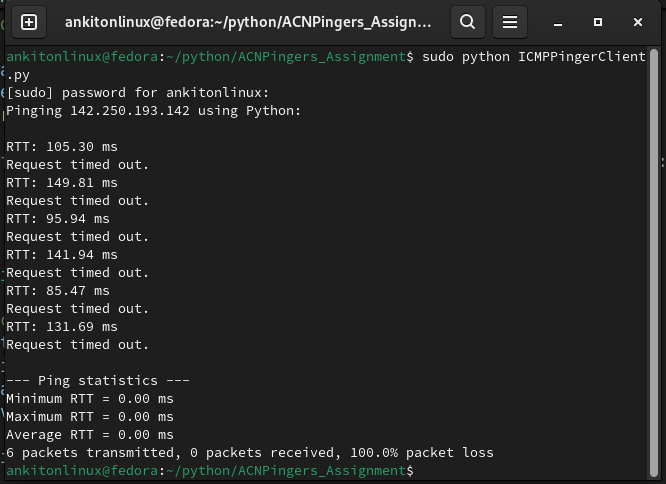
Average Round Trip Time (RTT) is 130.49 mili-seconds.

The rate of packet loss is 15 %.

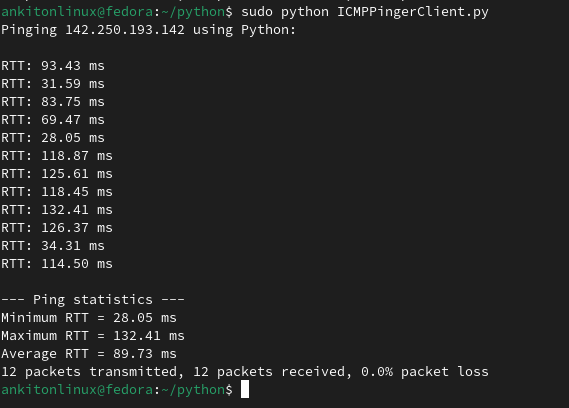
We send 20 pings to the server out of that ping no. 8,9 and 14 time out occurred. The shortest time recorded for ping number 12 which represents min delay between client request and server’s response. THe longest time seen for ping number 16 which shows significant delay during that instance of time.

**ICMP Documentation**

**1)ICMP with google.com**



As we had Skeleton Python Code for the ICMP Pinger given with destination address of google.com where we had to add functions to get the required output, For this we had the requirement of if packets are matching, we had to fetch the information from packet header such as type,checksum,etc and we had to print the RTTs of each ping request along with conclusion of min,max and avg RTT amongs them, also packet loss rate. So to get all of these thing added function of codes and after giving ping request of 6 pings got this as output, it is not the one we wanted as Request timed out is coming after every RTT as there was mistake while running the code.



After fixing the code and sending 12 pings got following results :

Min RTT of 28.05 ms,

Max RTT of 132.41 ms

Avg RTT of 89.73 ms

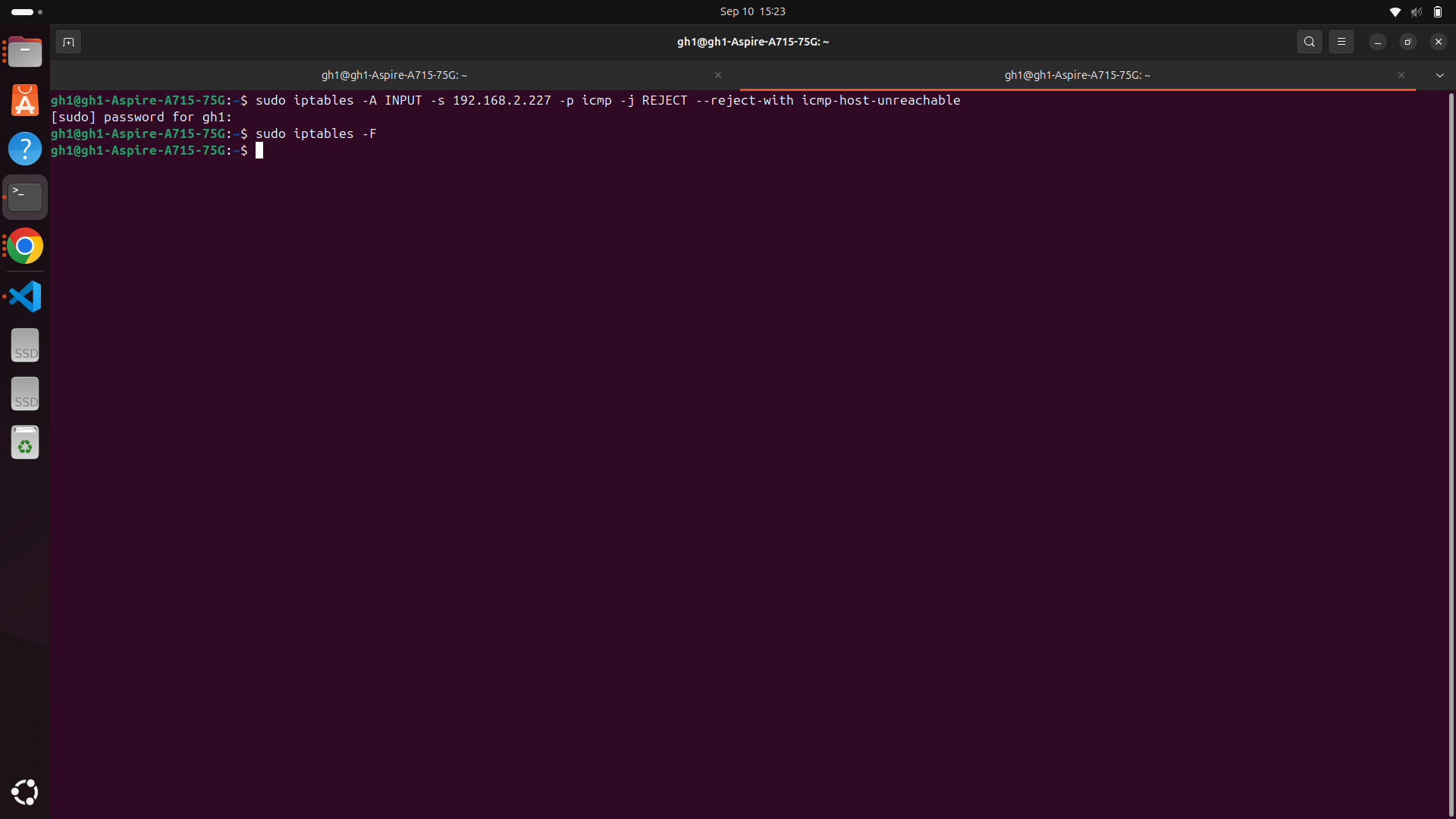
packet loss rate of 0%

These are displayed in the output.

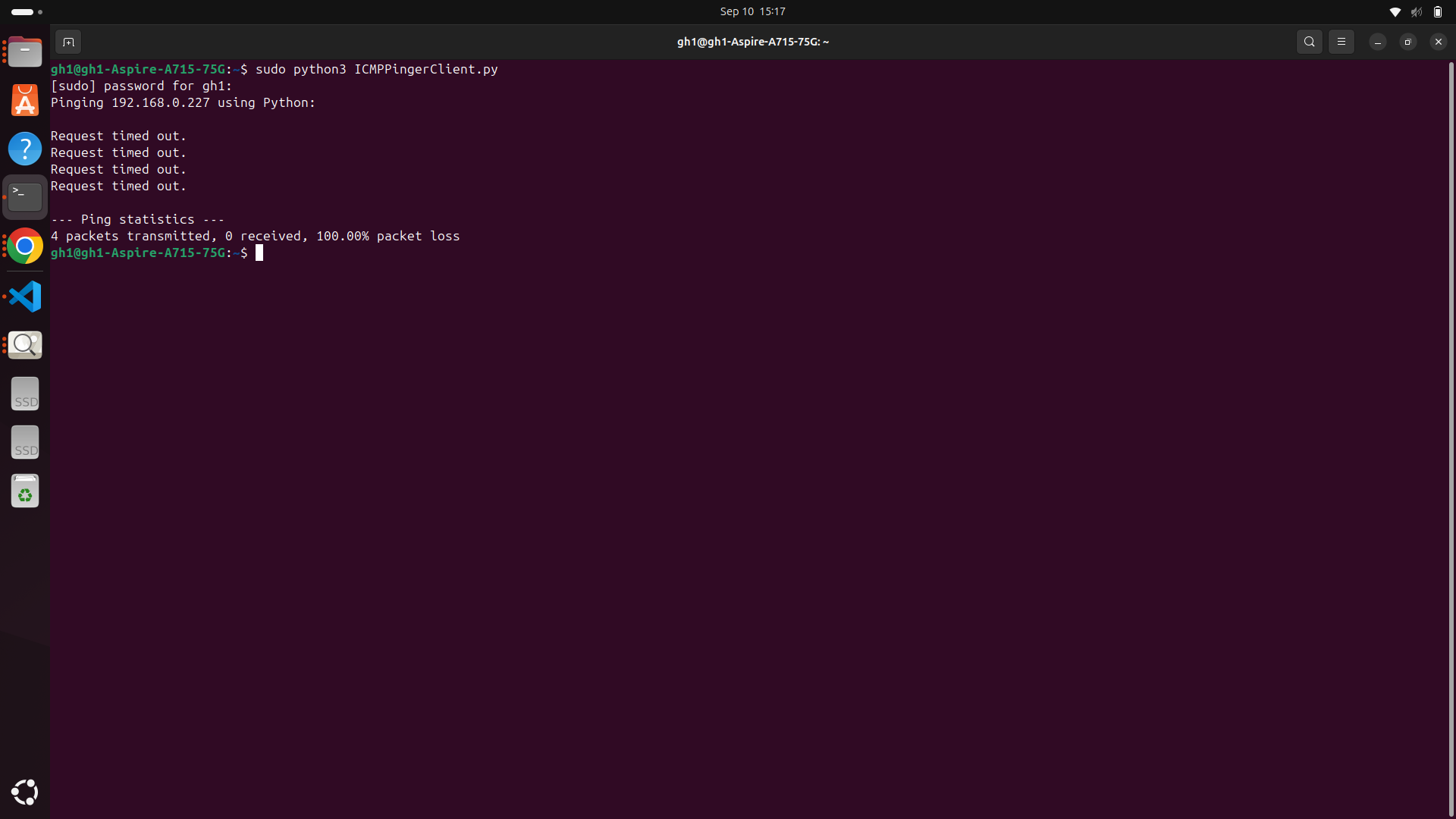
The problem was not getting code structure right thats why earlier got unwanted output.

**2)ICMP with IP blocking**

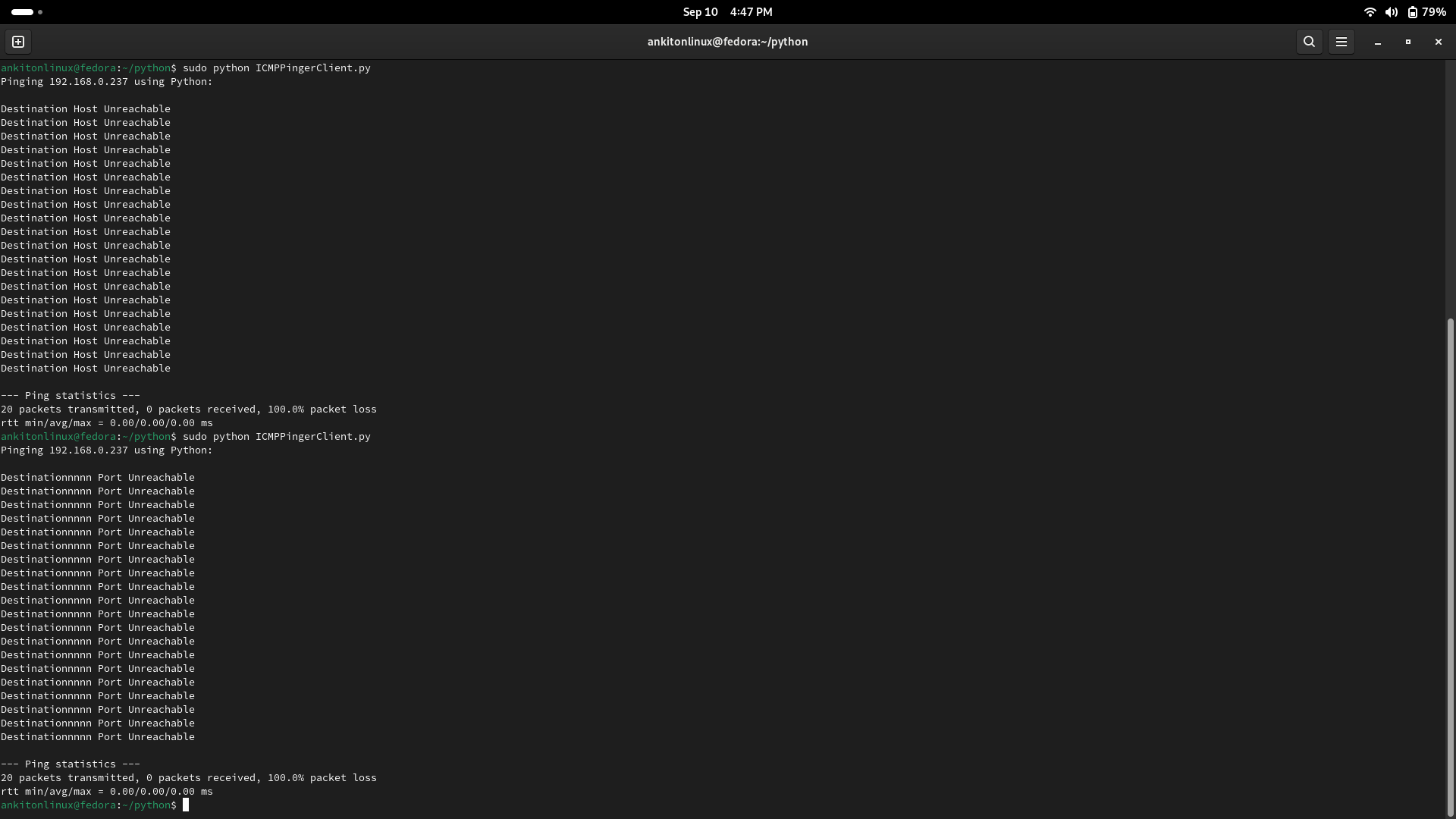
This is what was given to block the host form server machine and to unblock them too, this was properly performed to block client to get error.



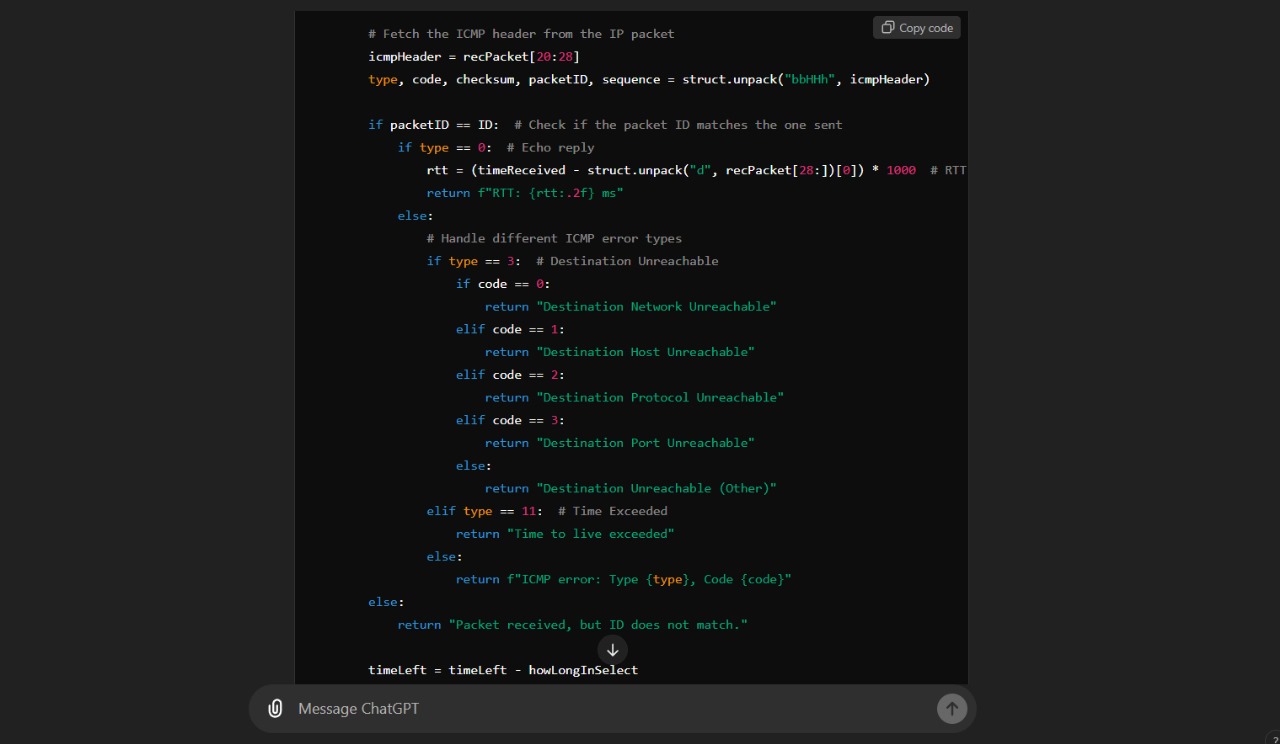
Output we got :



But at 1st when we tried we got all the packets output as Request timed out as can see in the screenshot above. This was because the code was not ok for what we wanted.



Then after fixing the code a little bit we got the output what we wanted. The problem was that we did not got the function well to handle ICMP errors, as there were some minor mistakes like if and else and some statements. So fixed them at first got error as Destination host unreachable after blocking host and got Destination port unreachable after blocking port.



Used LLM chatgpt for code of extracting information from ICMP headers and error handling in ICMP>

**Review** : As we gone through this part of assignment which was about ICMP we learned the functionality of ICMP pinger code which tells about icmp packet transmission structure which is :

**1)Echo Request :** Sending echo request from host to destination where each request includes header and data, abd each header have field as type, checksum etc.

**2)Echo Reply :** After request client gets reply back from server where extracting info of header helps knowing if reply was of the sent packet or not.

**3)Fetching information from header :** To know reply was of sent packet or not these two packets has to match and for that extracting header and matching them is important.

**4)Using iptables for blocking the ip :** To know if client is able to handle the error or not tried creating situation of error by blocking host and port of client using iptables.

**5)Handling ICMP error :** There will be situations when packets will not match or packet will be lost or anything with packet, and then program has to handle the errors. Some of the errors which are included in code as well are Destination host unreachable and Destination port unreachable when we tried by blocking host and port of Client respectively. AL

**6)Calculating RTT and Packet loss rate :** Then made sure whatever packets are transmitted we are able to get RTTs for them along with keeping info of if they reached or not.

| **Task/Section** | **Group Member <Enter Name of Student>** | **Contribution <describe the work done, in brief>** | **Challenges faced (if any** |
| --- | --- | --- | --- |
| **Research & Info gathering** | Ankit Khewale | Reserched info about how echo request and echo reply for ICMP protocol are structured. And gone through some information about ICMP header fields which are type,code,checksum, ID and sequence number. Explored how to use sockets in python, and some iptables commands. | Faced challenge on how will we extract information from header |
| **Code Development** | Ankit Khewale | Added necessary fields in the skeleton of code about extracting the info from header, making sure packet ids are matching and handling the different type of ICMP error. Also added statements to track RTTs of packets, getting max,min and avg of RTTs and packet loss rate . | The main challenge was to code about extracting the info from header, making sure packet ids are matching. Also not able to get required output of Destination Host Unreachable. |
| **Testing & Debugging** | Ankit Khewale | Tested for google.com, and got the RTTs and 0% packet loss, as received all packets. Used iptables command for blocking the ip and getting Destination host unreachable and Destination port unreachable where 0% packets received. | After blocking ip of another machine, had to run multiple times and checked as was not getting Unreachable thing earlier and instead was getting request timed out and packet not matching. |
| **Documentation & Report Writing** | Ankit Khewale | Prepared documentation for ICMP above. |  |
| **Final Review & Submission** | Ankit Khewale | Prepared Final review for ICMP above. |  |

| **Task/Section** | **Group Member <Enter Name of Student>** | **Contribution <describe the work done, in brief>** | **Challenges faced (if any** |
| --- | --- | --- | --- |
|  |  |  |  |
| **Research & Info gathering** | Raju Santhani | Researched and applied code standards to enhance readability and maintainability. Modified the code to follow best practices, including improved commenting and consistent formatting. |  |
| **Code Development** | Raju Santhani | Developed the UDP client code to connect with the server, send ping requests, and track Round Trip Time (RTT). | Had difficulty understanding and utilizing the time.time() function for accurate RTT calculations. |
| **Testing & Debugging** | Raju Santhani | Added error handling to manage invalid inputs, such as zero or non-positive integers. Removed the unused "close" argument from the TCP server code and adjusted the client code accordingly. |  |
| **Documentation & Report Writing** | Raju Santhani | Documented the functionality of the UDP client, the modified UDP server, and the error-handling methods. |  |
| **Final Review & Submission** | Raju Santhani |  |  |

| **Task/Section** | **Group Member <Enter Name of Student>** | **Contribution <describe the work done, in brief** | **Challenges faced (if any** |
| --- | --- | --- | --- |
| **Research & Info gathering** | Gulshan Hatzade | Understanding methods involved in TCP like bind,accept,listen how it differs in UDP,  UDP is faster than TCP | When client sending request from virtual machine to server, the connection was establishing with windows IP, which was not giving response (stucked) |
| **Code Development** | Gulshan Hatzade | Code for TCP client, contributed in UDP client and ICMP | Not getting desired output in terminal as implemented in code due mis handing of if statement |
| **Testing & Debugging** | Gulshan Hatzade | Contributed in testing and debuggig of UDP client,TCP client, ICMP debugging, debugging of TCP modified code by introducing 20% loss by traffic control methods | Getting error ‘request time out’ for TCP client may be due virtual environment- solved by using dual boot system,  Not getting error message like destination host unreachable due to one if statement causing the problem, not getting response even if all things are correct - but this is solved by changing ports on both systems |
| **Documentation & Report Writing** | Gulshan Hatzade | Read me file, Documentation and report of TCP part which includes TCP client, TCP server, TCP modified server |  |
| **Final Review & Submission** | Gulshan Hatzade | Studied TCP, UDP and ICMP protocols |  |

**ANTI-PLAGIARISM Statement**

We certify that this assignment/report is the result of our collaborative work, based on our collective study and research. All sources, including books, articles, software, datasets, reports, and communications, have been properly acknowledged. This work has not been previously submitted for assessment in any other course unless specific permission was granted by all involved instructors. We also acknowledge the use of AI tools, such as LLMs (e.g., ChatGPT), for assistance in refining this assignment, if used. We have ensured that their usage complies with the academic integrity policies of this course. We pledge to uphold the principles of honesty, integrity, and responsibility at CSE@IITH. Additionally, we understand our duty to report any violations of academic integrity by other if we become aware of them.

Names: CS24MTECH11016-CSMTECH14006-SM24MTECH12001

Date: 12/09/2024

Signatures: Ankit Khewale, Gulshan Hatzade, Raju Santhani