ACN Programming-Assignment-1-My Pingers: My Pingers

Students:

1. **Vinaykumar Kadari** (CS24MTECH14008)
2. **P.Rushi Keswar Reddy** (CS24MTECH11018)
3. **Kishor Kumar Patro** (SM24MTECH14001)

Part 1: UDP Pinger

Implementation of UDP Pinger Client :

The UDP pinger client program is implemented by using socket programming. The program takes N as input, which indicates the number of pings the client wants to send to the server. A UDP socket is created, which works on an IPv4 address. To avoid waiting a long time to receive a response from the server, a timeout is set to the client socket for 1 second. An array rtt_time is declared to store the Round trip times of each ping request response. Using Exception Handling in Python, errors are handled by printing out exceptions.

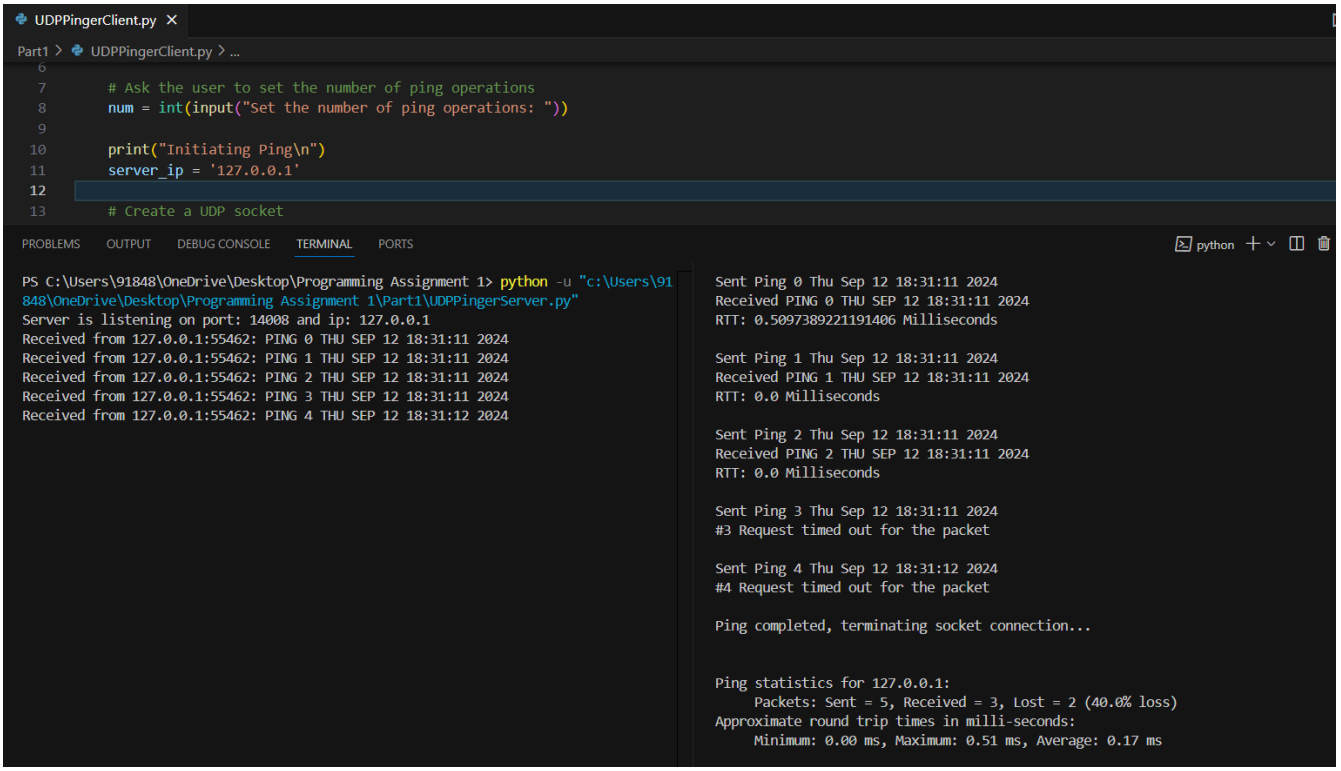
- The internal implementation of sending each ping Request:
- For loop is used to iterate N times. The messages are sent in the following format:

Ping + ith ping + time_stamp

- For calculating the RTT of each ping request, the time module is utilized to get timestamps before sending the request(start) and after getting the response(end). Using sendto method of client socket,encode message is sent to server_address using UDP protocol.
- Using recvfrom method of client socket , response of the request sent is received and time elapsed is determined by subtracting the time stamps start from end.Finally, rtt is calculated and appended into the list.
- If, during the above process, any ping response faces a timeout, an exception is raised, and that request is considered packet loss.
- After n ping requests, the created client socket is closed.
- The minimum, maximum, and average RTTs are calculated from the data.

1) Local Host

The given UDPPingerServer and implemented UDPPingerClient programs are executed on the same machine with the local host having an address as “127.0.01” for debugging purposes. Therefore the round trip time (RTTs) will be low and consistent.



2) Single Client

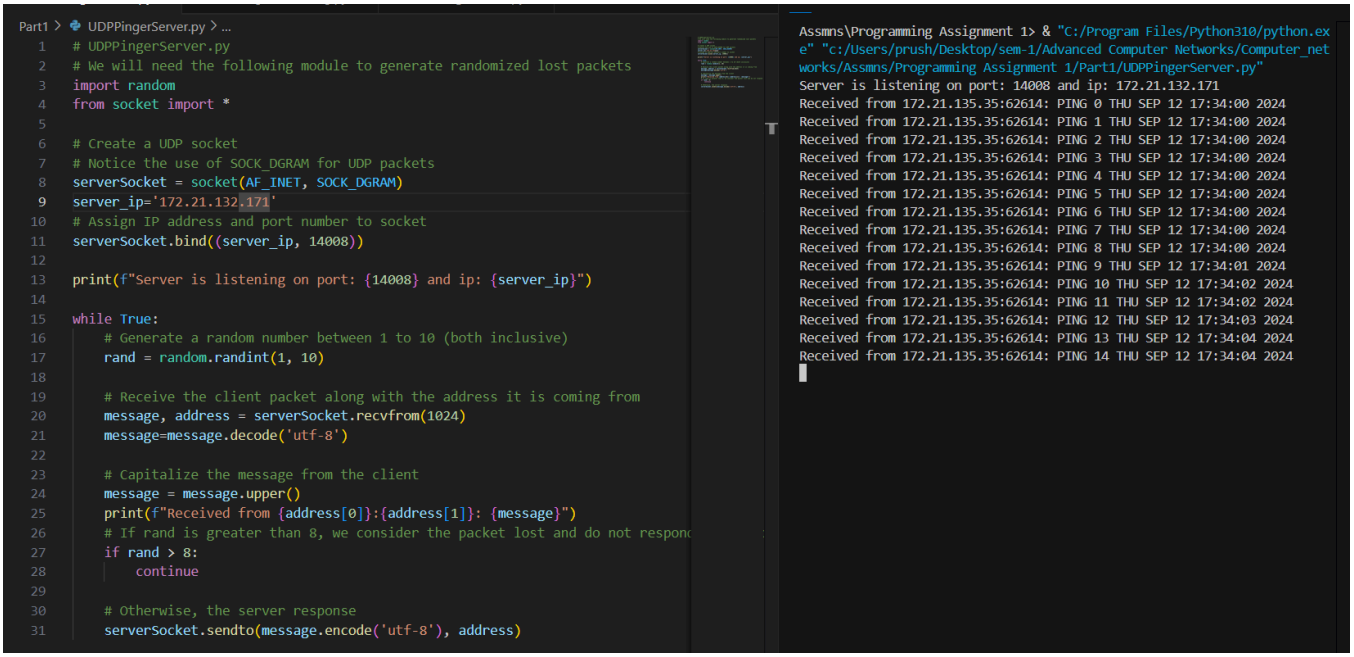
Simulation of UDP packet Loss:

A random module in Python is used to simulate the packet loss at the server end.

Packet loss simulation using random module:

Random integers between 1 to 10 are generated, if the generated integer is more than 8, then it is considered as packet loss, and the server won't send the response packet back to the client. Given server code is modified a little bit to achieve the above simulation.

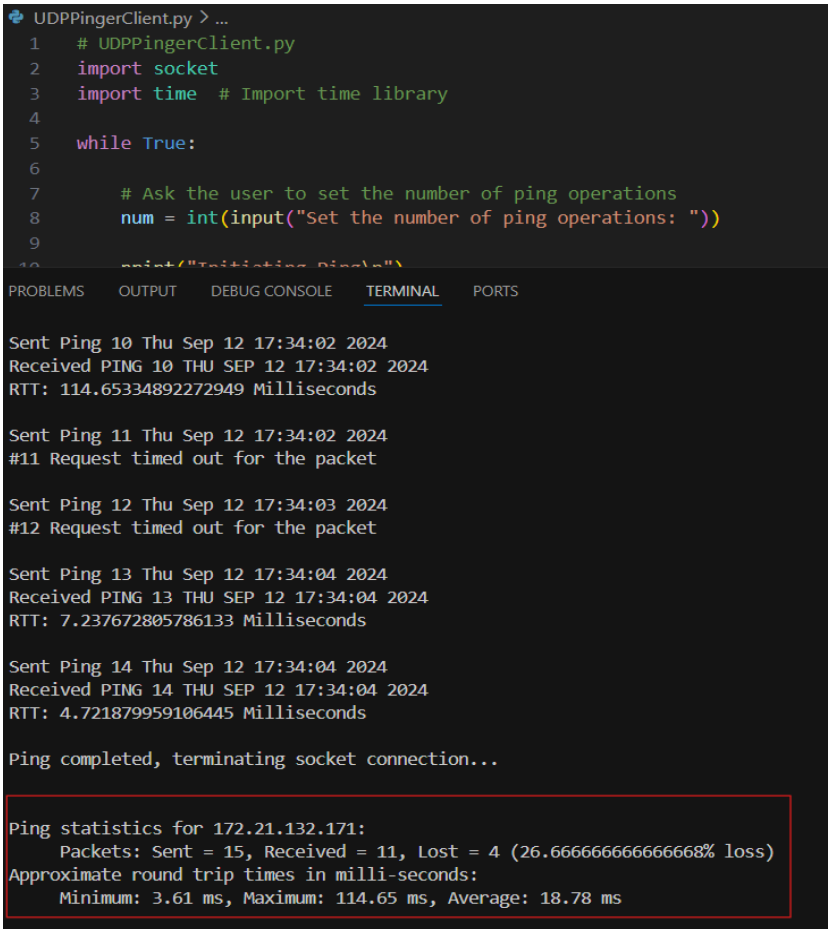
Server Output:



```
Part1 > UDPPingerServer.py > ...
# UDPPingerServer.py
# We will need the following module to generate randomized lost packets
import random
from socket import *

# Create a UDP socket
# Notice the use of SOCK_DGRAM for UDP packets
serverSocket = socket(AF_INET, SOCK_DGRAM)
server_ip='172.21.132.171'
# Assign IP address and port number to socket
serverSocket.bind((server_ip, 14008))

print(f"Server is listening on port: {14008} and ip: {server_ip}")

while True:
    # Generate a random number between 1 to 10 (both inclusive)
    rand = random.randint(1, 10)

    # Receive the client packet along with the address it is coming from
    message, address = serverSocket.recvfrom(1024)
    message=message.decode('utf-8')

    # Capitalize the message from the client
    message = message.upper()
    print(f"Received from {address[0]}:{address[1]}: {message}")
    # If rand is greater than 8, we consider the packet lost and do not respond
    if rand > 8:
        continue

    # Otherwise, the server response
    serverSocket.sendto(message.encode('utf-8'), address)
```

```
Assmns\Programming Assignment 1> & "C:/Program Files/Python310/python.exe" "c:/Users/prush/Desktop/sem-1/Advanced Computer Networks/Computer_networks/Assmns/Programming Assignment 1/Part1/UDPPingerServer.py"
Server is listening on port: 14008 and ip: 172.21.132.171
Received from 172.21.135.35:62614: PING 0 THU SEP 12 17:34:00 2024
Received from 172.21.135.35:62614: PING 1 THU SEP 12 17:34:00 2024
Received from 172.21.135.35:62614: PING 2 THU SEP 12 17:34:00 2024
Received from 172.21.135.35:62614: PING 3 THU SEP 12 17:34:00 2024
Received from 172.21.135.35:62614: PING 4 THU SEP 12 17:34:00 2024
Received from 172.21.135.35:62614: PING 5 THU SEP 12 17:34:00 2024
Received from 172.21.135.35:62614: PING 6 THU SEP 12 17:34:00 2024
Received from 172.21.135.35:62614: PING 7 THU SEP 12 17:34:00 2024
Received from 172.21.135.35:62614: PING 8 THU SEP 12 17:34:00 2024
Received from 172.21.135.35:62614: PING 9 THU SEP 12 17:34:01 2024
Received from 172.21.135.35:62614: PING 10 THU SEP 12 17:34:02 2024
Received from 172.21.135.35:62614: PING 11 THU SEP 12 17:34:02 2024
Received from 172.21.135.35:62614: PING 12 THU SEP 12 17:34:03 2024
Received from 172.21.135.35:62614: PING 13 THU SEP 12 17:34:04 2024
Received from 172.21.135.35:62614: PING 14 THU SEP 12 17:34:04 2024
```

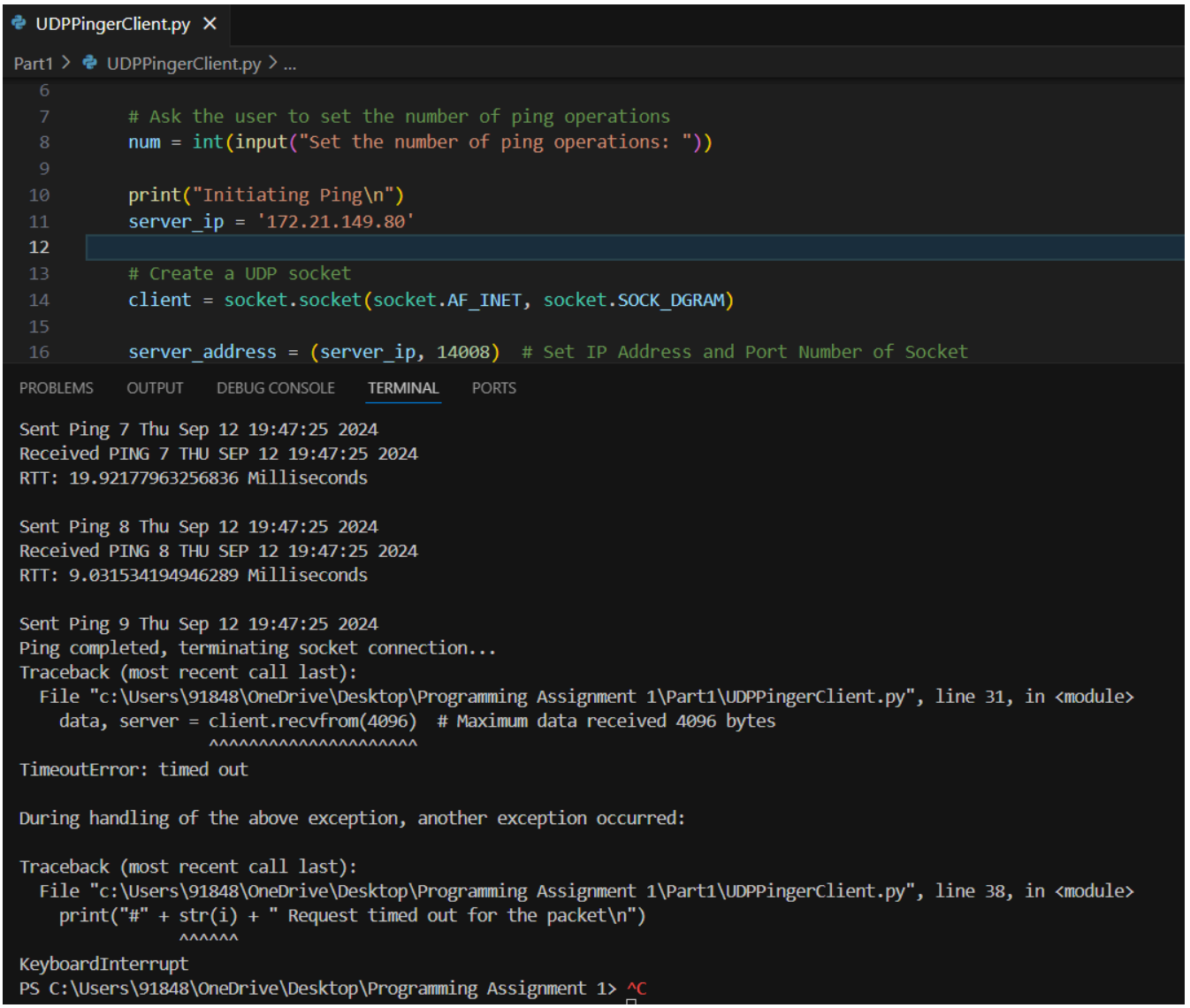
Client Output:



```
UDPPingerClient.py > ...
# UDPPingerClient.py
import socket
import time  # Import time library

while True:

    # Ask the user to set the number of ping operations
    num = int(input("Set the number of ping operations: "))
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Sent Ping 10 Thu Sep 12 17:34:02 2024
Received PING 10 THU SEP 12 17:34:02 2024
RTT: 114.65334892272949 Milliseconds

Sent Ping 11 Thu Sep 12 17:34:02 2024
#11 Request timed out for the packet

Sent Ping 12 Thu Sep 12 17:34:03 2024
#12 Request timed out for the packet

Sent Ping 13 Thu Sep 12 17:34:04 2024
Received PING 13 THU SEP 12 17:34:04 2024
RTT: 7.237672805786133 Milliseconds

Sent Ping 14 Thu Sep 12 17:34:04 2024
Received PING 14 THU SEP 12 17:34:04 2024
RTT: 4.721879959106445 Milliseconds

Ping completed, terminating socket connection...

Ping statistics for 172.21.132.171:
    Packets: Sent = 15, Received = 11, Lost = 4 (26.666666666666668% loss)
Approximate round trip times in milli-seconds:
    Minimum: 3.61 ms, Maximum: 114.65 ms, Average: 18.78 ms
```

3) Client Killed



4) <u>Emulation of UDP packet loss at NIC by tc-netem</u>

To emulate packet loss at NIC of the machine where server code is running, the following commands are

sudo tc qdisc add dev enp0s3 root netem loss 20%

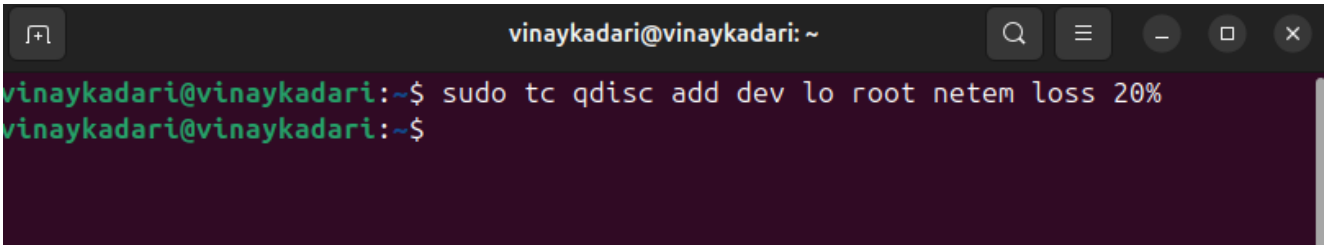
This command sets up 20% packet loss on the wifi (ethernet) interface enp0s3 of the machine running server code.

tc qdisc show dev enp0s3

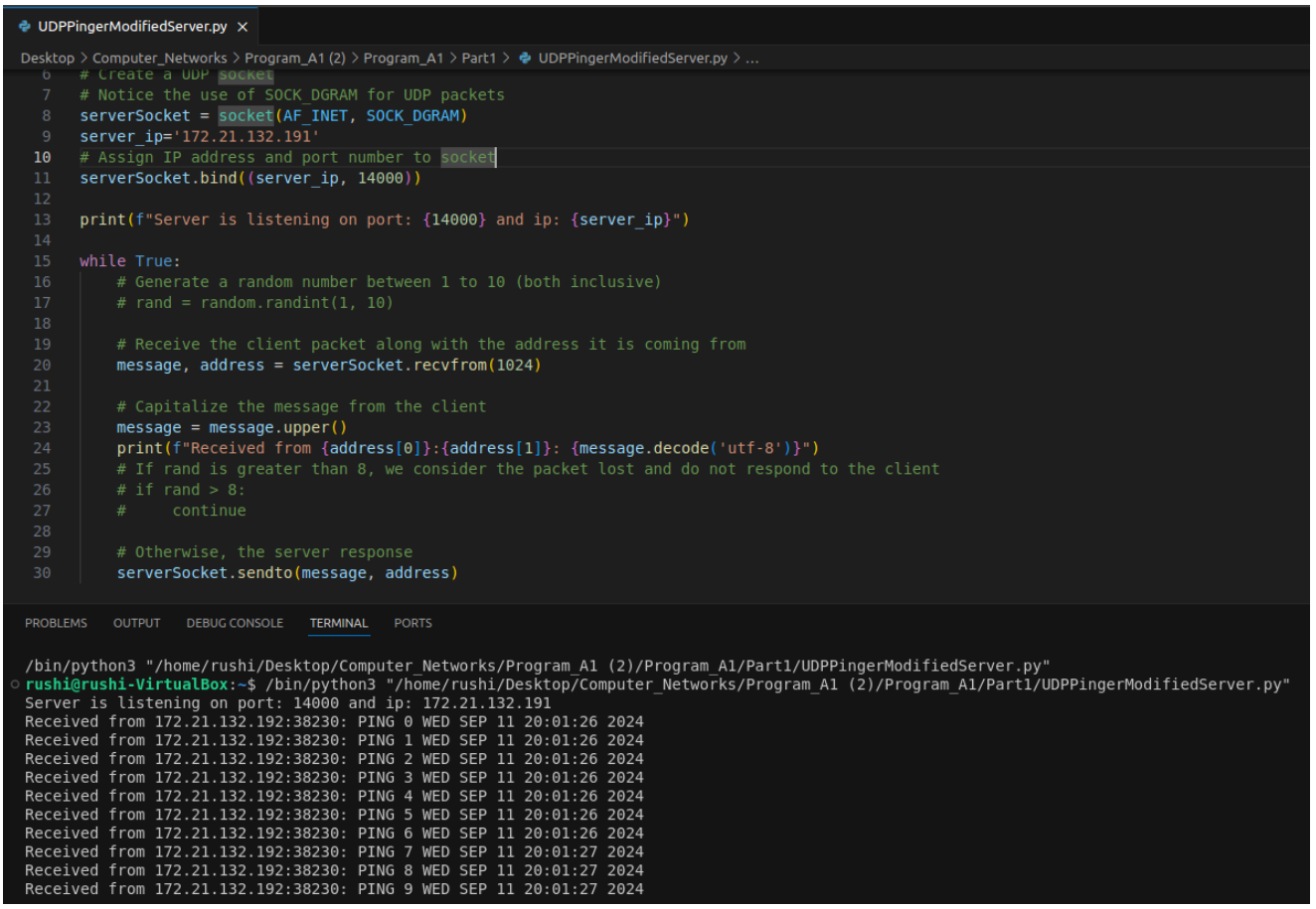
This command verifies the traffic control rule at the Wi-Fi interface

sudo tc qdisc del dev enp0s3 root

This command is finally used to delete the setup loss at NIC.



Server Output:



Client Output:

UDPPingerClient.py

Part1 > UDPPingerClient.py > ...

```
13      # Create a UDP socket
14      client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
15
16      server_address = (server_ip, 14000)  # Set IP Address and Port Number of Socket
17
18      client.settimeout(1)  # Sets a timeout value of 1 second
19
20      rtt_time = []  # List to store Round-Trip Times (RTTs)
21      packet_lost = 0  # Count of lost (timed-out) pings
22
23      try:
24          # Loop to ping the server 'num_pings' times
25          for i in range(num):
26              start = time.time()  # Start time when message is sent to server
27              message = 'Ping ' + str(i) + " " + time.ctime(start)
28              try:
29                  sent = client.sendto(message.encode("utf-8"), server_address)
30                  print("Sent " + message)
31                  data, server = client.recvfrom(4096)  # Maximum data received 4096 bytes
32                  print("Received " + str(data.decode("utf-8")))
```

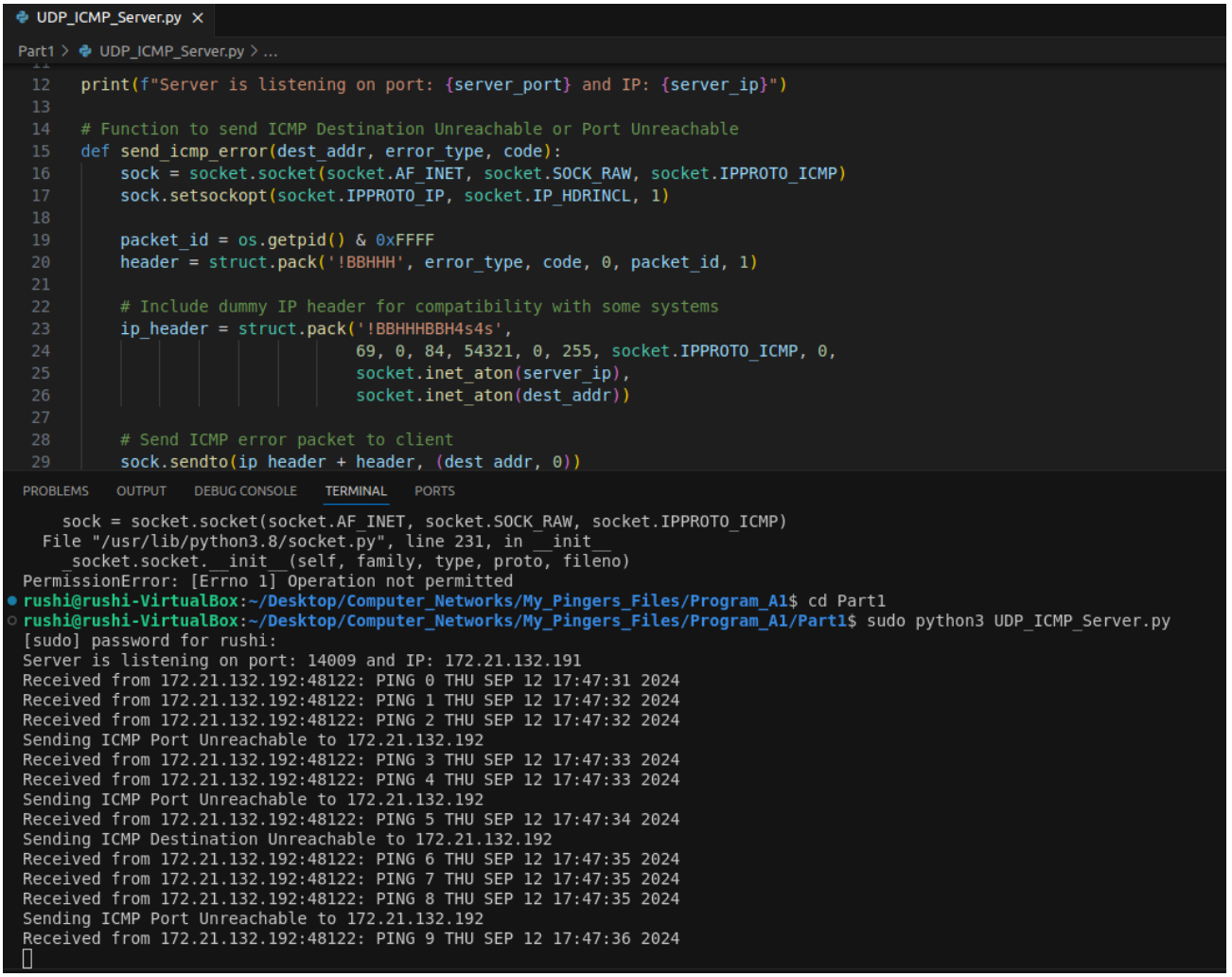
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Ping completed, terminating socket connection...


Ping statistics for 172.21.132.191:
     Packets: Sent = 10, Received = 9, Lost = 1 (10.0% loss)
Approximate round trip times in milli-seconds:
     Minimum: 4.22 ms, Maximum: 57.62 ms, Average: 10.95 ms
```

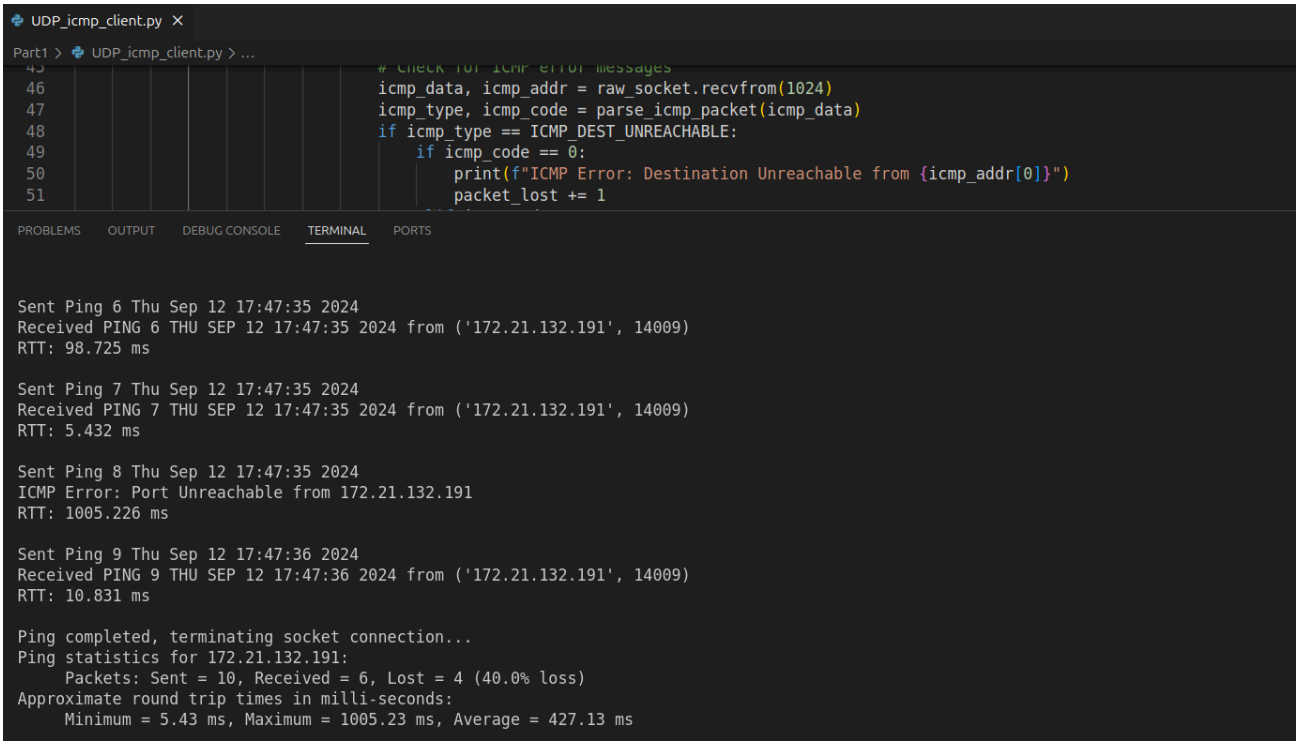
5) Error Handling

As done in packet loss simulation, the random module simulates ICMP error responses. Randint function generates integers from 1 to 10.If the value generated is less than are equal to 6 then the response is received successfully.if the value is greater than 6 and less than and equal to 8 then it is considered as “ICMP Destination Unreachable,” and if the value is greater than 8 then it is considered as “ICMP Port Unreachable”.For these cases, a raw socket is created to send an ICMP error packet to the client.

Server output



Client Output:



Part 2: TCP Pinger

Implementation of TCP Pinger Client and Server:

In this packet we are using TCP socket instead of UDP. TCP provides a **reliable** and **connection-oriented communication** channel. Both client and server pingers are implemented. Along with this, multithreading is also incorporated into server code to deal with multiple clients concurrently.

Outline of the TCP pinger client Implementation:

a) Creation of socket.
b) Establishment of the TCP connection with server address and port number.
c) Sending Ping requests to the server.
d) Response receiving and Error handling in case of Exception
e) Calculation of minimum, maximum, and Average RTT values.Packet loss rate is also calculated based on the number of packets lost.
f) Closing Connection and Socket.

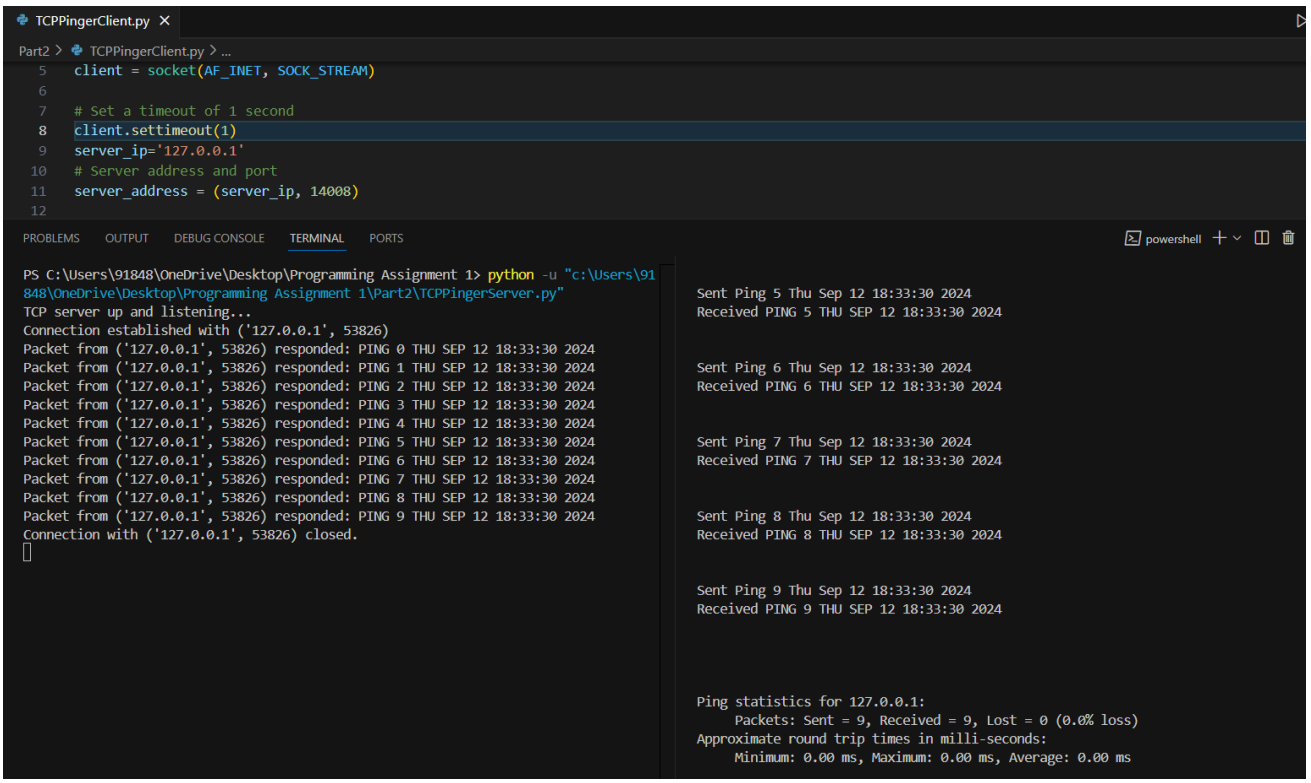
Outline of the TCP server Implementation:

a) Creation of Server Socket.
b) Binding server address and Port and start listening for incoming connection.
c) Connection acceptance and processing of request
d) Sending response and Error Handling
e) Closing connection

Results:

1) Local Host

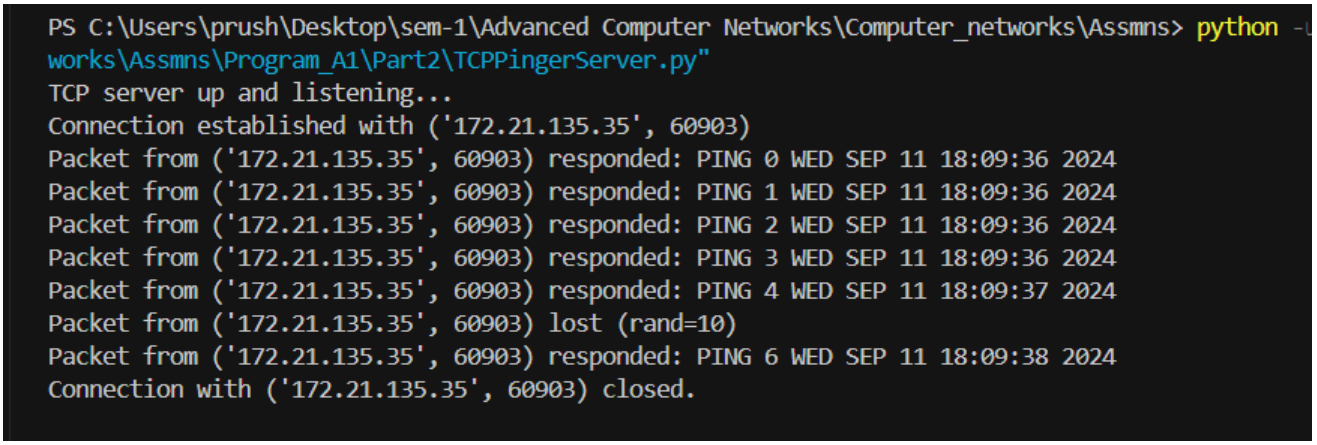
Running both TCP pinger server and TCP pinger client both on same local machine with local server address “127.0.0.1”.



2) Single Client

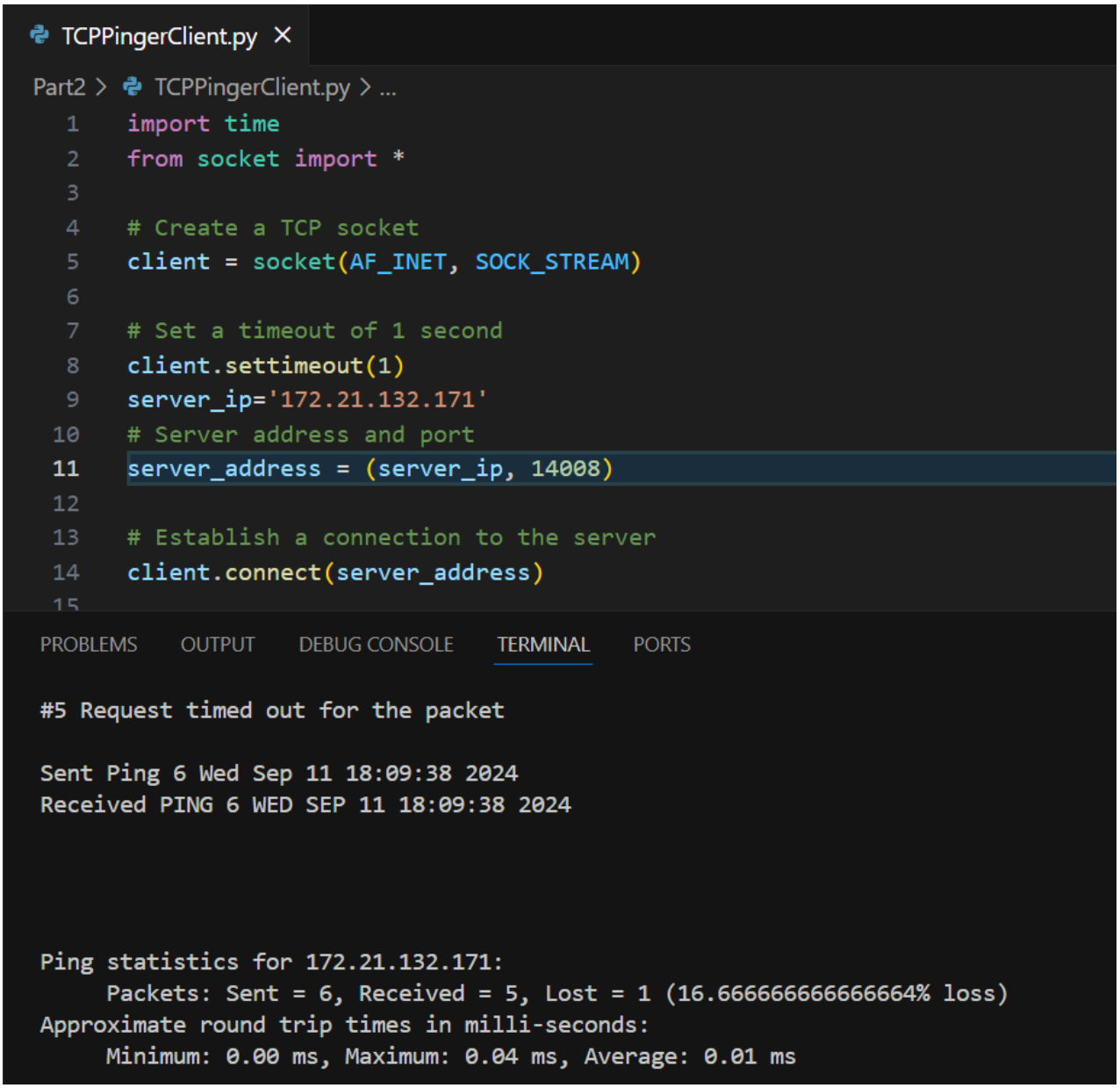
Server code and client code are executed on two different machines connected to the same network. The packet loss is simulated by random() function of random module in python.

Server Output:



Client Output:

As observed below, there is one packet lost occurred. The RTT statistics of all Ping Packets are displayed.



3) Emulation of TCP packet loss at NIC by tc-netem

Instead of using a random module in Python,tc-netem utility in linux is used to inject a loss of 20% at NIC of machine at which server code is running.The following packet losses are observed.

Server Output:

```
TCP server up and listening...
Connection established with ('192.168.167.16', 51397)
Packet from ('192.168.167.16', 51397) responded: PING 1 THU SEP 12 20:39:46 2024
Packet from ('192.168.167.16', 51397) responded: PING 2 THU SEP 12 20:39:46 2024
Packet from ('192.168.167.16', 51397) responded: PING 3 THU SEP 12 20:39:46 2024
Packet from ('192.168.167.16', 51397) responded: PING 4 THU SEP 12 20:39:46 2024
Packet from ('192.168.167.16', 51397) responded: PING 5 THU SEP 12 20:39:46 2024
Packet from ('192.168.167.16', 51397) responded: PING 6 THU SEP 12 20:39:46 2024
Packet from ('192.168.167.16', 51397) responded: PING 10 THU SEP 12 20:39:49 2024
Connection with ('192.168.167.16', 51397) closed.
```

Client Output:

Out of 10 packets, nearly 4 packets are lost, which is simulated with tc-netem in linux.

TCPPingerClient.py

Part2 > TCPPingerClient.py > ...

```
1   import time
2   from socket import *
3
4   # Create a TCP socket
5   client = socket(AF_INET, SOCK_STREAM)
6
7   # Set a timeout of 1 second
8   client.settimeout(1)
9   server_ip='192.168.167.187'
10  # Server address and port
11  server_address = (server_ip, 14008)
12
13  # Establish a connection to the server
14  client.connect(server_address)
15
```

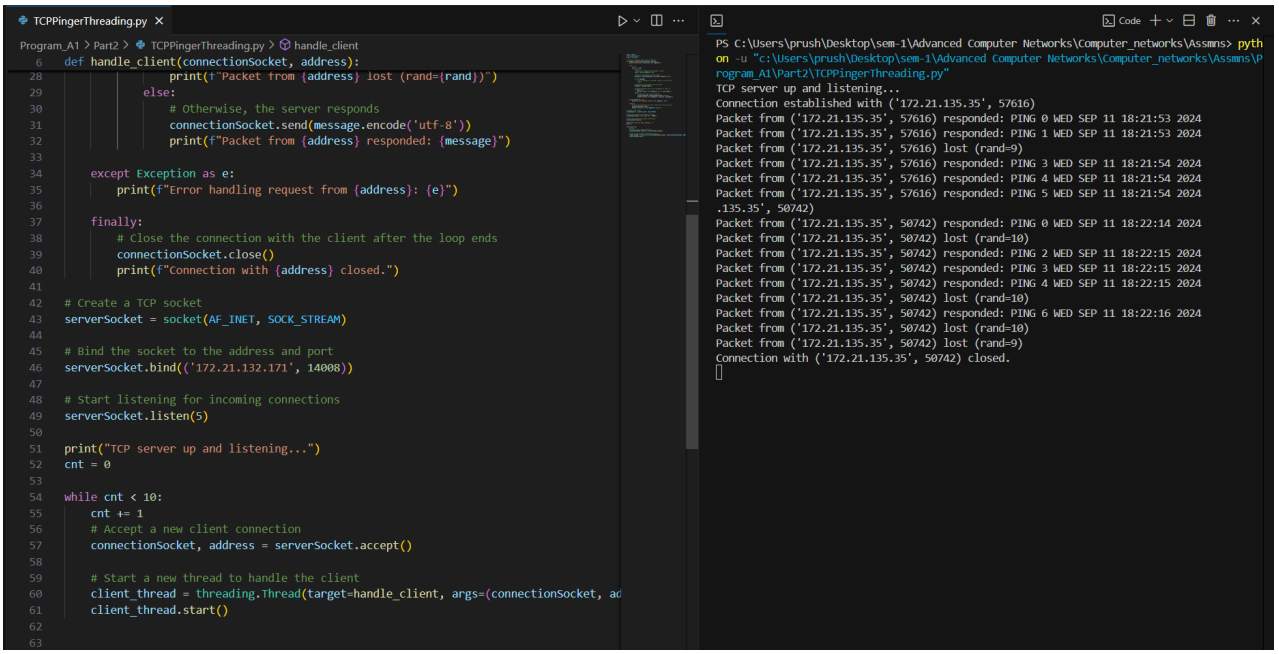
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Sent Ping 5 Thu Sep 12 20:39:46 2024
Received PING 5 THU SEP 12 20:39:46 2024


Sent Ping 6 Thu Sep 12 20:39:46 2024
Received PING 6 THU SEP 12 20:39:46 2024


Sent Ping 7 Thu Sep 12 20:39:46 2024
#7 Request timed out for the packet

Sent Ping 8 Thu Sep 12 20:39:47 2024
#8 Request timed out for the packet

Sent Ping 9 Thu Sep 12 20:39:48 2024
#9 Request timed out for the packet

Sent Ping 10 Thu Sep 12 20:39:49 2024
Received PING 10 THU SEP 12 20:39:49 2024



Ping statistics for 192.168.167.187:
    Packets: Sent = 10, Received = 6, Lost = 4 (40.0% loss)
Approximate round trip times in milli-seconds:
    Minimum: 0.00 ms, Maximum: 0.22 ms, Average: 0.05 ms
```

4) Multi-Client TCP Server using Threading

To handle multiple clients, threading is used at the server end. Each thread is used to handle a single client. Based on the number of clients, those many threads are created by the os of the server to manage all connections concurrently

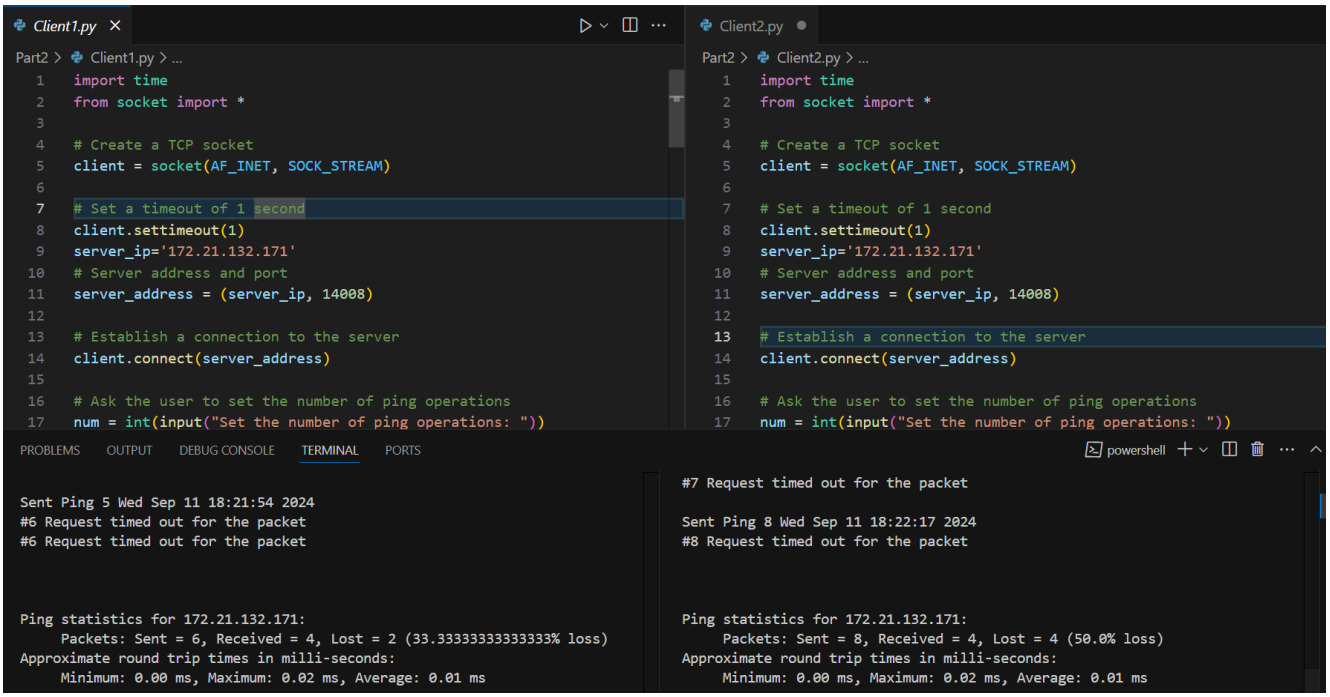
Server



```
def handle_client(connectionSocket, address):
                print(f"Packet from {address} lost (rand={rand})")
            else:
                # Otherwise, the server responds
                connectionSocket.send(message.encode('utf-8'))
                print(f"Packet from {address} responded: {message}")

    except Exception as e:
        print(f"Error handling request from {address}: {e}")

    finally:
        # Close the connection with the client after the loop ends
        connectionSocket.close()
        print(f"Connection with {address} closed.")

# Create a TCP socket
serverSocket = socket(AF_INET, SOCK_STREAM)

# Bind the socket to the address and port
serverSocket.bind(('172.21.132.171', 14008))

# Start listening for incoming connections
serverSocket.listen(5)

print("TCP server up and listening...")
cnt = 0

while cnt < 10:
    cnt += 1
    # Accept a new client connection
    connectionSocket, address = serverSocket.accept()

    # Start a new thread to handle the client
    client_thread = threading.Thread(target=handle_client, args=(connectionSocket, ad
    client_thread.start()
```

```
PS C:\Users\prush\Desktop\sem-1\Advanced Computer Networks\Computer_networks\Assmns> pyth
on -u "c:\Users\prush\Desktop\sem-1\Advanced Computer Networks\Computer_networks\Assmns\P
rogram_A1\Part2\TCPPingerThreading.py"
TCP server up and listening...
Connection established with ('172.21.135.35', 57616)
Packet from ('172.21.135.35', 57616) responded: PING 0 WED SEP 11 18:21:53 2024
Packet from ('172.21.135.35', 57616) responded: PING 1 WED SEP 11 18:21:53 2024
Packet from ('172.21.135.35', 57616) lost (rand=9)
Packet from ('172.21.135.35', 57616) responded: PING 3 WED SEP 11 18:21:54 2024
Packet from ('172.21.135.35', 57616) responded: PING 4 WED SEP 11 18:21:54 2024
Packet from ('172.21.135.35', 57616) responded: PING 5 WED SEP 11 18:21:54 2024
.135.35', 50742)
Packet from ('172.21.135.35', 50742) responded: PING 0 WED SEP 11 18:22:14 2024
Packet from ('172.21.135.35', 50742) lost (rand=10)
Packet from ('172.21.135.35', 50742) responded: PING 2 WED SEP 11 18:22:15 2024
Packet from ('172.21.135.35', 50742) responded: PING 3 WED SEP 11 18:22:15 2024
Packet from ('172.21.135.35', 50742) responded: PING 4 WED SEP 11 18:22:15 2024
Packet from ('172.21.135.35', 50742) lost (rand=10)
Packet from ('172.21.135.35', 50742) responded: PING 6 WED SEP 11 18:22:16 2024
Packet from ('172.21.135.35', 50742) lost (rand=10)
Packet from ('172.21.135.35', 50742) lost (rand=9)
Connection with ('172.21.135.35', 50742) closed.
```

Client : Multiple client requests through different port numbers to the server.



```
import time
from socket import *

# Create a TCP socket
client = socket(AF_INET, SOCK_STREAM)

# Set a timeout of 1 second
client.settimeout(1)
server_ip='172.21.132.171'
# Server address and port
server_address = (server_ip, 14008)

# Establish a connection to the server
client.connect(server_address)

# Ask the user to set the number of ping operations
num = int(input("Set the number of ping operations: "))
```

```
import time
from socket import *

# Create a TCP socket
client = socket(AF_INET, SOCK_STREAM)

# Set a timeout of 1 second
client.settimeout(1)
server_ip='172.21.132.171'
# Server address and port
server_address = (server_ip, 14008)

# Establish a connection to the server
client.connect(server_address)

# Ask the user to set the number of ping operations
num = int(input("Set the number of ping operations: "))
```

```
Sent Ping 5 Wed Sep 11 18:21:54 2024
#6 Request timed out for the packet
#6 Request timed out for the packet


Ping statistics for 172.21.132.171:
    Packets: Sent = 6, Received = 4, Lost = 2 (33.33333333333333% loss)
Approximate round trip times in milli-seconds:
    Minimum: 0.00 ms, Maximum: 0.02 ms, Average: 0.01 ms
```

```
#7 Request timed out for the packet

Sent Ping 8 Wed Sep 11 18:22:17 2024
#8 Request timed out for the packet


Ping statistics for 172.21.132.171:
    Packets: Sent = 8, Received = 4, Lost = 4 (50.0% loss)
Approximate round trip times in milli-seconds:
    Minimum: 0.00 ms, Maximum: 0.02 ms, Average: 0.01 ms
```

5) TCP with ICMP error handling

Server

TCP_icmp_server.py

```
Part2 > TCP_icmp_server.py > ...
21
22  # Bind the TCP socket
23  serverSocket = socket(AF_INET, SOCK_STREAM)
24  serverSocket.bind(('172.21.132.192', 14008))
25  serverSocket.listen(5)
26
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
vinaykadari@vinaykadari:~/Downloads/Program_A1 (2)/Program_A1$ cd Part1
vinaykadari@vinaykadari:~/Downloads/Program_A1 (2)/Program_A1/Part1$ cd ..
vinaykadari@vinaykadari:~/Downloads/Program_A1 (2)/Program_A1$ cd Part2
vinaykadari@vinaykadari:~/Downloads/Program_A1 (2)/Program_A1/Part2$ sudo python3 TCP_icmp_server.py
[sudo] password for vinaykadari:
TCP server up and listening...
Connection established with ('172.21.132.191', 52836)
Packet from ('172.21.132.191', 52836) responded: PING 0 THU SEP 12 18:06:42 2024
Packet from ('172.21.132.191', 52836) responded: PING 1 THU SEP 12 18:06:42 2024
Packet from ('172.21.132.191', 52836) responded: PING 2 THU SEP 12 18:06:42 2024
Packet from ('172.21.132.191', 52836) responded: PING 3 THU SEP 12 18:06:42 2024
Packet from ('172.21.132.191', 52836) responded: PING 4 THU SEP 12 18:06:42 2024
Sending ICMP Destination Unreachable to ('172.21.132.191', 52836) (rand=8)
Packet from ('172.21.132.191', 52836) responded: PING 6 THU SEP 12 18:06:43 2024
Packet from ('172.21.132.191', 52836) responded: PING 7 THU SEP 12 18:06:43 2024
Sending ICMP Destination Unreachable to ('172.21.132.191', 52836) (rand=7)
Sending ICMP Port Unreachable to ('172.21.132.191', 52836) (rand=9)
Sending ICMP Destination Unreachable to ('172.21.132.191', 52836) (rand=8)
Connection with ('172.21.132.191', 52836) closed.
```

Client

TCP_ICMP_Client.py

```
Part2 > TCP_ICMP_Client.py > ...
1   import time
2   from socket import *
3
4   # Create a TCP socket
5   client = socket(AF_INET, SOCK_STREAM)
6
7   # Set a timeout of 1 secondS
8   client.settimeout(1)
9   server_ip = '172.21.132.192'
10  # Server address and port
11  server_address = (server_ip, 14008)
12
13  # Establish a connection to the server
14  client.connect(server_address)
15
16  # Ask the user to set the number of ping operations
17  num = int(input("Set the number of ping operations: "))
18
19  print("Initiating Ping\n")
```

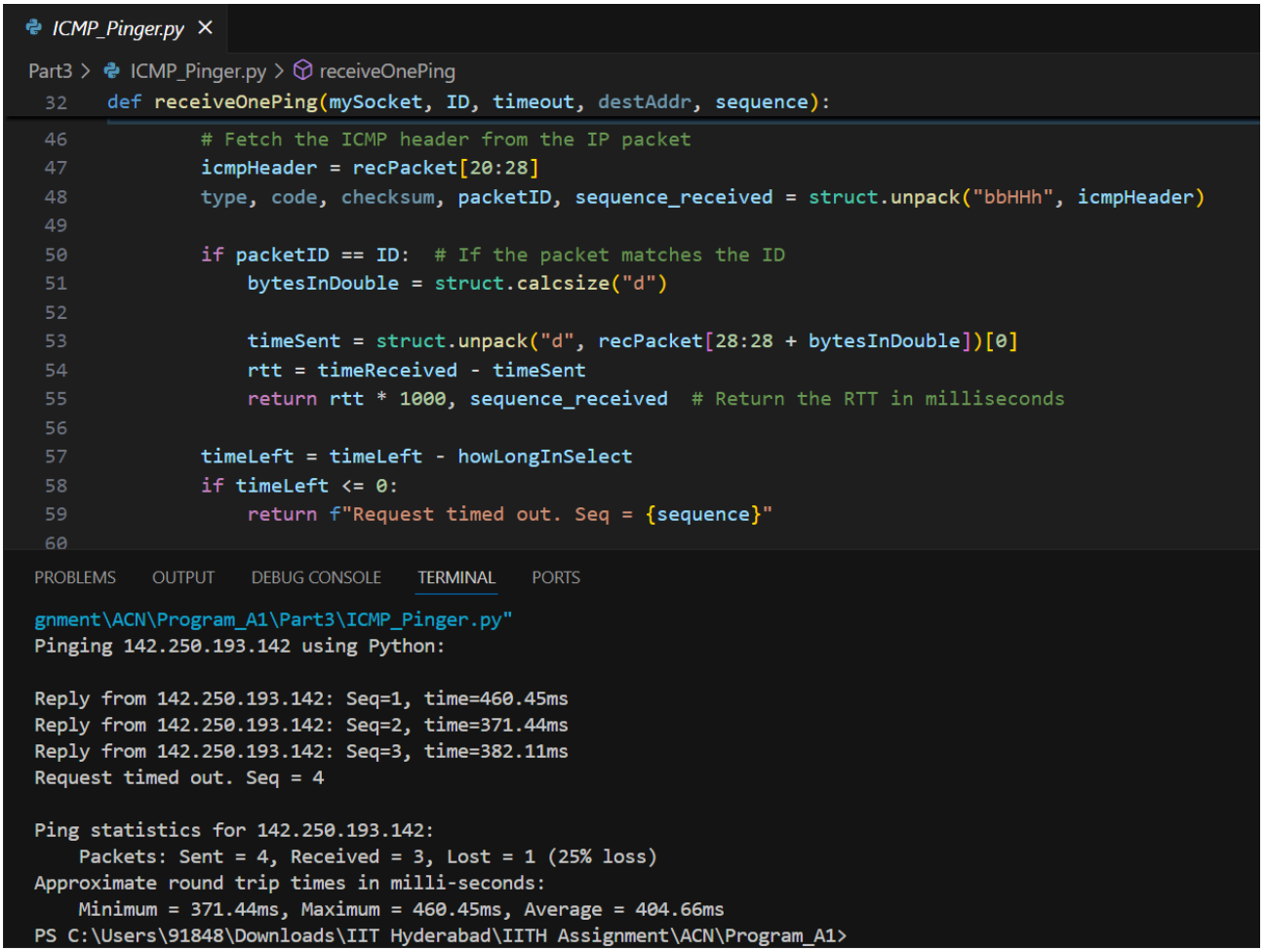
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Sent Ping 7 Thu Sep 12 18:06:43 2024
Received PING 7 THU SEP 12 18:06:43 2024


Sent Ping 8 Thu Sep 12 18:06:43 2024
ICMP Error: Port Unreachable

Sent Ping 9 Thu Sep 12 18:06:44 2024
ICMP Error: Port Unreachable

Sent Ping 10 Thu Sep 12 18:06:45 2024
ICMP Error: Port Unreachable



Ping statistics for 172.21.132.192:
     Packets: Sent = 10, Received = 6, Lost = 4 (40.0% loss)
Approximate round trip times in milli-seconds:
     Minimum: 0.01 ms, Maximum: 0.08 ms, Average: 0.04 ms

rushi@rushi-VirtualBox:~/Desktop/Computer_Networks/My_Pingers_Files/Program_A1$
```

Part 3: ICMP Pinger

1. ICMP to google without packet loss :

Sending ICMP ping requests to the google.com server, and the following responses are received.As observed below there is one packet loss.



2. ICMP Pinger to another machine without packet loss

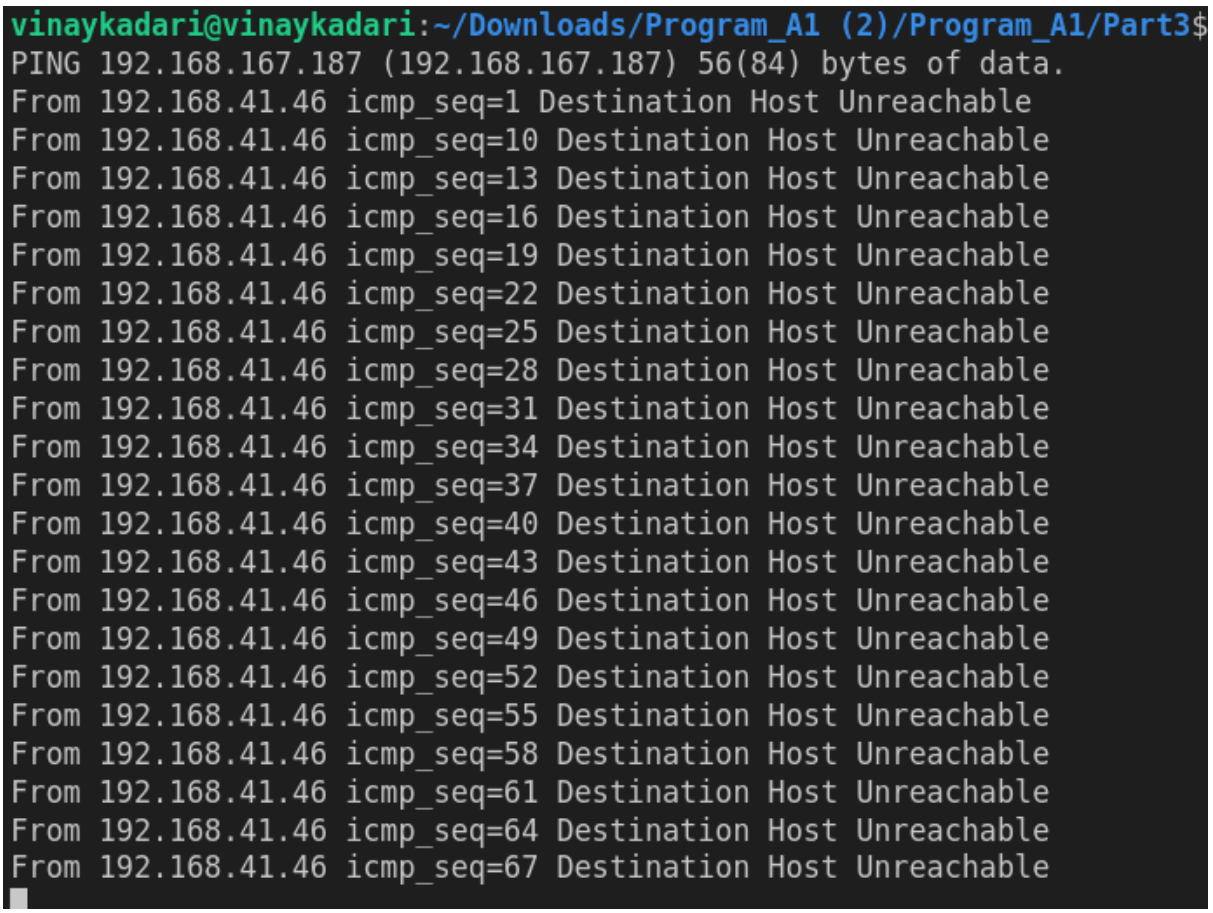
ICMP_Pinger.py

```
def ping(host,numPing, timeout=1):
            avg_rtt = sum(rtts) / len(rtts)
            packet_loss_rate = ((numPing - len(rtts)) / numPing) * 100

            print(f"\nMinimum RTT: {min_rtt:.2f} ms")
            print(f"Maximum RTT: {max_rtt:.2f} ms")
            print(f"Average RTT: {avg_rtt:.2f} ms")
            print(f"Packet loss rate: {packet_loss_rate:.2f}%")
        else:
            print("No RTTs recorded.")
            print("Packet loss rate: 100%")
    numPing=int(input("Enter number of pings :"))
    ping("192.168.167.187",numPing)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\91848\Downloads\IIT Hyderabad\mp> python -u "c:\Users\91848\Downloa
Enter number of pings :5
Pinging 192.168.167.187 using Python:

Reply from 192.168.167.187: time=76.32ms
Reply from 192.168.167.187: time=230.20ms
Reply from 192.168.167.187: time=25.61ms
Reply from 192.168.167.187: time=223.30ms
Reply from 192.168.167.187: time=229.51ms

Minimum RTT: 25.61 ms
Maximum RTT: 230.20 ms
Average RTT: 156.99 ms
Packet loss rate: 0.00%
PS C:\Users\91848\Downloads\IIT Hyderabad\mp>
```

3. ICMP with handling error



Work Distribution Summary

Name: Vinaykumar Kadari

Roll No: CS24MTECH14008

Task/Section	Contribution	Challenges faced
Research & Info gathering	My contribution is on TCP Pinger, ICMP Pinger and Error Handling Reference: a) Multithreaded TCP servers and clients,Multiple Clients Model Multithreading b) Modified TCP without tc-netem NIC level by using tc (traffic control) netem	Understanding the structure of ICMP packet and TCP pinger packet and improve it and customize the code takes traffic of time
Code Development	TCP client & TCP server	It is crucial to manage exception and error in socket programming

	TCP_icmp_server TCP_icmp_client ICMP_Pinger	to avoid crashes and work smoothly
Testing & Debugging	TCP client & TCP server TCP_icmp_server TCP_icmp_client ICMP_Pinger	While testing the server gets unresponsive sometime at that time the client might hang or receive incomplete packets.
Documentation & Report Writing	Formatting the documentations and correctly keeping screenshots	
Final Review & Submission	Ensure the code, report and readme file are correctly formatted	

Name: P.Rushi Keswar Reddy

Roll No: CS24MTECH11018

Task/Section	**Contribution**	**Challenges faced**
Research & Info gathering	My contribution is on ICMP_Pinger.py, TCPPingerThreading.py and UDP_icmp_client and server Reference: TCP Socket Programming ICMP Pinger	In Pinger, ICMP is different from UDP and TCP sockets, and accessing raw sockets might require administrative privileges.
Code Development	TCPPingerThreading.py UDP_icmp_client UDP_icmp_server ICMP_Pinger	It is crucial to manage exception and error in socket programming to avoid crashes and work smoothly
Testing & Debugging	TCPPingerThreading.py UDP_icmp_client UDP_icmp_server ICMP_Pinger	During testing, if the server becomes unresponsive, the client may hang or receive incomplete packets.
Documentation & Report Writing	Explaining the screenshots and correcting the sentences	
Final Review & Submission	Review and test the code thoroughly and check whether the code correctly	

	works or not	

Name: KISHOR KUMAR PATRO

Roll No: SM24MTECH14001

Task/Section	**Contribution <describe the work done, in brief>**	**Challenges faced (if any).**
Research & Info gathering	UDPPingerServer.py	Understanding UDP Lack of reliability mechanisms, UDP doesn't have built-in reliability features like acknowledgments and retransmissions.UDP has more speed compare to tcp
Code Development	UDP client UDP server UDP_icmp_server	As UDP doesn’t provide mechanism for re-transmission there is a packet loss over congested network, unlike tcp it lacks of feedback
Testing & Debugging	UDP client UDP server UDP_icmp_server	While testing the udp code there is a request timeout error as iith wifi blocks udp icmp packets.Distinguishing host vs network issues: It's difficult to tell if packet loss is due to end-host failure or genuine network problems.
Documentation & Report Writing		
Final Review & Submission		

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 others if we become aware of them.

Names & Roll No:

Vinaykumar Kadari (CS24MTECH14008)

P.Rushi Keswar Reddy (CS24MTECH11018)

Kishor Kumar Patro (SM24MTECH14001)

Date: 12-09-2024

Signatures: Vinaykumar, Rushikeswar, Kishor