Prg - Asg1 - Report - CS23MTECH11026

Some convection commonly used in all code

- Using Bob1 container as a Server and Alice1 container as a Client for VM's. In the case of Concurrent Server the Alice1 Container has 2 instances.

- Implementing the Packet loss physically at NIC by the command
  " sudo tc qdisc add dev eth0 root netem loss 33% "

- The Output is in the following format:
  PING seq-num start-timestamp – end-timestamp RTT

- Whenever we are running our client-server application in the VM's container, first of all change the IP in the server and client code by the IP of bob1 (which is used as server) then run the respective files.

PART-1: UDP Pinger

1. Implementing the UDPPingerClient.py for UDPPingerServer.py
   Below is the code for the UDP Server (with Artificial loss) and UDP Client respectively.

UDP Server code

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

# Create a UDP socket
# AF_INET for Address Family - IPv4 Network, SOCK_DGRAM for UDP packets
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# Assign IP address and port number to socket
serverSocket.bind(('127.0.0.1', 7000))

while True:
   # Generate a random number between 0 to 11 (both included)
   rand = random.randint(0, 11)
   # Receive the client packet along with the address it is coming from
   message, address = serverSocket.recvfrom(1024)

   # Breaking out form the loop if no message received
   if not message:
      break

   # Capitalize the message from the client
   message = message.upper()

   # Creating Artificial Packet loss
   if rand < 4:
     continue

   # Otherwise, the server responds
   serverSocket.sendto(message, address)

serverSocket.close()
```

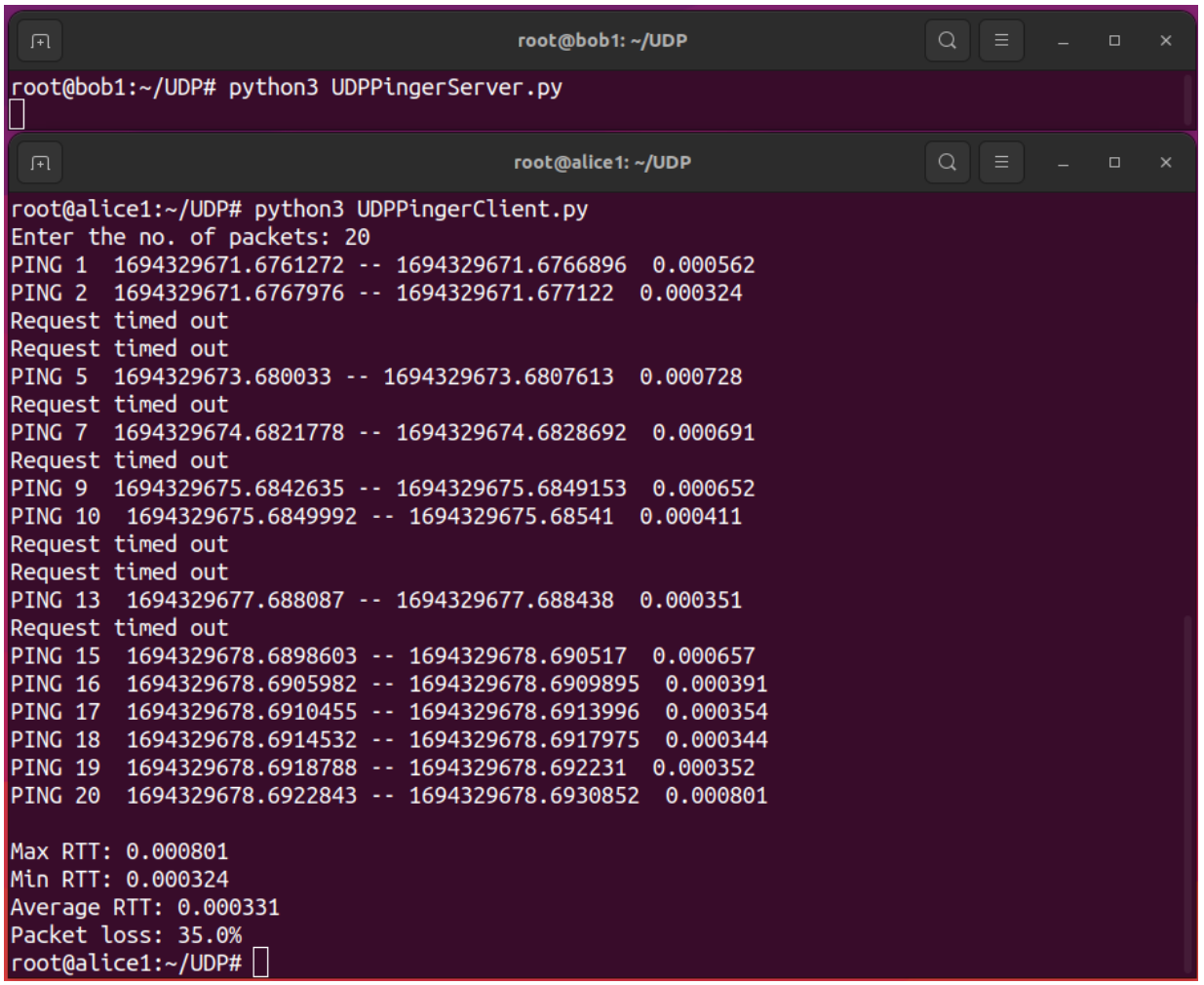
UDP Client code

```
# UDPPingerClient.py
# Importing the required libraries. Time for timestamping and sys for min max
values
import socket
import time
import sys

# Create a UDP socket
# AF_INET for Address Family - IPv4 Network, SOCK_DGRAM for UDP packets
clientSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Creating packet which will sent for N(user defined) times
message = "ping"
N = int(input("Enter the no. of packets: "))

# Encoding the message in bytes
message = message.encode()

# setting timeout for client so that it doesn't wait for infinite time
clientSocket.settimeout(1)

# Variables for Statistics for RTT
maxRTT = sys.maxsize*-1
minRTT = sys.maxsize
avgRTT = 0
pktloss = 0

for i in range(N):
   try:
       # Creating the timestamp before sending & after receiving the packet using
time library
       stime = time.time()
       # Sending message to server with it's address and port
       clientSocket.sendto(message, ('127.0.0.1', 7000))
       # Receiving the client packet along with the address it is coming from
       response, address = clientSocket.recvfrom(1024)
       etime = time.time()

       # Calculation of RTT and updating minRTT & maxRTT
       diff = etime - stime
       maxRTT = max(maxRTT, diff)
       minRTT = min(minRTT, diff)

       # Used for Calculation of AvgRTT at the end
```

```
        avgRTT += diff

        # First decoding the message received then printing
        response = response.decode()
        print(f'{response} {i+1}  {stime} -- {etime}  {round((diff), 6)}')

    except TimeoutError:
        # Handling the Timeout Error
        print("Request timed out")
        pktloss += 1
        continue

# Finally after N packets are sent & receive the client Socket is closed
clientSocket.close()

# Printing the RTT Statistics
print(f"\nMax RTT: {round((maxRTT), 6)}")
print(f"Min RTT: {round((minRTT), 6)}")
print(f"Average RTT: {round((avgRTT/N), 6)}")
print(f"Packet loss: {round(((pktloss/N)*100), 2)}%")
```

- Running the UDPPingerServer.py first and then UDPPingerClient.py in the different containers of the VM we get the following output.

UDPPingerServer.py UDPPingerClient.py with Artificial Packet Loss in VM's container



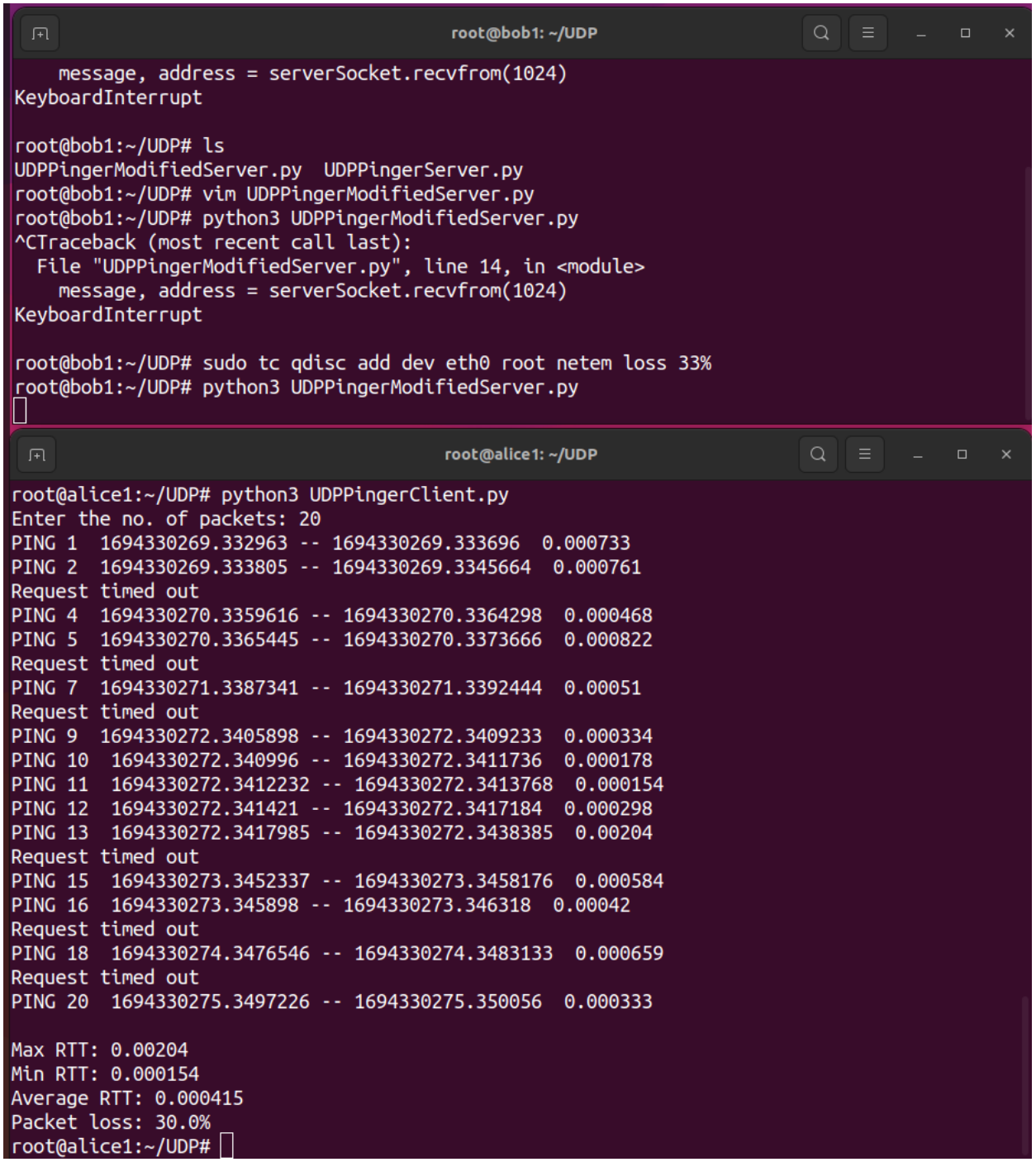
2. Now Removing the Artificial loss (which was done using a random number) and implement the loss at the hardware level (physically to NIC)

UDP Modified Server code

```
# UDPPingerModifiedServer.py
# Here we are going to generate packet loss at NIC
import socket

# Create a UDP socket
# AF_INET for Address Family - IPv4 Network, SOCK_DGRAM for UDP packets
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# Assign IP address and port number to socket
serverSocket.bind(('127.0.0.1', 7000))

while True:
   # Receive the client packet along with the address it is coming from
   message, address = serverSocket.recvfrom(1024)

   # Breaking out form the loop if no message received
   if not message:
      break

   # Capitalize the message from the client
   message = message.upper()

   # Sending the Capitalize message to the client
   serverSocket.sendto(message, address)

serverSocket.close()
```

- The code for the UDPPingerClient.py remains the same. Just before running client code we have to inject packet loss at NIC.
- The Packet loss which is not exactly 33% because so many factors comes into consideration like it loss 33% of all the packet which are coming not just in our client-server application. In the output of VM I got 30% packet loss

UDPPingerModifiedServer.py UDPPingerClient.py with NIC 33% packet loss in VM's container.



PART-2: TCP Pinger

1. Creating the client-server application using TCP protocol. Here we will implement the Artificial loss by using the random number. As we are using the TCP before transmitting first we have to establish a connection with the client. In TCP protocol no packet is lost but there would be retransmissions. The code for TCP Server and TCP Client are as below

TCP Server code

```
# TCPPingerServer.py
# We will need the following libraries to generate randomized lost packets
import socket
import random

# Create a TCP socket
# AF_INET for Address Family - IPv4 Network, SOCK_STREAM for TCP packets
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Assign IP address and port number to socket
serverSocket.bind(('127.0.0.1', 9003))

# Server listening for incoming connection
serverSocket.listen()
# Server accepts the incoming connection along with address it is coming form
clientSocket, address = serverSocket.accept()
print(f"Connection with {address} has been established")

try:
   while True:
       # Receiving client message
       msg = clientSocket.recv(1024)
       rand = random.randint(1, 11)

       # If no message is there then breaking out form loop
       if not msg:
           break

       # Decoding and Capitalize the message from the client
       msg = msg.decode().upper()

       # Performing the Artificial Loss
       if rand < 4:
           continue
```

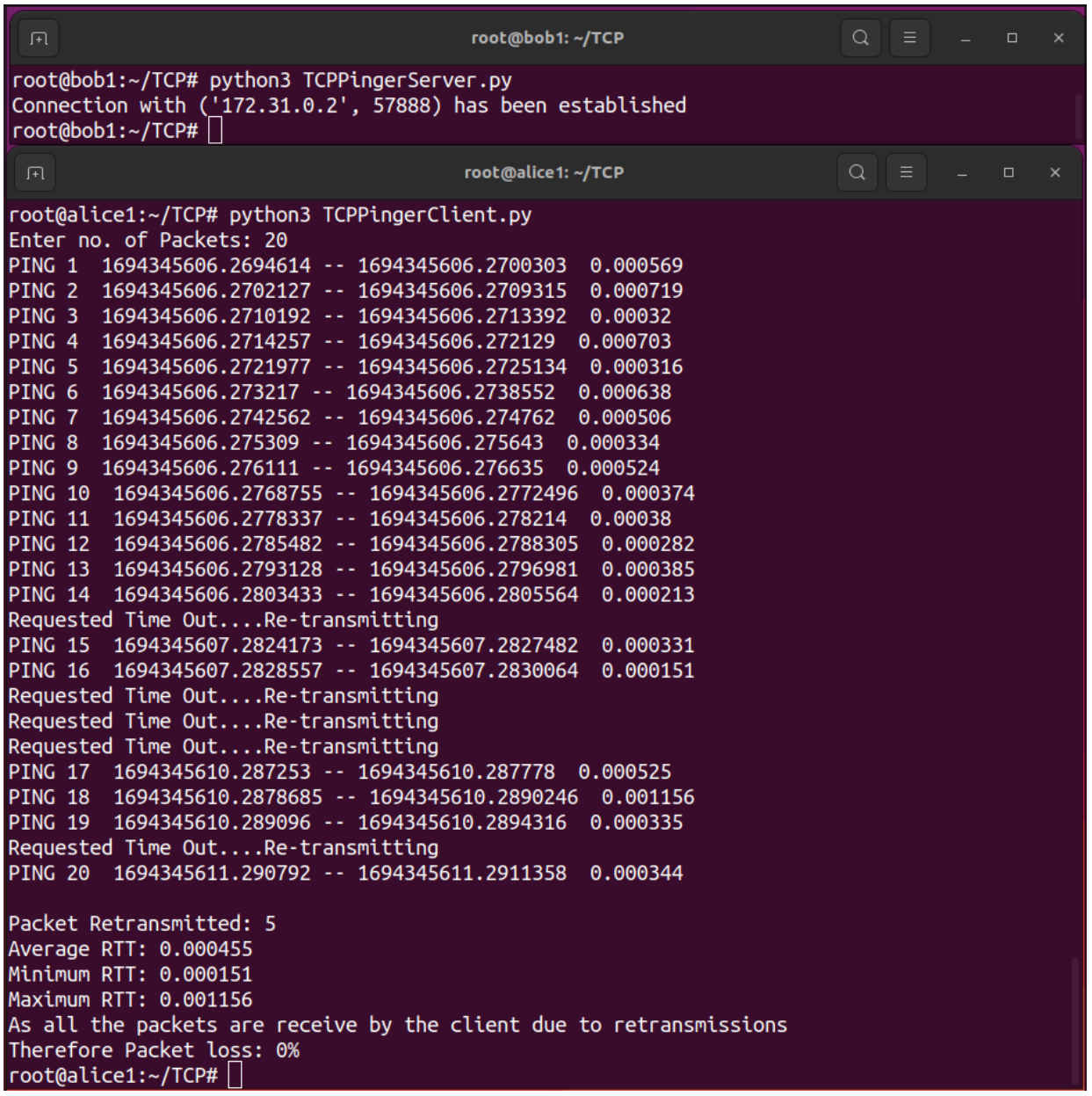
```
        # Otherwise, the server responds
        clientSocket.send(msg.encode())

finally:
    serverSocket.close()
```

TCP Client code

```
# TCPPingerClient.py
# Libraries required for socket & calculatinge RTT and it's Statistics
import socket
import time
import sys

# Create a TCP socket
# AF_INET for Address Family - IPv4 Network, SOCK_STREAM for TCP packets
clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# As the server is listening the client request's for connection
clientSocket.connect(("127.0.0.1", 9003))

# Client will wait for 1sec if no response comes form server
clientSocket.settimeout(1)

# Getting input from user to send N packets
N = int(input("Enter no. of Packets: "))

# Some variables for calculation of RTT and it's Statistics
pktRetrans = 0
maxRTT = sys.maxsize*(-1)
minRTT = sys.maxsize
avgRTT = 0

for i in range(N):
    msg = f"ping {i+1}"

    # Start timestamp just before sending the packet
    stime = time.time()
    clientSocket.send(msg.encode())

    while True:
        try:
            # Response form server and ending timestamp
            response = clientSocket.recv(1024)
```

```
            etime = time.time()

            # Calculating RTT and updating min max RTT
            diff = etime - stime
            maxRTT = max(maxRTT, diff)
            minRTT = min(minRTT, diff)

            # Summing all the RTT for avgerage at the end
            avgRTT += diff

            # Decoding and printing the response and it's RTT
            pkt = response.decode()
            print(f"{pkt}  {stime} -- {etime}  {round(diff, 6)}")
            break

        except socket.error as e:
            # Printing the Time out error and incrementing the packet retransmited
variable
            print(f"Requested Time Out....Re-transmitting")
            pktRetrans += 1

            # Again giving the new start time as we are retransmitting
            stime = time.time()
            clientSocket.send(msg.encode())

clientSocket.close()

TotalpktTrans = pktRetrans + N
pktloss = pktRetrans / TotalpktTrans

# printing the RTT and it's Statistics
print(f"\nPacket Retransmitted: {(pktRetrans)}")
print(f"Average RTT: {round((avgRTT/N), 6)}")
print(f"Minimum RTT: {round(minRTT, 6)}")
print(f"Maximum RTT: {round(maxRTT, 6)}")
print(f"Packet loss: {round(pktloss*100, 2)}%")
```

TCPPingerServer.py TCPPingerClient.py Artificial Packet loss in VM's container:



2. Now without Artificial loss we have created TCPPingerModifiedServer.py file so that we can implement loss physically at NIC.Therefore first checking without NIC loss.

TCP Modified Server code

```
# TCPPingerModifiedServer.py
import socket

# Create a TCP socket
# AF_INET for Address Family - IPv4 Network, SOCK_STREAM for TCP packets
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Assign IP address and port number to socket
serverSocket.bind(('127.0.0.1', 9003))

# Server listening for incoming connection
serverSocket.listen()
# Server accepts the incoming connection along with address it is coming form
clientSocket, address = serverSocket.accept()
print(f"Connection with {address} has been established")

try:
   while True:
       # Receiving client message
       msg = clientSocket.recv(1024)

       # If no message is there then breaking out form loop
       if not msg:
           break

       # Decoding and Capitalize the message from the client
       msg = msg.decode().upper()

       # Sending the Capitalize message to the client
       clientSocket.send(msg.encode())

finally:
   serverSocket.close()
```

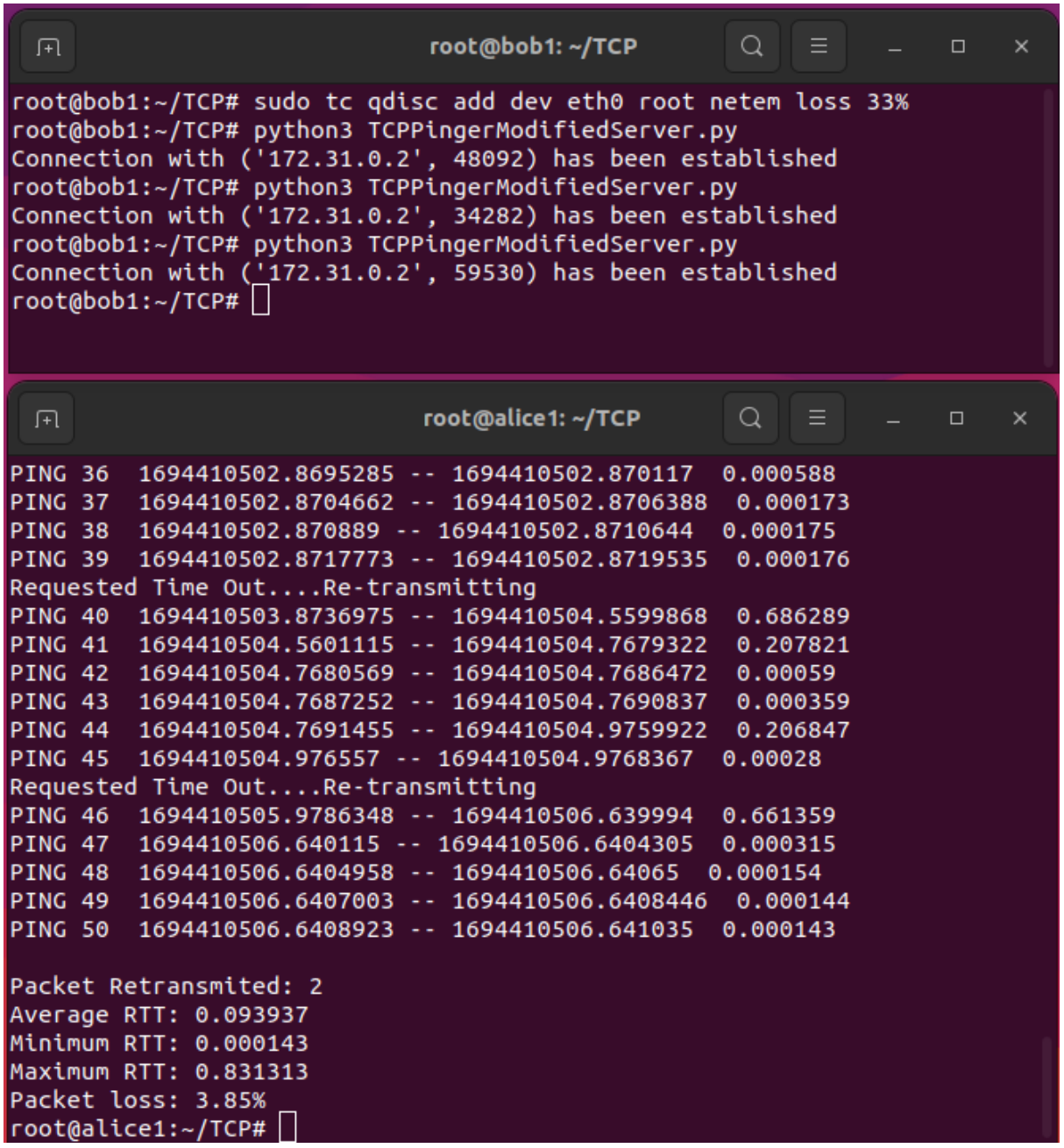
TCP Modified Client code

```
# TCPPingerModifiedClient.py
# Libraries required for socket & calculatinge RTT and it's Statistics
import socket
import time
import sys

# Create a TCP socket
# AF_INET for Address Family - IPv4 Network, SOCK_STREAM for TCP packets
clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# As the server is listening the client request's for connection
clientSocket.connect(("127.0.0.1", 9003))

# Client will wait for 1sec if no response comes form server
clientSocket.settimeout(1)

# Getting input from user to send N packets
N = int(input("Enter no. of Packets: "))

# Some variables for calculation of RTT and it's Statistics
pktRetrans = 0
minRTT = sys.maxsize
maxRTT = sys.maxsize*(-1)
avgRTT = 0

for i in range(N):
   msg = f"ping {i+1}"

   # Start timestamp just before sending the packet
   stime = time.time()
   clientSocket.send(msg.encode())
   while True:
       try:
           # Response form server and ending timestamp
           response = clientSocket.recv(1024)
           etime = time.time()

           # Calculating RTT and updating min max RTT
           diff = etime - stime
           maxRTT = max(maxRTT, diff)
           minRTT = min(minRTT, diff)

           # Summing all the RTT for avgerage at the end
           avgRTT += diff
```

```
            # Decoding and printing the response and it's RTT
            response = response.decode()
            print(f"{response}  {stime} -- {etime}  {round(diff, 6)}")
            break

        except socket.error as e:
            # Printing the Time out error and incrementing the packet retransmited
variable
            print(f"Requested Time Out....Re-transmitting")
            pktRetrans += 1

            # As packet is not received we again start the timer
            # Still the while loop is running for the same packet then finally we
get response
            stime = time.time()

clientSocket.close()

TotalpktTrans = pktRetrans + N
pktloss = pktRetrans / TotalpktTrans

# Printing the RTT and it's Statistics
print(f"\nPacket Retransmited: {(pktRetrans)}")
print(f"Average RTT: {round((avgRTT/N), 6)}")
print(f"Minimum RTT: {round(minRTT, 6)}")
print(f"Maximum RTT: {round(maxRTT, 6)}")
print(f"Packet loss: {round(pktloss*100, 2)}%")
```

- Implementing the NIC loss of 33% by using the same command as used before in UDP.

TCPPingerModifiedServer.py TCPPingerModifiedClient.py with NIC Packet loss of 33% at VM's container



3. In the case of the Concurrent Server, we create a thread for each client who establishes the connection with the concurrent server. Then handling them individually by targeting the client handling function.

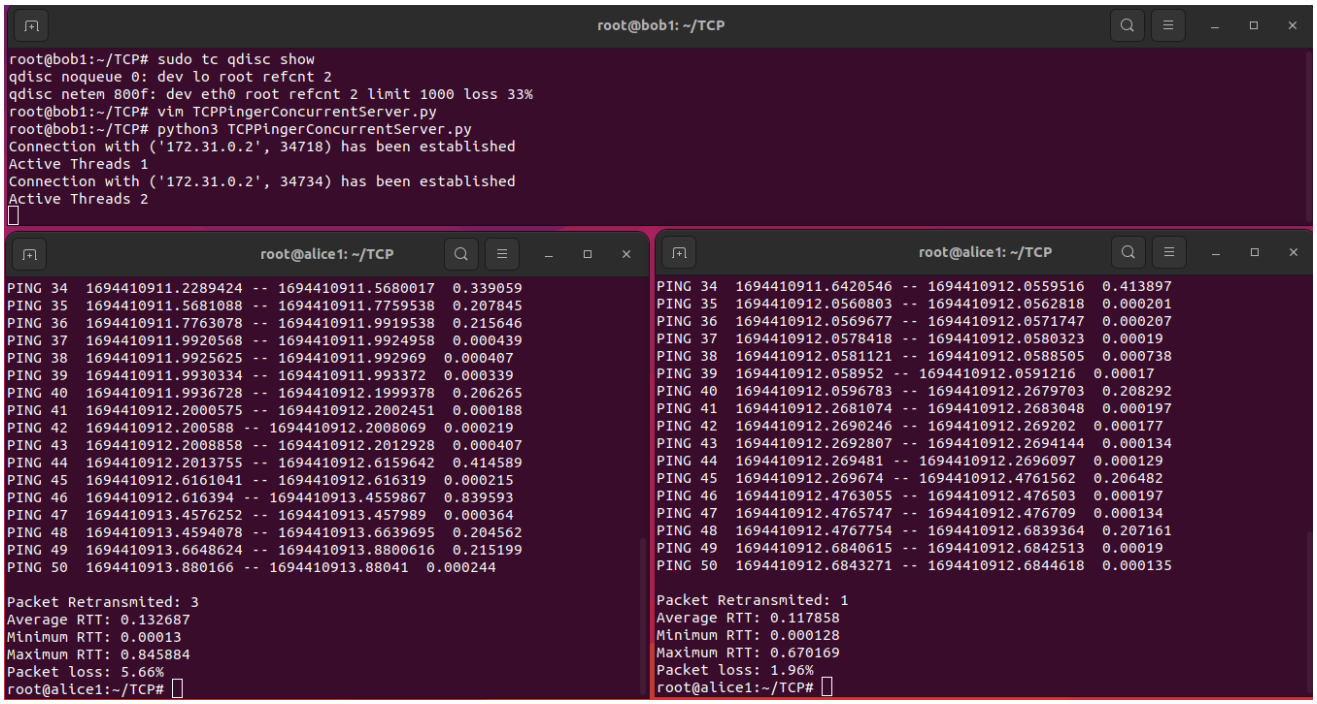
TCP Concurrent Server code

```
# TCPPingerConcurrentServer.py
# To create thread per client we require threading library
import socket
import threading

# Create a TCP socket
# AF_INET for Address Family - IPv4 Network, SOCK_STREAM for TCP packets
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Assign IP address and port number to socket
serverSocket.bind(("127.0.0.1", 9003))

# Server listening for incoming connection
serverSocket.listen()

# Function to handel the each Client thread
def client(clientSocket, address):
    connected = True
    while connected:
        # Receving and decoding the message form client
        message = clientSocket.recv(1024).decode()
        # Capitalize the message received form the client
        message = message.upper()

        # condition to break out form while loop
        if not message:
            connected = False

        # Sending back the modified message
        clientSocket.send(message.encode())

    # Closing the socket
    clientSocket.close()

try:
    while True:
        # for every client we have to accept the connection request
        clientSocket, address = serverSocket.accept()
        print(f"Connection with {address} has been established")
```

```
        # After accepting we create Thread for each client
        thread = threading.Thread(target=client, args=(clientSocket, address))
        thread.start()

        # Printing the No. of Threads that are acttive
        print(f"Active Threads {threading.active_count() - 1}")

except socket.error as e:
    print(e)

finally:
    serverSocket.close()
```

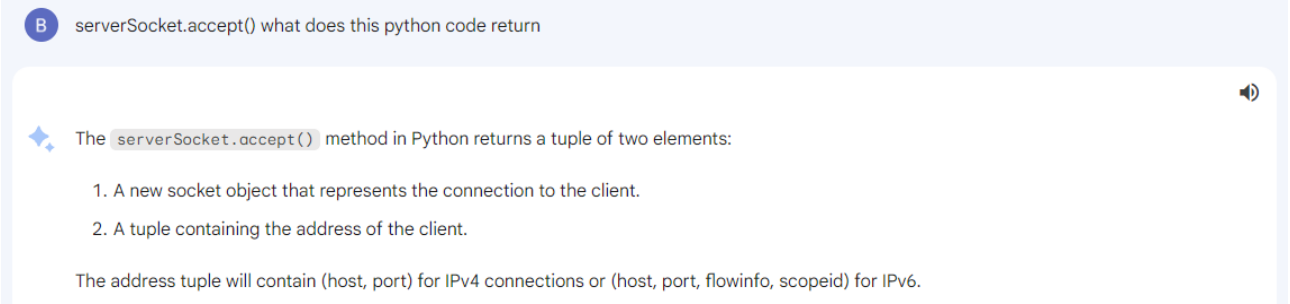
- The Client code would be the same as the TCP Modified Client.
- After giving the command to the terminal for having packet loss of 33% at NIC we get the output as below. To check for the concurrent we have two instances of alice1 (client) and both requesting bob1 (Concurrent server).

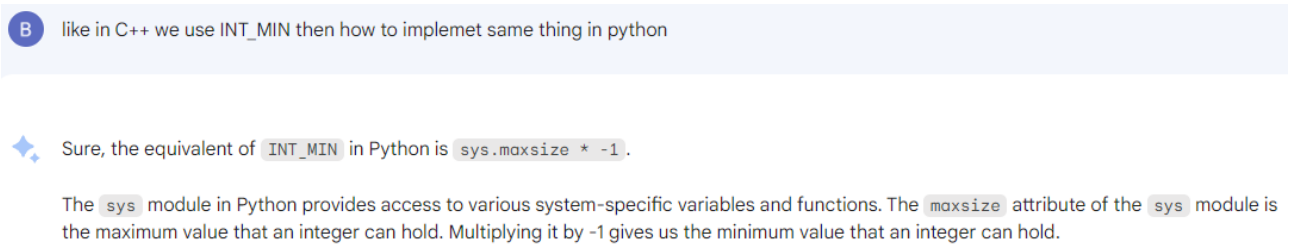
TCPPingerConcurrentServer.py TCPPingerModifiedClient.py with NIC packet loss at VM's container:

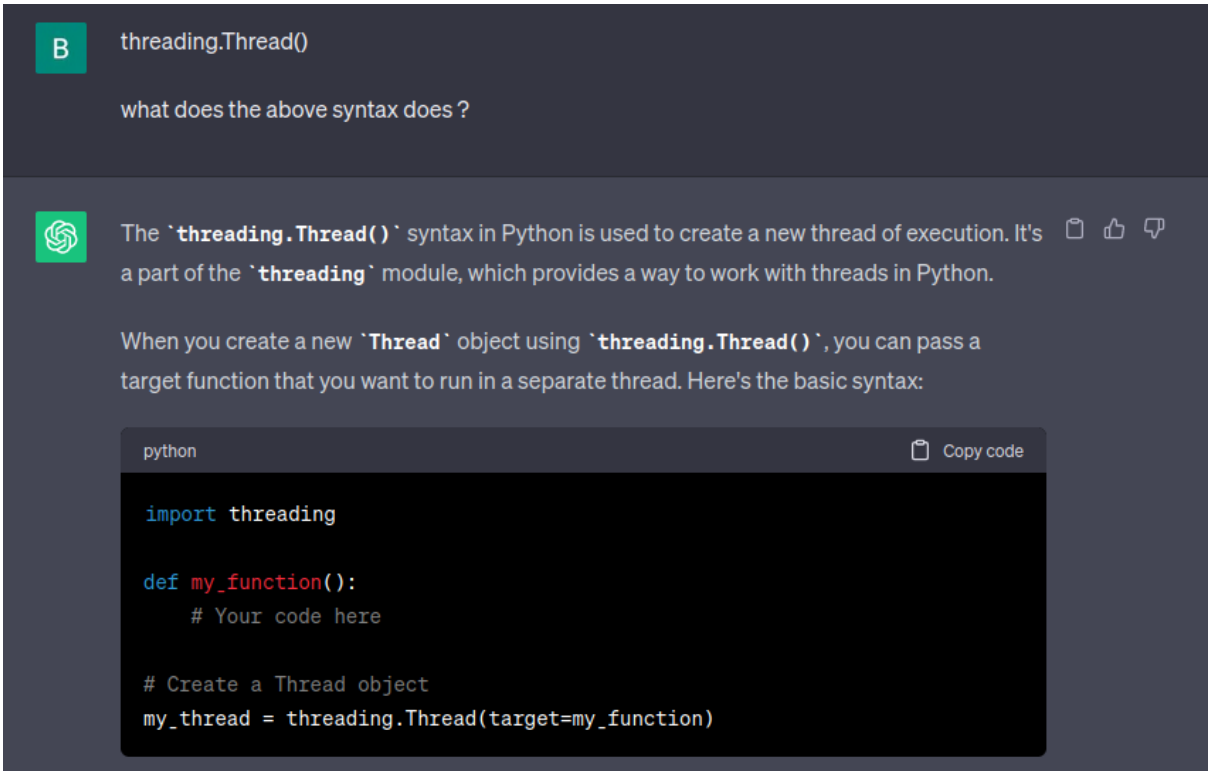


Interactions with LLM's:

Taking some reference form LLM and modifying the code for my application, also used for understanding some syntax, what they return, why it is used for etc.







B

what does the time.time.now() returns

In Python, `time.time()` and `time.time.now()` are two different functions/methods, and they return different types of objects:

1. `time.time()`:
   `time.time()` is a function from the `time` module that returns the current time as a floating-point number representing the number of seconds since the epoch (January 1, 1970). It returns a single value, a float.
   Example:

```python
import time

current_time = time.time()
print(current_time)
```

2. `datetime.datetime.now()`:
   `datetime.datetime.now()` is a method from the `datetime` module that returns the current date and time as a `datetime.datetime` object. It provides a more detailed representation of the current date and time, including year, month, day, hour, minute, second, and microsecond.
   Example:

```python
import datetime

current_datetime = datetime.datetime.now()
print(current_datetime)
```

B

how to create client socket and send message to the server in python

To create a client socket and send a message to a server in Python, you can use the `socket` module. Here's a step-by-step example:

1. Import the `socket` module.
2. Create a socket object using `socket.socket()`.
3. Specify the server's IP address and port number.
4. Encode the message into bytes (if it's a string).
5. Use the `send` method to send the message to the server.
6. Close the socket when done.

Here's a basic example:

python Copy code

```python
import socket

# Define the server's IP address and port
server_ip = "127.0.0.1"  # Replace with the actual server IP
server_port = 5050

# Create a socket object
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect to the server
client_socket.connect((server_ip, server_port))

# Define the message you want to send
message = "Hello, server!"

# Encode the message into bytes (if it's a string)
message_bytes = message.encode('utf-8')

# Send the message to the server
client_socket.send(message_bytes)

# Close the socket when done
client_socket.close()
```

ANTI-PLAGIARISM Statement

I certify that this assignment/report is my own work, based on my personal study or research, and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, packages, datasets, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment/report has not previously been submitted for assessment/project in any other course lab, except where specific permission has been granted from all course instructors involved or at any other time in this course, and that I have not copied in part or whole or otherwise plagiarized the work of other students and/or persons. Additionally, I acknowledge that I may have used AI tools, such as language models (e.g., ChatGPT, Bard), for assistance in generating and refining my assignment, and I have made all reasonable efforts to ensure that such usage complies with the academic integrity policies set for the course. I pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, I understand my responsibility to report honour violations by other students if I become aware of it.

Name <Roll No.>: Bhargav Piyushkumar Patel <CS23MTECH11026>
Date: 11th September 2023
Signature: B.P.P

References:

1. A Python socket tutorial is http://docs.python.org/howto/sockets.html
2. https://man7.org/linux/man-pages/man8/tc-netem.8.html
3.https://srtlab.github.io/srt-cookbook/how-to-articles/using-netem-to-emulate-networks.html
4. https://www.cs.unm.edu/~crandall/netsfall13/TCtutorial.pdf
5. https://realpython.com/intro-to-python-threading/
6. https://www.tutorialspoint.com/python/python_multithreading.htm
7. https://docs.python.org/3/library/concurrency.html