Programming Assignment-1

TASK – 1: UDP Pinger

This part implements a Ping client-server system using the UDP protocol

UDPPingerServer.py

This Python code is for a UDP server that listens on port 12000. It receives UDP packets, capitalizes the message content, and sends the capitalized message back to the client. The server also simulates packet loss by randomly dropping packets (around 33% of the time). The code uses the socket library and an infinite loop to continuously handle incoming packets.

```
# 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)
# Assign IP address and port number to socket
serverSocket.bind((gethostname(), 12000))

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)
# Capitalize the message from the client
    message = message.upper()
# If rand is less is than 4, we consider the packet lost and do not respond
    if rand < 4:
        continue

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

UDPPingerModifiedServer.py

This Server Code is similar with the packet loss not being simulated at application layer, rather it has been emulated at the NIC interface using the tc netum utility of Linux.

```
# 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)
# Assign IP address and port number to socket
serverSocket.bind((gethostname(), 12000))
```

```
while True:

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

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

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

UDPPingerClient.py

This Python code implements a Ping client side utility using UDP. It prompts the user for the number of packets (N) to send. For each packet, it creates a UDP socket, records the send timestamp, and sends a packet to a server. It waits for a response, calculates the round-trip time (RTT), and prints request/response timestamps. It keeps track of RTT statistics (min, max, sum) and counts packet loss due to timeouts. Finally, it displays the maximum RTT, minimum RTT, average RTT, and packet loss percentage for the entire Ping operation.

```
# We will need the following packets for socket connecgtion, RTT calculation
from socket import *
from time import *
from datetime import *
from select import *
from math import *

# The following variable are needed for some showing the summary of Ping
min=inf
max=-inf
sum=0
count=0
N = int(input("Enter Value of N:"))
diff=N

# Loop with iterates around number of packets the user want to send for Ping
echo
for i in range(N):

    # Client send a word 'ping' to server as part of Probe Packet
    message = "ping "+ str(i+1)
    # Creates a UDP socket
    # Notice the use of SOCK_DGRAM for UDP Packet
    UDPClientSocket = socket(AF_INET,SOCK_DGRAM)
    # Notice the use of below method attaches a upperbound of time on the
Socket operation
    UDPClientSocket.settimeout(1)
    # Used to Mark timestamp at which the packet is sent to the server
    datetime_request = datetime.now()
    request_time = datetime_request.strftime("%H:%M:%S:%f")
    # Sends the Packet
    UDPClientSocket.sendto(str.encode(message), ('172.31.0.3',12000))

    # Exception Handling to wait for the response for one second
    try:
        # is_ready, _, _ = select([UDPClientSocket],[],[],1)
        # if is_ready:
```

```
            # Recieves the packet
            response_message = UDPClientSocket.recvfrom(1024)
            # Used to Mark timestamp at which the packet is recieved by the
client
            datetime_response = datetime.now()
            response_time = datetime_response.strftime("%H:%M:%S:%f")
            RTT = (datetime_response - datetime_request).microseconds/1000
            # Prints the Request Timestamp, Resposne Timestamp and RTT for each
Ping probe packet
            print(response_message[0].decode() +" "+ request_time +" "+
response_time + " " + str(RTT))
            RTT=float(RTT)
            # Calculations for summary of entire Ping Utility
            sum=sum+RTT
            if RTT>max:
                max=RTT
            if RTT<min:
                min=RTT
        # else:
        #     print("Requested Time Out")

    except timeout:

        # Counts the number of packet that got lost
        count+=1
        print("Requested Time Out")

if N != count:
     diff = N-count

# Prints the Summary of Entire Ping Utility i.e. Minimim, Maximum, Average RTT
and Packet Loss
print(f"\nMax:{max}\nMin:{min}\nAverage:{sum/diff}\nPacket
Loss:{((N-diff)/N)*100}%")
```

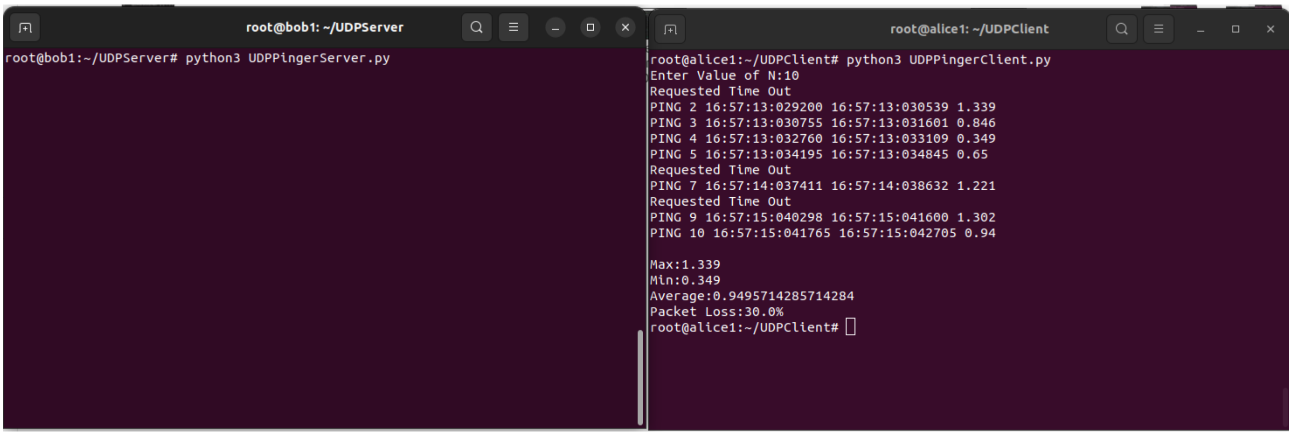
Results:

(1) Simulation of packet loss at application layer :-

Here as we can see server side console and client side console on the left and right side respectively. As clearly observed the entries on the client side have something popping as “Requested Time Out” which actually means that the ping echo packets send from the server didn’t recieved any response from the server within a timeframe of one second. Here an intresting thing to note is that the logic for simulation of packet loss is passive i.e. we are simply generating a random number between 0 and 11 (inclusive) and if the number is less than 4 then we skip that itertaion on the server side making server to wait the client from more than one second and hence the client considers it to be dropped.

Here, in ideal case the probability of packet loss = 4/12 = 33.33%, but that wont be the case as generating random number is uncontrollabel, not in our hand.

In this case the packet loss is 30%.

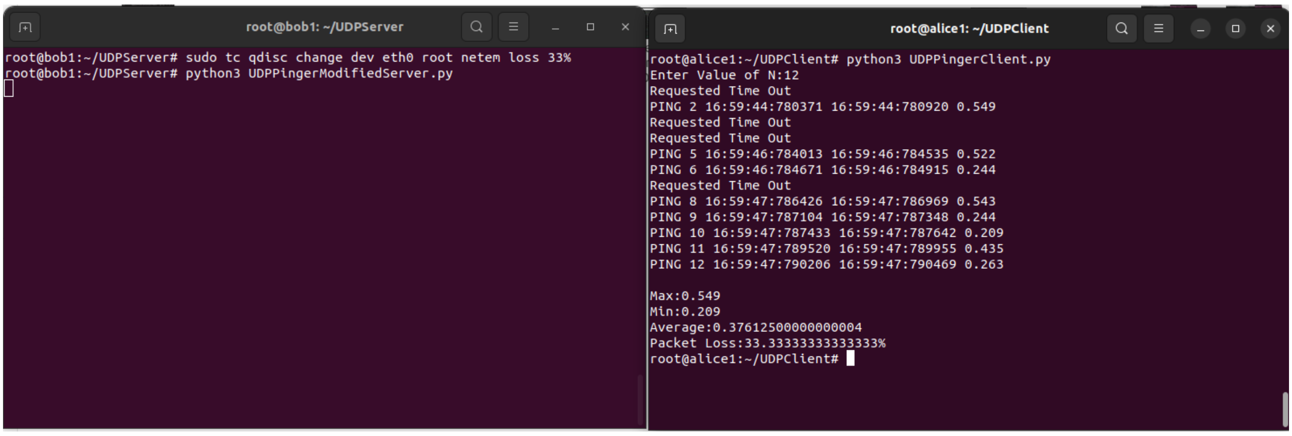


Simulation of packet loss

(2) Emulation of Packet loss at NIC Interface :-

The case is just a better modified version of the above one, where we are actually making the hardware i.e. NIC Card to explicitly have a packet loss of a specific required percentage. This is the most acurate method if one requires a packet loss. The case can be clearly visible as here the packet loss is 33% which is as required.

The command for packet loss on Intefrace is mention below with on the top most line of server side console



Emulation of packet loss

TASK – 2: TCP Pinger

This part implements a Ping client-server system using the TCP protocol

TCPPingerServer.py

This Python code sets up a TCP server that listens for client connections on port 12000. It accepts incoming connections, receives messages from clients, and occasionally simulates packet loss. The code generates a random number, capitalizes received messages, and responds to clients. If the random number is less than 4, it simulates packet loss by not responding. The server also simulates packet loss by randomly dropping packets (around 33% of the time). The server runs in an infinite loop, continuously accepting and handling client connections. It closes the connection with clients after processing each message.

```
# We will need the following module to generate randomized lost packets
import random
from socket import *

# Create a TCP socket
# Notice the use of SOCK_STREAM for TCP packets
serverSocket = socket(AF_INET, SOCK_STREAM)
# Assign IP address and port number to socket
serverSocket.bind((gethostname(), 12000))

# For contineous listening
while True:

    # Servers listens contineously for clients
    serverSocket.listen()
    # Accepts the connection made by clients i.e. makes TCP pipes
    conn, address = serverSocket.accept()

    try:

        while True:

            # Generate a random number between 0 to 11 (both included)
            rand = random.randint(0, 11)

            # Capitalize the message from the client
            data = conn.recv(1024)
            message = data.upper()
            # Breaks the connection with client in case of it doesn't recieves
message from client
            if not message:
                break
            # If rand is less is than 4, we consider the packet lost and do not
respond
            if rand < 4:
                continue
            # Otherwise, the server responds
            conn.sendall(message)

    finally:

        # Breaks the connection of client socket i.e. pipe
        conn.close()
```

TCPPingerModifiedServer.py

This Server Code is similar with the packet loss not being simulated at application layer, rather it has been emulated at the NIC interface using the tc netum utility of Linux.

```
# We will need the following module to generate randomized lost packets
import random
from socket import *

# Create a TCP socket
# Notice the use of SOCK_STREAM for TCP packets
serverSocket = socket(AF_INET, SOCK_STREAM)
# Assign IP address and port number to socket
serverSocket.bind((gethostname(), 12000))

# For contineous listening
while True:

    # Servers listens contineously for clients
    serverSocket.listen()
    # Accepts the connection made by clients i.e. makes TCP pipes
    conn, address = serverSocket.accept()

    try:

        while True:

            # Capitalize the message from the client
            data = conn.recv(1024)
            message = data.upper()
            # Breaks the connection with client in case of it doesn't recieves
message from client
            if not message:
                break
            # Otherwise, the server responds
            conn.sendall(message)

    finally:

        # Breaks the connection of client socket i.e. pipe
        conn.close()
```

TCPPingerConcurrentServer.py

This Python code establishes a multithreaded TCP server to efficiently handle multiple client connections. As usual, it listens on port 12000, accepting client connections and processing their messages. Each client is managed in a separate thread, allowing concurrent communication. And at the end messages are capitalized and echoed, ensuring responsive service for multiple clients. This implementation enhances server performance by leveraging multithreading.

```python
# We will need the following module to generate randomized lost packets
import random
from socket import *
from threading import *

# For Thread Handling
def handle_thread(conn,address):

    try:

        # Loop for handeling a particulat Pipe
        while True:

                # Capitalize the message from the client
                data = conn.recv(1024)
                message = data.upper()
                if not message:
                    break
                # Otherwise, the server responds
                conn.sendall(message)

    finally:

        # Breaks the connection of client socket i.e. pipe
         conn.close()

def main():

    # Create a TCP socket
    # Notice the use of SOCK_STREAM for TCP packets
    serverSocket = socket(AF_INET, SOCK_STREAM)
    # Assign IP address and port number to socket
    serverSocket.bind((gethostname(), 12000))
    # Listens for a client Connection
    serverSocket.listen()

    # For continuity
    while True:

        # Accepts the connection made by client i.e. makes TCP pipe
        conn, address = serverSocket.accept()
        # Creates thread
        thread = Thread(target=handle_thread,args=(conn,address))
        # Starts Thread
        thread.start()

if __name__ == "__main__":
    main()
```

<u>TCPPingerClient.py</u>

This Python code creates a Ping utility client:

1. It imports necessary modules for socket communication, time, and statistics.
2. The user inputs the number of Ping requests (N).

3. The code connects to a server, sends Ping requests, records timestamps, calculates RTT, and prints details for each request.
4. It tracks minimum, maximum, and average RTT, as well as the percentage of lost packets.
5. Finally, it summarizes the Ping utility with these statistics, offering insights into network performance

```
# We will need the following packets for socket connecgtion, RTT calculation
from socket import *
from time import *
from datetime import *
from select import *
from math import *

# The following variable are needed for some showing the summary of Ping
min=inf
max=-inf
sum=0
total_packet=0
lost_packet=0
N = int(input("Enter Value of N:"))

# Exception Handling for the client Socket Connection
try:

    # TCP Packet
    TCPClientSocket = socket(AF_INET,SOCK_STREAM)
    # Attemps for connection with Server, listining at mentioned address and
port number
    TCPClientSocket.connect(('172.31.0.3',12000))
    # Notice the use of below method attaches a upperbound of time on the
Socket operation
    TCPClientSocket.settimeout(1)
    # is_ready, _, _ = select([TCPClientSocket],[],[],1)
    # if is_ready: can also be used for fine control of only recieve socket
operation
    # Loop with iterates around number of packets the user want to send for
Ping echo
    for i in range(N):

        # Message Formation
        message = "ping "+ str(i+1)
        total_packet+=1

        # Loops until the client recieves successful response for the current
packet from the server
        while True:

            # Exception Handling to wait for the response for one second
            try:

                # Used to Mark timestamp at which the packet is sent to the
server
                datetime_request = datetime.now()
                request_time = datetime_request.strftime("%H:%M:%S:%f")
                # Sends the packets explicitly since the logic for packet loss
is embedded forcefully and so would be unknown to both client and server
```

```
                TCPClientSocket.sendall(str.encode(message))
                # Recieves the Packet
                response_message = TCPClientSocket.recv(1024)
                # Used to Mark timestamp at which the packet is recieved by the
client
                datetime_response = datetime.now()
                response_time = datetime_response.strftime("%H:%M:%S:%f")
                RTT = (datetime_response - datetime_request).microseconds/1000
                # Prints the Request Timestamp, Resposne Timestamp and RTT for
each Ping probe packet
                print(response_message.decode() +" "+ request_time +" "+
response_time + " " + str(RTT))
                # Calculations for summary of entire Ping Utility
                RTT=float(RTT)
                sum=sum+RTT
                if RTT>max:
                    max=RTT
                if RTT<min:
                    min=RTT
                break
    # else:
    #     print("Requested Time Out")
            except timeout:

                # Counts the number of packet that got lost along with adding
into the total packets that a client sends
                lost_packet+=1
                total_packet+=1
                print("Requested Time Out...Re-transmitting")

except error:

    print(error.with_traceback)

finally:

    # Closes the Client Socket
    TCPClientSocket.close()

# Prints the Summary of Entire Ping Utility i.e. Minimim, Maximum, Average RTT
and Packet Loss
print(f"\nMax:{max}\nMin:{min}\nAverage:{sum/(total_packet-lost_packet)}\nPacket
Loss:{(lost_packet/total_packet)*100}%")
```

<u>TCPPingerModifiedClient.py</u>

This Python code establishes a multithreaded TCP server:

1. It imports necessary modules, including random, socket and thread
2. Defines a `handle_thread` function to manage individual client connections. It capitalizes received messages and responds within separate threads.
3. The `main` function sets up a TCP server on port 12000, continuously listens for client connections, and creates a new thread for each client.

4. Upon client connection, a new thread is created to handle communication concurrently, enabling multiple clients to interact simultaneously.
5. The server capitalizes and echoes client messages while maintaining efficient concurrent communication.

```
# We will need the following module to generate randomized lost packets
import random
from socket import *
from threading import *

# For Thread Handling
def handle_thread(conn,address):

    try:

        # Loop for handeling a particulat Pipe
        while True:

                # Capitalize the message from the client
                data = conn.recv(1024)
                message = data.upper()
                if not message:
                    break
                # Otherwise, the server responds
                conn.sendall(message)

    finally:

        # Breaks the connection of client socket i.e. pipe
         conn.close()

def main():

    # Create a TCP socket
    # Notice the use of SOCK_STREAM for TCP packets
    serverSocket = socket(AF_INET, SOCK_STREAM)
    # Assign IP address and port number to socket
    serverSocket.bind((gethostname(), 12000))
    # Listens for a client Connection
    serverSocket.listen()

    # For continuity
    while True:

        # Accepts the connection made by client i.e. makes TCP pipe
        conn, address = serverSocket.accept()
        # Creates thread
        thread = Thread(target=handle_thread,args=(conn,address))
        # Starts Thread
        thread.start()

if __name__ == "__main__":
    main()
```

Results:

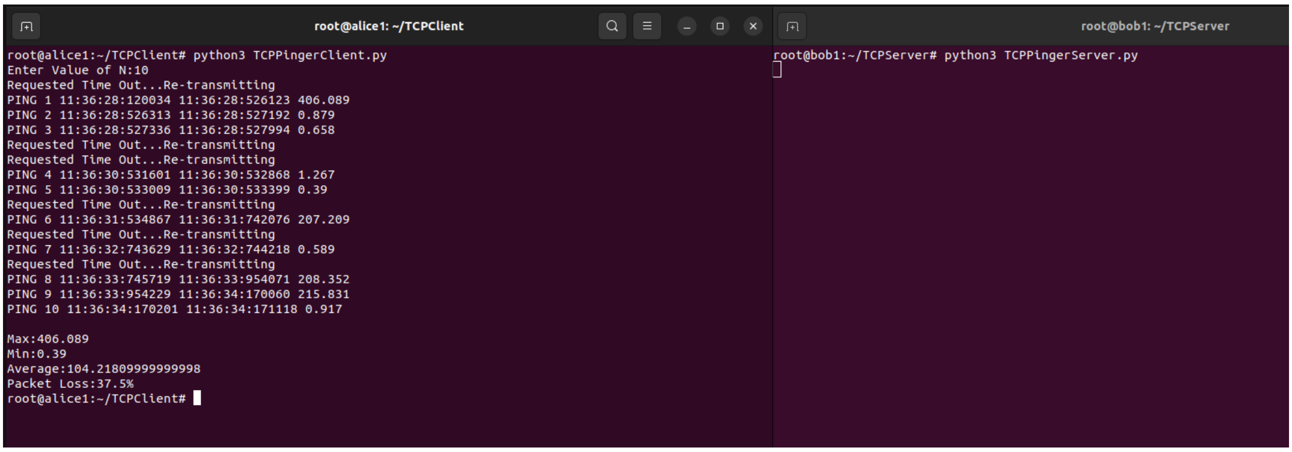
(1) Simulation of packet loss at application layer :-

Here as we can see server side console and client side console on the left and right side respectively. As clearly observed the entries on the client side have something popping as “Requested Time Out” which actually means that the ping echo packets send from the server didn’t recieved any response from the server within a timeframe of one second. Here an intresting thing to note is that the logic for simulation of packet loss is passive i.e. we are simply generating a random number between 0 and 11 (inclusive) and if the number is less than 4 then we skip that itertaion on the server side making server to wait the client from more than one second and hence the client considers it to be dropped.

Here, in ideal case the probability of packet loss = 4/12 = 33.33%, but that wont be the case as generating random number is uncontrollabel, not in our hand.

In this case the packet loss is 37.5%.

The only difference here from ping with UDP protocol is that becasue TCP guarentees reliable transport so even after packet loss the server reattemps to send the same lost packet until it successfully reaches client. But the crux is that as we are implementing a logical loss so have to make client resend the packet incase it gets lost. Otherwise the client thinks that it has sent and the server logically skips that packet and in the similar fashion the client hops that it will get resposne from server and that leads to deadlock.



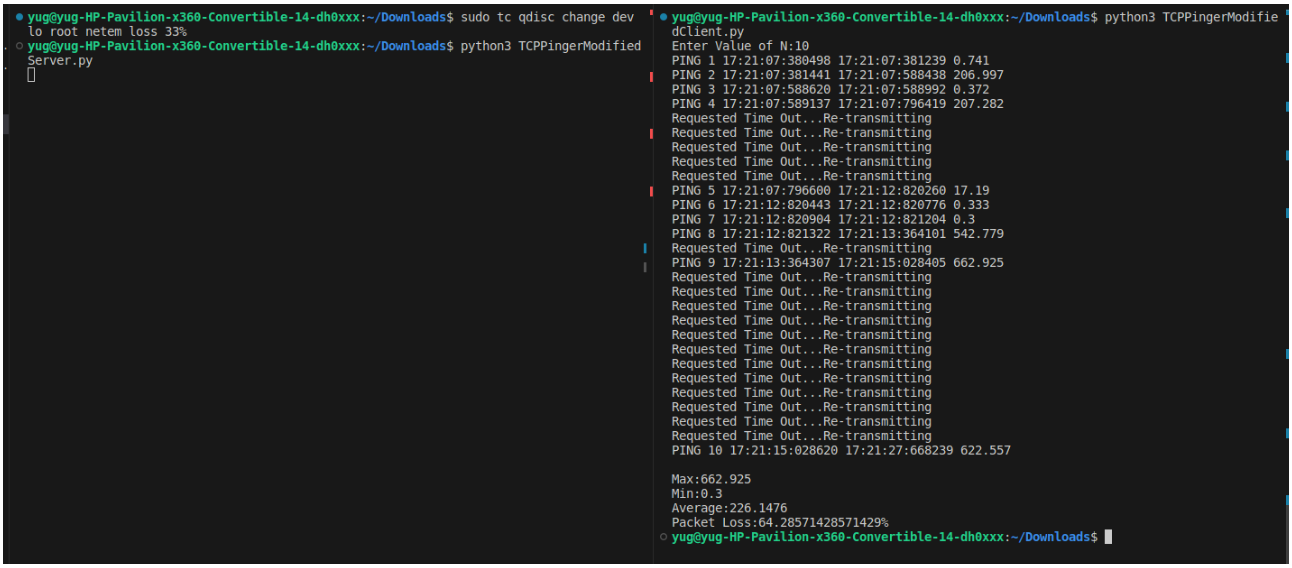
Simulation of packet loss

(2) Emulation of Packet loss at NIC Interface :-

The case is just a better modified version of the above one, where we are actually making the hardware i.e. NIC Card to explicitly have a packet loss of a specific required percentage. This is the most acurate method if one requires a packet loss. The case can be clearly visible as here the packet loss is 33% which is as required.

The command for packet loss on Intefrace is mention below with on the top most line of server side console.

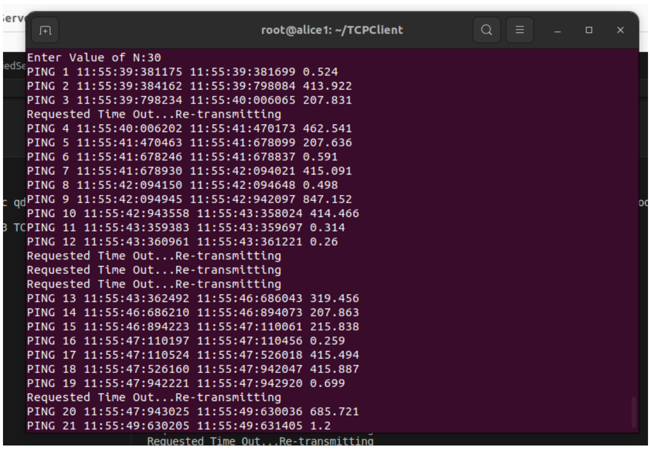
The only difference here from ping with UDP protocol is that becasue TCP guarentees reliable transport so even after packet loss the server reattemps to send the same lost packet until it successfully reaches client. Here there is no need to explicitly resent in case of loss as we have implemented the loss at hardware level due to which the server knows that there is loss and will surely resent the lost packet automitically.



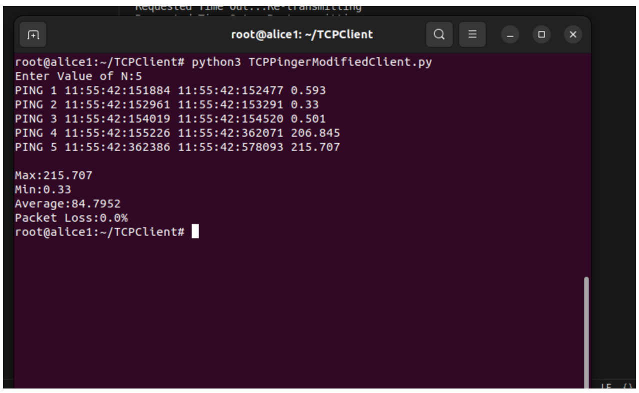
Emulation of packet loss

(3) Concurrency

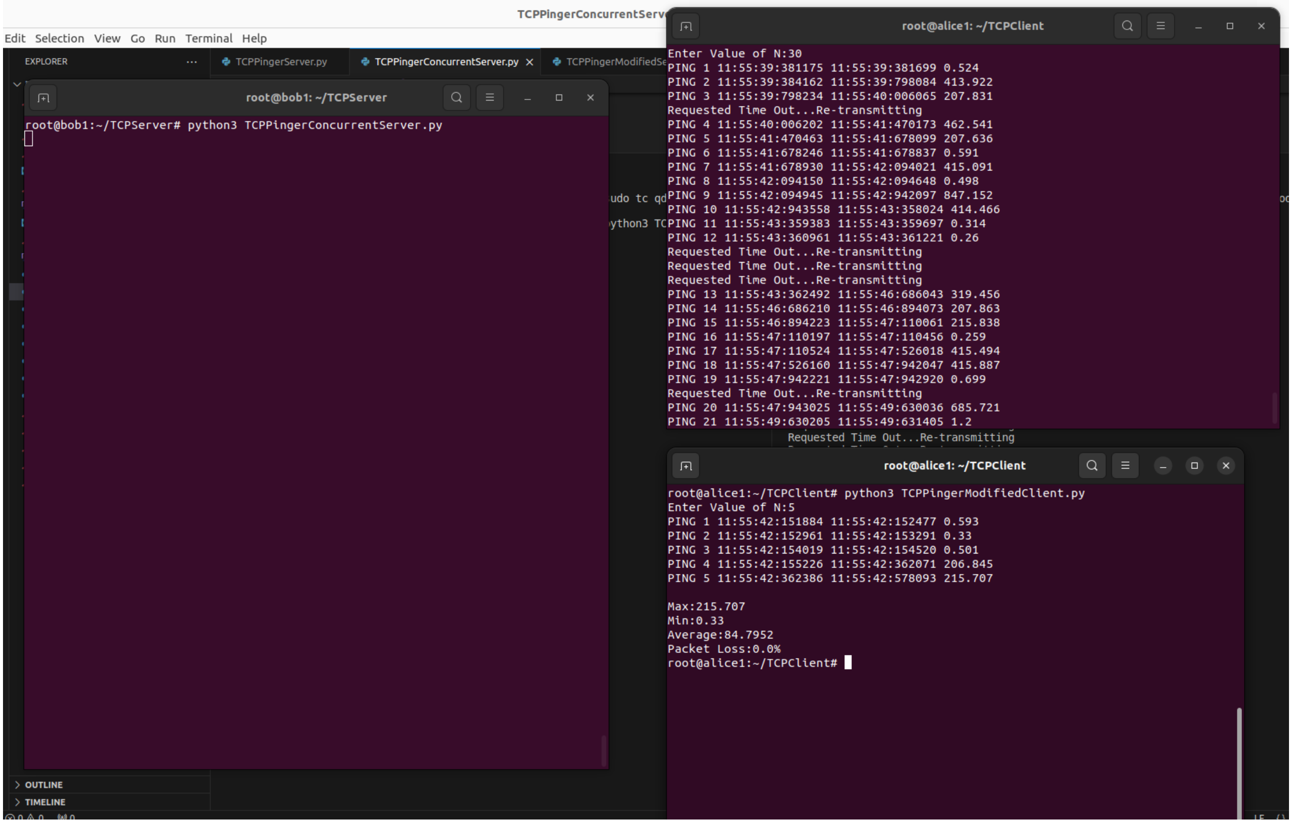
A multithreaded Ping server with TCP protocol receives Ping echo requests from clients, processes them concurrently using multiple threads, and sends responses back to the clients. Each thread handles an individual client connection, ensuring that multiple clients can send Ping requests simultaneously without blocking the server. The server records the round-trip time (RTT) for each Ping request, calculates statistics like minimum, maximum, and average RTT, and may also track packet loss. This architecture enhances server efficiency, allowing it to respond to multiple clients in parallel while measuring network response times.



Conatiner alice1 instance 1



Conatiner alice1 instance 2



clearly it can be seen that some timestamp between both the instnaces of container overlaps which clearly indicates the concurrency

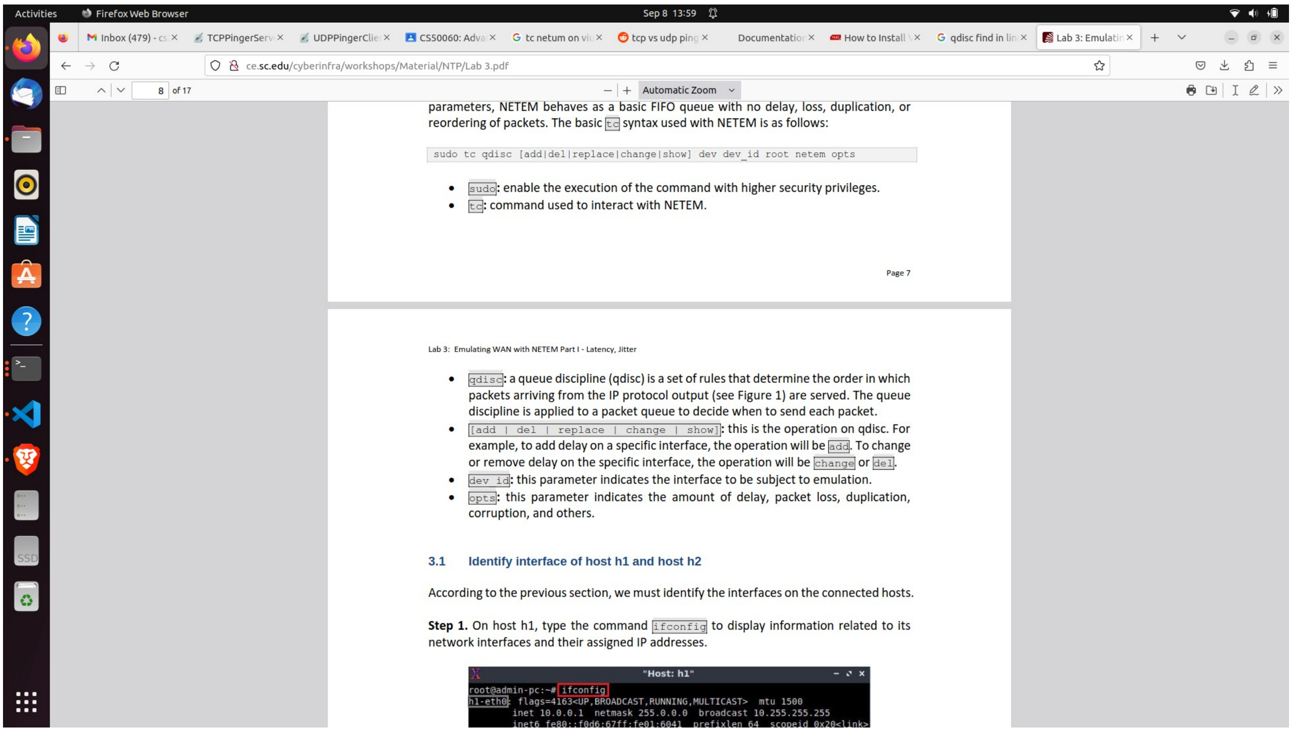
ANTI-PLAGIARISM Statement

I certify that this assignment/report is my own work, based on my personal study and/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: YugPatel, CS23MTECH14019
Date:11/09/2023
Signature: Yug

References:

1. 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
8. For TC-netem



9. For Concurrency

https://github.com/nikhilroxtomar/multithread-client-server-in-python