

# 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 connection, 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%.

```
root@bob1: ~/UDPServer
root@bob1:~/UDPServer# python3 UDPPingerServer.py

root@alice1: ~/UDPCClient
root@alice1:~/UDPCClient# python3 UDPPingerClient.py
Enter Value of N:10
Requested Time Out
PING 2 16:57:13:029200 16:57:13:030539 1.339
PING 3 16:57:13:030755 16:57:13:031601 0.846
PING 4 16:57:13:032760 16:57:13:033109 0.349
PING 5 16:57:13:034195 16:57:13:034845 0.65
Requested Time Out
PING 7 16:57:14:037411 16:57:14:038632 1.221
Requested Time Out
PING 9 16:57:15:040298 16:57:15:041600 1.302
PING 10 16:57:15:041765 16:57:15:042705 0.94

Max:1.339
Min:0.349
Average:0.9495714285714284
Packet Loss:30.0%
root@alice1:~/UDPCClient#
```

*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 accurate 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 Interface is mention below with on the top most line of server side console

```
root@bob1: ~/UDPServer
root@bob1:~/UDPServer# sudo tc qdisc change dev eth0 root netem loss 33%
root@bob1:~/UDPServer# python3 UDPPingerModifiedServer.py

root@alice1: ~/UDPCClient
root@alice1:~/UDPCClient# python3 UDPPingerClient.py
Enter Value of N:12
Requested Time Out
PING 2 16:59:44:780371 16:59:44:780920 0.549
Requested Time Out
Requested Time Out
PING 5 16:59:46:784013 16:59:46:784535 0.522
PING 6 16:59:46:784671 16:59:46:784915 0.244
Requested Time Out
PING 8 16:59:47:786426 16:59:47:786969 0.543
PING 9 16:59:47:787104 16:59:47:787348 0.244
PING 10 16:59:47:787433 16:59:47:787642 0.209
PING 11 16:59:47:789520 16:59:47:789955 0.435
PING 12 16:59:47:790206 16:59:47:790469 0.263

Max:0.549
Min:0.209
Average:0.37612500000000004
Packet Loss:33.33333333333333%
root@alice1:~/UDPCClient#
```

*Emulation of packet loss*

## TASK – 2: TCP Pinger

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

### TCPPingServer.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()
```

### TCPPingModifiedServer.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 netem 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()
```

### TCPPingConcurrentServer.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.

```

# 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 handling a particular 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()

```

### TCPPingClient.py

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 connecgion, 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)
    # Attempts for connection with Server, listing 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}%")

```

### TCPPingModifiedClient.py

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 handling a particular 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:

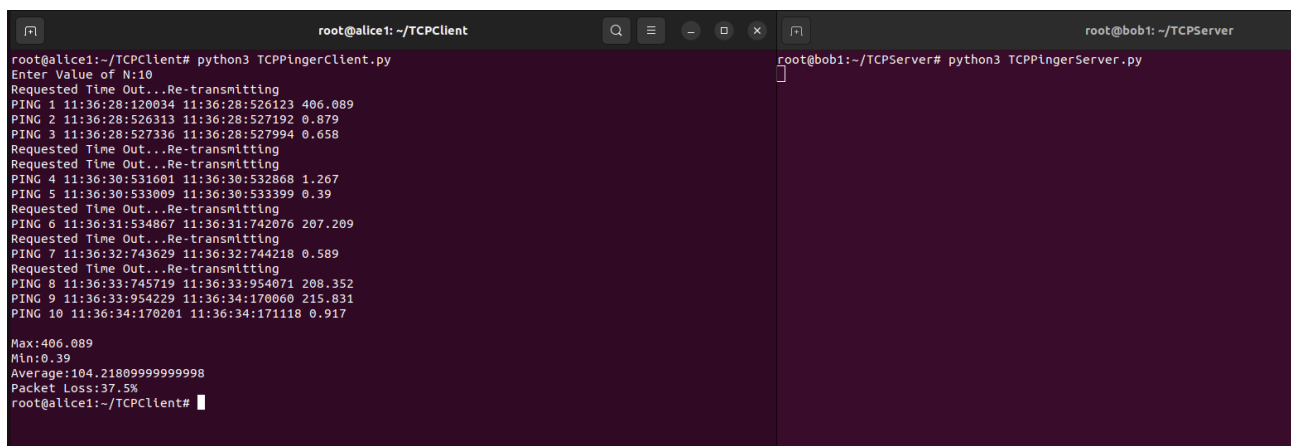
### (1) Simulation of packet loss at application layer :-

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 received any response from the server within a timeframe of one second. Here an interesting 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 iteration on the server side making server to wait the client for 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 won't be the case as generating random number is uncontrollable, not in our hand.

In this case the packet loss is 37.5%.

The only difference here from ping with UDP protocol is that because TCP guarantees reliable transport so even after packet loss the server reattempts 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 in case 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 response from server and that leads to deadlock.



```
root@alice1: ~/TCPClient
python3 TCPpingClient.py
Enter Value of N:10
Requested Time Out...Re-transmitting
PING 1 11:36:28:120034 11:36:28:526123 406.089
PING 2 11:36:28:526313 11:36:28:527192 0.879
PING 3 11:36:28:527336 11:36:28:527994 0.658
Requested Time Out...Re-transmitting
Requested Time Out...Re-transmitting
PING 4 11:36:30:531601 11:36:30:532868 1.267
PING 5 11:36:30:533009 11:36:30:533399 0.39
Requested Time Out...Re-transmitting
PING 6 11:36:31:534867 11:36:31:742076 207.209
Requested Time Out...Re-transmitting
PING 7 11:36:32:743629 11:36:32:744218 0.589
Requested Time Out...Re-transmitting
PING 8 11:36:33:745719 11:36:33:954071 208.352
PING 9 11:36:33:954229 11:36:34:170060 215.831
PING 10 11:36:34:170201 11:36:34:171118 0.917

Max:406.089
Min:0.39
Average:104.21809999999998
Packet Loss:37.5%
root@alice1:~/TCPClient#
```

```
root@bob1: ~/TCPServer
python3 TCPpingServer.py
```

*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 accurate 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 Interface is mention below with on the top most line of server side console.

The only difference here from ping with UDP protocol is that because TCP guarantees reliable transport so even after packet loss the server reattempts 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 automatically.

```

yug@yug-HP-Pavilion-x360-Convertible-14-dh0xxx:~/Downloads$ sudo tc qdisc change dev
lo root netem loss 33%
yug@yug-HP-Pavilion-x360-Convertible-14-dh0xxx:~/Downloads$ python3 TCPpingModifie
Server.py
[]

yug@yug-HP-Pavilion-x360-Convertible-14-dh0xxx:~/Downloads$ python3 TCPpingModifie
dClient.py
Enter Value of N:10
PING 1 17:21:07:380498 17:21:07:381239 0.741
PING 2 17:21:07:381441 17:21:07:588438 206.997
PING 3 17:21:07:588620 17:21:07:588992 0.372
PING 4 17:21:07:589137 17:21:07:796419 207.282
Requested Time Out...Re-transmitting
Requested Time Out...Re-transmitting
Requested Time Out...Re-transmitting
Requested Time Out...Re-transmitting
PING 5 17:21:07:796600 17:21:12:820260 17.19
PING 6 17:21:12:820443 17:21:12:820776 0.333
PING 7 17:21:12:820904 17:21:12:821204 0.3
PING 8 17:21:12:821322 17:21:13:364101 542.779
Requested Time Out...Re-transmitting
PING 9 17:21:13:364307 17:21:15:028405 662.925
Requested Time Out...Re-transmitting
Requested Time Out...Re-transmitting
Requested Time Out...Re-transmitting
Requested Time Out...Re-transmitting
Requested Time Out...Re-transmitting
Requested Time Out...Re-transmitting
Requested Time Out...Re-transmitting
Requested Time Out...Re-transmitting
Requested Time Out...Re-transmitting
Requested Time Out...Re-transmitting
PING 10 17:21:15:028620 17:21:27:668239 622.557

Max:662.925
Min:0.3
Average:226.1476
Packet Loss:64.28571428571429%
yug@yug-HP-Pavilion-x360-Convertible-14-dh0xxx:~/Downloads$

```

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.

```

root@alice1: ~/TCPClient
Enter Value of N:30
PING 1 11:55:39:381175 11:55:39:381699 0.524
PING 2 11:55:39:384162 11:55:39:798084 413.922
PING 3 11:55:39:798234 11:55:40:006065 207.831
Requested Time Out...Re-transmitting
PING 4 11:55:40:006202 11:55:41:470173 462.541
PING 5 11:55:41:470463 11:55:41:678099 207.636
PING 6 11:55:41:678246 11:55:41:678837 0.591
PING 7 11:55:41:678930 11:55:42:094021 415.091
PING 8 11:55:42:094150 11:55:42:094648 0.498
PING 9 11:55:42:094945 11:55:42:942097 847.152
PING 10 11:55:42:943558 11:55:43:358024 414.466
PING 11 11:55:43:359383 11:55:43:359697 0.314
PING 12 11:55:43:360961 11:55:43:361221 0.26
Requested Time Out...Re-transmitting
Requested Time Out...Re-transmitting
PING 13 11:55:43:362492 11:55:46:686043 319.456
PING 14 11:55:46:686210 11:55:46:894073 207.863
PING 15 11:55:46:894223 11:55:47:110061 215.838
PING 16 11:55:47:110197 11:55:47:110450 0.259
PING 17 11:55:47:110524 11:55:47:526018 415.494
PING 18 11:55:47:526160 11:55:47:942047 415.887
PING 19 11:55:47:942221 11:55:47:942920 0.699
Requested Time Out...Re-transmitting
PING 20 11:55:47:943025 11:55:49:630036 685.721
PING 21 11:55:49:630205 11:55:49:631405 1.2
Requested Time Out...Re-transmitting

```

Container alice1 instance 1

```

root@alice1: ~/TCPClient# python3 TCPpingModifiedClient.py
Enter Value of N:5
PING 1 11:55:42:151884 11:55:42:152477 0.593
PING 2 11:55:42:152961 11:55:42:153291 0.33
PING 3 11:55:42:154019 11:55:42:154520 0.501
PING 4 11:55:42:155226 11:55:42:362071 206.845
PING 5 11:55:42:362386 11:55:42:578093 215.707

Max:215.707
Min:0.33
Average:84.7952
Packet Loss:0.0%
root@alice1:~/TCPClient#

```

Container alice1 instance 2

```
root@bob1: ~/TCPServer
python3 TCPPingConcurrentServer.py

root@alice1: ~/TCPClient
Enter Value of N:30
PING 1 11:55:39:381175 11:55:39:381699 0.524
PING 2 11:55:39:384162 11:55:39:798084 413.922
PING 3 11:55:39:798234 11:55:40:006065 207.831
Requested Time Out...Re-transmitting
PING 4 11:55:40:006202 11:55:41:470173 462.541
PING 5 11:55:41:470463 11:55:41:678099 207.636
PING 6 11:55:41:678246 11:55:41:678837 0.591
PING 7 11:55:41:678930 11:55:42:094021 415.091
PING 8 11:55:42:094150 11:55:42:094648 0.498
PING 9 11:55:42:094945 11:55:42:942097 847.152
PING 10 11:55:42:943558 11:55:43:358024 414.466
PING 11 11:55:43:359383 11:55:43:359697 0.314
PING 12 11:55:43:360961 11:55:43:361221 0.26
Requested Time Out...Re-transmitting
Requested Time Out...Re-transmitting
Requested Time Out...Re-transmitting
PING 13 11:55:43:362492 11:55:46:686043 319.456
PING 14 11:55:46:686210 11:55:46:894073 207.863
PING 15 11:55:46:894223 11:55:47:110061 215.838
PING 16 11:55:47:110197 11:55:47:110456 0.259
PING 17 11:55:47:110524 11:55:47:526018 415.494
PING 18 11:55:47:526100 11:55:47:942047 415.887
PING 19 11:55:47:942221 11:55:47:942920 0.699
Requested Time Out...Re-transmitting
PING 20 11:55:47:943025 11:55:49:630036 685.721
PING 21 11:55:49:630205 11:55:49:631405 1.2
Requested Time Out...Re-transmitting

root@alice1: ~/TCPClient
python3 TCPPingModifiedClient.py
Enter Value of N:5
PING 1 11:55:42:151884 11:55:42:152477 0.593
PING 2 11:55:42:152961 11:55:42:153291 0.33
PING 3 11:55:42:154019 11:55:42:154520 0.501
PING 4 11:55:42:155226 11:55:42:362071 206.845
PING 5 11:55:42:362386 11:55:42:578093 215.707
Max:215.707
Min:0.33
Average:84.7952
Packet Loss:0.0%
root@alice1:~/TCPClient#
```

clearly it can be seen that some timestamp between both the instances 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.

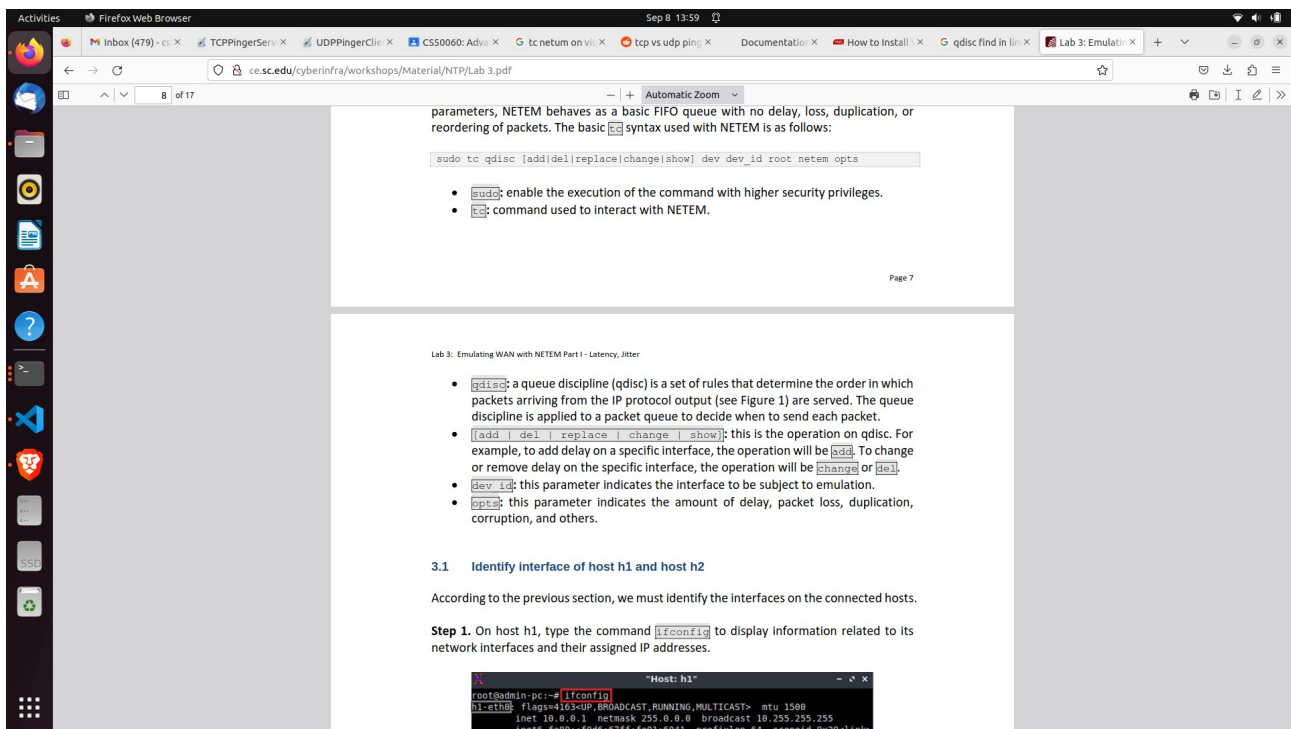
Name: YugPatel, CS23MTECH14019

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](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>