

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

```
root@bob1: ~/UDP
root@bob1:~/UDP# python3 UDPPingerServer.py
[]

root@alice1: ~/UDP
root@alice1:~/UDP# python3 UDPPingerClient.py
Enter the no. of packets: 20
PING 1 1694329671.6761272 -- 1694329671.6766896 0.000562
PING 2 1694329671.6767976 -- 1694329671.677122 0.000324
Request timed out
Request timed out
PING 5 1694329673.680033 -- 1694329673.6807613 0.000728
Request timed out
PING 7 1694329674.6821778 -- 1694329674.6828692 0.000691
Request timed out
PING 9 1694329675.6842635 -- 1694329675.6849153 0.000652
PING 10 1694329675.6849992 -- 1694329675.68541 0.000411
Request timed out
Request timed out
PING 13 1694329677.688087 -- 1694329677.688438 0.000351
Request timed out
PING 15 1694329678.6898603 -- 1694329678.690517 0.000657
PING 16 1694329678.6905982 -- 1694329678.6909895 0.000391
PING 17 1694329678.6910455 -- 1694329678.6913996 0.000354
PING 18 1694329678.6914532 -- 1694329678.6917975 0.000344
PING 19 1694329678.6918788 -- 1694329678.692231 0.000352
PING 20 1694329678.6922843 -- 1694329678.6930852 0.000801

Max RTT: 0.000801
Min RTT: 0.000324
Average RTT: 0.000331
Packet loss: 35.0%
root@alice1:~/UDP#
```

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.

```
root@bob1: ~/UDP
message, address = serverSocket.recvfrom(1024)
KeyboardInterrupt

root@bob1:~/UDP# ls
UDPPingerModifiedServer.py UDPPingerServer.py
root@bob1:~/UDP# vim UDPPingerModifiedServer.py
root@bob1:~/UDP# python3 UDPPingerModifiedServer.py
^CTraceback (most recent call last):
  File "UDPPingerModifiedServer.py", line 14, in <module>
    message, address = serverSocket.recvfrom(1024)
KeyboardInterrupt

root@bob1:~/UDP# sudo tc qdisc add dev eth0 root netem loss 33%
root@bob1:~/UDP# python3 UDPPingerModifiedServer.py
█

root@alice1: ~/UDP
root@alice1:~/UDP# python3 UDPPingerClient.py
Enter the no. of packets: 20
PING 1 1694330269.332963 -- 1694330269.333696 0.000733
PING 2 1694330269.333805 -- 1694330269.3345664 0.000761
Request timed out
PING 4 1694330270.3359616 -- 1694330270.3364298 0.000468
PING 5 1694330270.3365445 -- 1694330270.3373666 0.000822
Request timed out
PING 7 1694330271.3387341 -- 1694330271.3392444 0.00051
Request timed out
PING 9 1694330272.3405898 -- 1694330272.3409233 0.000334
PING 10 1694330272.340996 -- 1694330272.3411736 0.000178
PING 11 1694330272.3412232 -- 1694330272.3413768 0.000154
PING 12 1694330272.341421 -- 1694330272.3417184 0.000298
PING 13 1694330272.3417985 -- 1694330272.3438385 0.00204
Request timed out
PING 15 1694330273.3452337 -- 1694330273.3458176 0.000584
PING 16 1694330273.345898 -- 1694330273.346318 0.00042
Request timed out
PING 18 1694330274.3476546 -- 1694330274.3483133 0.000659
Request timed out
PING 20 1694330275.3497226 -- 1694330275.350056 0.000333

Max RTT: 0.00204
Min RTT: 0.000154
Average RTT: 0.000415
Packet loss: 30.0%
root@alice1:~/UDP# █
```

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

```
# TCPFingerServer.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 from
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 from 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

```

# TCPFingerClient.py
# Libraries required for socket & calculating 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 retransmitted
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)}%")

```

TCPPingServer.py TCPPingClient.py Artificial Packet loss in VM's container:

```
root@bob1: ~/TCP
root@bob1:~/TCP# python3 TCPPingServer.py
Connection with ('172.31.0.2', 57888) has been established
root@bob1:~/TCP#

root@alice1: ~/TCP
root@alice1:~/TCP# python3 TCPPingClient.py
Enter no. of Packets: 20
PING 1 1694345606.2694614 -- 1694345606.2700303 0.000569
PING 2 1694345606.2702127 -- 1694345606.2709315 0.000719
PING 3 1694345606.2710192 -- 1694345606.2713392 0.00032
PING 4 1694345606.2714257 -- 1694345606.272129 0.000703
PING 5 1694345606.2721977 -- 1694345606.2725134 0.000316
PING 6 1694345606.273217 -- 1694345606.2738552 0.000638
PING 7 1694345606.2742562 -- 1694345606.274762 0.000506
PING 8 1694345606.275309 -- 1694345606.275643 0.000334
PING 9 1694345606.276111 -- 1694345606.276635 0.000524
PING 10 1694345606.2768755 -- 1694345606.2772496 0.000374
PING 11 1694345606.2778337 -- 1694345606.278214 0.00038
PING 12 1694345606.2785482 -- 1694345606.2788305 0.000282
PING 13 1694345606.2793128 -- 1694345606.2796981 0.000385
PING 14 1694345606.2803433 -- 1694345606.2805564 0.000213
Requested Time Out....Re-transmitting
PING 15 1694345607.2824173 -- 1694345607.2827482 0.000331
PING 16 1694345607.2828557 -- 1694345607.2830064 0.000151
Requested Time Out....Re-transmitting
Requested Time Out....Re-transmitting
Requested Time Out....Re-transmitting
PING 17 1694345610.287253 -- 1694345610.287778 0.000525
PING 18 1694345610.2878685 -- 1694345610.2890246 0.001156
PING 19 1694345610.289096 -- 1694345610.2894316 0.000335
Requested Time Out....Re-transmitting
PING 20 1694345611.290792 -- 1694345611.2911358 0.000344

Packet Retransmitted: 5
Average RTT: 0.000455
Minimum RTT: 0.000151
Maximum RTT: 0.001156
As all the packets are receive by the client due to retransmissions
Therefore Packet loss: 0%
root@alice1:~/TCP#
```

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

TCP Modified Server code

```
# TCPPingModifiedServer.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 from
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 from 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 & calculating 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 retransmitted
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 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)}%")

```

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

TCPPingModifiedServer.py TCPPingModifiedClient.py with NIC Packet loss of 33% at VM's container

```
root@bob1: ~/TCP
root@bob1:~/TCP# sudo tc qdisc add dev eth0 root netem loss 33%
root@bob1:~/TCP# python3 TCPPingModifiedServer.py
Connection with ('172.31.0.2', 48092) has been established
root@bob1:~/TCP# python3 TCPPingModifiedServer.py
Connection with ('172.31.0.2', 34282) has been established
root@bob1:~/TCP# python3 TCPPingModifiedServer.py
Connection with ('172.31.0.2', 59530) has been established
root@bob1:~/TCP#

root@alice1: ~/TCP
PING 36 1694410502.8695285 -- 1694410502.870117 0.000588
PING 37 1694410502.8704662 -- 1694410502.8706388 0.000173
PING 38 1694410502.870889 -- 1694410502.8710644 0.000175
PING 39 1694410502.8717773 -- 1694410502.8719535 0.000176
Requested Time Out...Re-transmitting
PING 40 1694410503.8736975 -- 1694410504.5599868 0.686289
PING 41 1694410504.5601115 -- 1694410504.7679322 0.207821
PING 42 1694410504.7680569 -- 1694410504.7686472 0.00059
PING 43 1694410504.7687252 -- 1694410504.7690837 0.000359
PING 44 1694410504.7691455 -- 1694410504.9759922 0.206847
PING 45 1694410504.976557 -- 1694410504.9768367 0.00028
Requested Time Out...Re-transmitting
PING 46 1694410505.9786348 -- 1694410506.639994 0.661359
PING 47 1694410506.640115 -- 1694410506.6404305 0.000315
PING 48 1694410506.6404958 -- 1694410506.64065 0.000154
PING 49 1694410506.6407003 -- 1694410506.6408446 0.000144
PING 50 1694410506.6408923 -- 1694410506.641035 0.000143

Packet Retransmitted: 2
Average RTT: 0.093937
Minimum RTT: 0.000143
Maximum RTT: 0.831313
Packet loss: 3.85%
root@alice1:~/TCP#
```

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 active
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).

TCPPingConcurrentServer.py TCPPingModifiedClient.py with NIC packet loss at VM's container:

```

root@bob1: ~/TCP
root@bob1:~/TCP# sudo tc qdisc show
qdisc noqueue 0: dev lo root refcnt 2
qdisc netem 800f: dev eth0 root refcnt 2 limit 1000 loss 33%
root@bob1:~/TCP# vim TCPPingConcurrentServer.py
root@bob1:~/TCP# python3 TCPPingConcurrentServer.py
Connection with ('172.31.0.2', 34718) has been established
Active Threads 1
Connection with ('172.31.0.2', 34734) has been established
Active Threads 2
^

root@alice1: ~/TCP
PING 34 1694410911.2289424 -- 1694410911.5680017 0.339059
PING 35 1694410911.5681088 -- 1694410911.7759538 0.207845
PING 36 1694410911.7763078 -- 1694410911.9919538 0.215646
PING 37 1694410911.9920568 -- 1694410911.9924958 0.000439
PING 38 1694410911.9925625 -- 1694410911.992969 0.000407
PING 39 1694410911.9930334 -- 1694410911.993372 0.000339
PING 40 1694410911.9936728 -- 1694410912.1999378 0.206265
PING 41 1694410912.2000575 -- 1694410912.2002451 0.000188
PING 42 1694410912.200588 -- 1694410912.2008069 0.000219
PING 43 1694410912.2008858 -- 1694410912.2012928 0.000407
PING 44 1694410912.2013755 -- 1694410912.6159642 0.414589
PING 45 1694410912.6161041 -- 1694410912.616319 0.000215
PING 46 1694410912.616394 -- 1694410913.4559867 0.839593
PING 47 1694410913.4576252 -- 1694410913.457989 0.000364
PING 48 1694410913.4594078 -- 1694410913.6639695 0.204562
PING 49 1694410913.6648624 -- 1694410913.8800616 0.215199
PING 50 1694410913.880166 -- 1694410913.88041 0.000244

Packet Retransmitted: 3
Average RTT: 0.132687
Minimum RTT: 0.00013
Maximum RTT: 0.845884
Packet loss: 5.66%
root@alice1:~/TCP#

root@alice1: ~/TCP
PING 34 1694410911.6420546 -- 1694410912.0559516 0.413897
PING 35 1694410912.0560803 -- 1694410912.0562818 0.000201
PING 36 1694410912.0569677 -- 1694410912.0571747 0.000207
PING 37 1694410912.0578418 -- 1694410912.0580323 0.00019
PING 38 1694410912.0581121 -- 1694410912.0588505 0.000738
PING 39 1694410912.058952 -- 1694410912.0591216 0.00017
PING 40 1694410912.0596783 -- 1694410912.2679703 0.208292
PING 41 1694410912.2681074 -- 1694410912.2683048 0.000197
PING 42 1694410912.2690246 -- 1694410912.269202 0.000177
PING 43 1694410912.2692807 -- 1694410912.2694144 0.000134
PING 44 1694410912.269481 -- 1694410912.2696097 0.000129
PING 45 1694410912.269674 -- 1694410912.4761562 0.206482
PING 46 1694410912.4763055 -- 1694410912.476503 0.000197
PING 47 1694410912.4765747 -- 1694410912.476709 0.000134
PING 48 1694410912.4767754 -- 1694410912.6839364 0.207161
PING 49 1694410912.6840615 -- 1694410912.6842513 0.00019
PING 50 1694410912.6843271 -- 1694410912.6844618 0.000135

Packet Retransmitted: 1
Average RTT: 0.117858
Minimum RTT: 0.000128
Maximum RTT: 0.670169
Packet loss: 1.96%
root@alice1:~/TCP#

```

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 serverSocket.accept() what does this python code return

✦ The `serverSocket.accept()` method in Python returns a tuple of two elements:

1. A new socket object that represents the connection to the client.
2. A tuple containing the address of the client.

The address tuple will contain (host, port) for IPv4 connections or (host, port, flowinfo, scopeid) for IPv6.

B like in C++ we use INT_MIN then how to implemet same thing in python

✦ Sure, the equivalent of `INT_MIN` in Python is `sys.maxsize * -1`.

The `sys` module in Python provides access to various system-specific variables and functions. The `maxsize` attribute of the `sys` module is the maximum value that an integer can hold. Multiplying it by -1 gives us the minimum value that an integer can hold.

B `threading.Thread()`

what does the above syntax does ?



The `threading.Thread()` syntax in Python is used to create a new thread of execution. It's a part of the `threading` module, which provides a way to work with threads in Python.

When you create a new `Thread` object using `threading.Thread()`, you can pass a target function that you want to run in a separate thread. Here's the basic syntax:

```
python
import threading

def my_function():
    # Your code here

# Create a Thread object
my_thread = threading.Thread(target=my_function)
```

Copy code

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

Copy code

```
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

Copy code

```
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

```
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>