Report on Assignment 3
Socket Programming

Name- Anupam Kumar   Roll. No.- 11940160

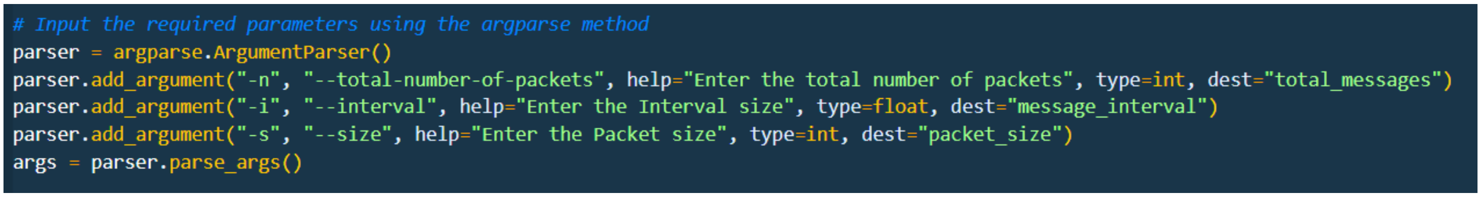
Video Explanation (folder containing videos for all parts)

PART 1: Extending Echo Client/Server for network analysis

1. The source links: client.py, server.py, video

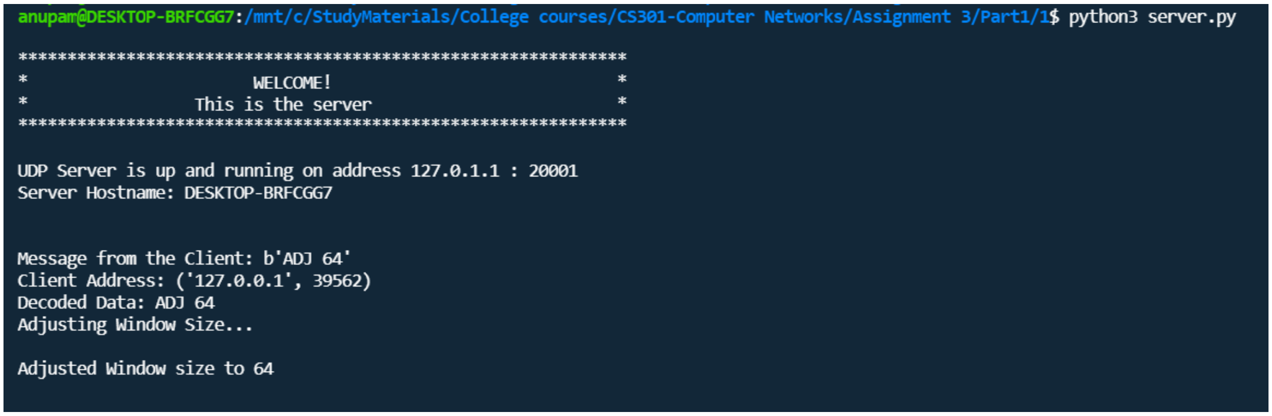
- Process Overview:
    - I have implemented the UDP echo-client server for measuring the Round trip Time between the client and the server.
    - The server creates a UDP Socket and is running and receiving the messages from the client.
    - The server terminates when the client requests termination.
    - The client creates a UDP Socket and it sends the echo packets to the server.
    - We are calculating the Round Trip Time for each packet sent and the average RTT and Average Loss at the end.
    - Especially for getting Loss, we have defined Packet Loss with a probability of 0.1 in the server.

- How To Run and Explanation:
    - Since it is mentioned to take inputs using the Command Line Arguments.
    - I have taken the Packet Number, Interval Size, and Packet Size using arguments as shown below:

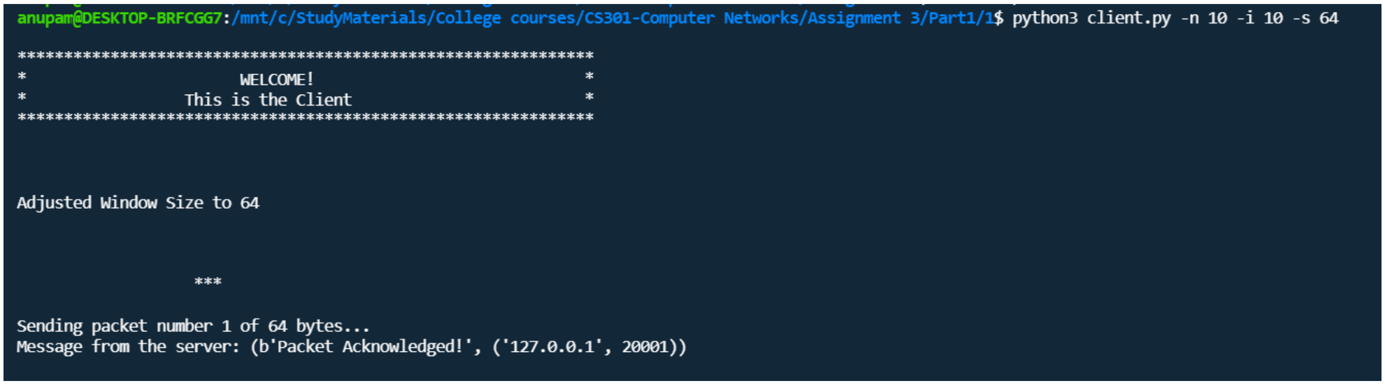


```
# Input the required parameters using the argparse method
parser = argparse.ArgumentParser()
parser.add_argument("-n", "--total-number-of-packets", help="Enter the total number of packets", type=int, dest="total_messages")
parser.add_argument("-i", "--interval", help="Enter the Interval size", type=float, dest="message_interval")
parser.add_argument("-s", "--size", help="Enter the Packet size", type=int, dest="packet_size")
args = parser.parse_args()
```

    - -n : Packet Size, -i: Interval Size, -s: Packet Size
    - We first have to run the server.py file using the command:
        - python3 server.py
    - Then, we run the client.py file using the command:
        - python3 client.py -n 10 -i 10 -s 64
        - (enter the value of n, i, and s according to your choice)
    - A demo input is shown below (server and client):

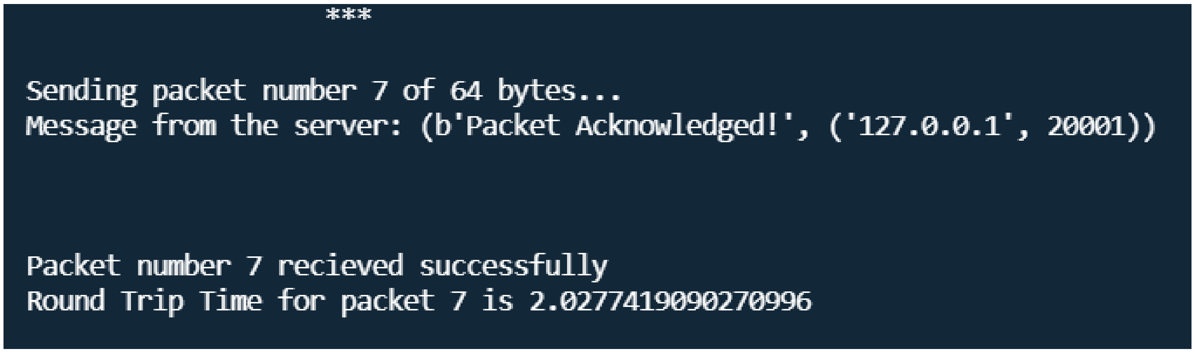


```
anupam@DESKTOP-BRFCGG7:/mnt/c/StudyMaterials/College courses/CS301-Computer Networks/Assignment 3/Part1/1$ python3 server.py

**************************************************************
*                       WELCOME!                             *
*                 This is the server                         *
**************************************************************

UDP Server is up and running on address 127.0.1.1 : 20001
Server Hostname: DESKTOP-BRFCGG7


Message from the Client: b'ADJ 64'
Client Address: ('127.0.0.1', 39562)
Decoded Data: ADJ 64
Adjusting Window Size...

Adjusted Window size to 64
```

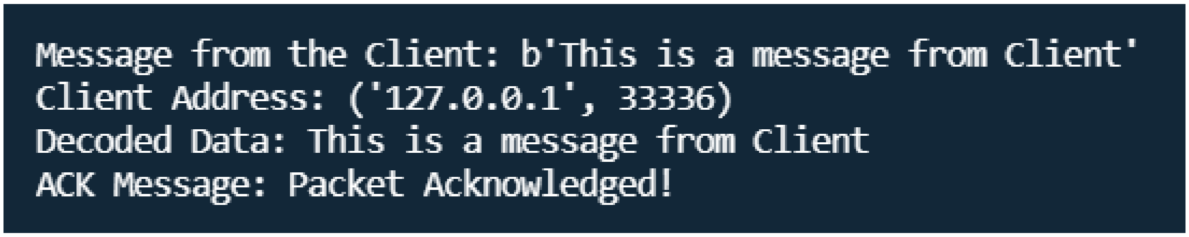


```
anupam@DESKTOP-BRFCGG7:/mnt/c/StudyMaterials/College courses/CS301-Computer Networks/Assignment 3/Part1/1$ python3 client.py -n 10 -i 10 -s 64

**************************************************************
*                       WELCOME!                             *
*                 This is the Client                         *
**************************************************************



Adjusted Window Size to 64



                   ***

Sending packet number 1 of 64 bytes...
Message from the server: (b'Packet Acknowledged!', ('127.0.0.1', 20001))
```

- ○ If the packet is received successfully, the message is displayed for the same as shown below(client and server):

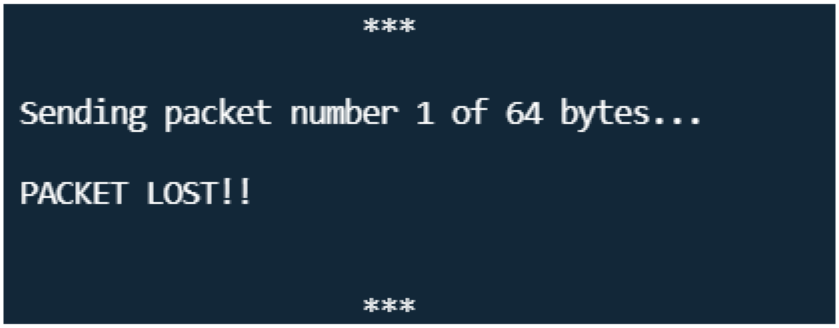


```
                   ***

Sending packet number 7 of 64 bytes...
Message from the server: (b'Packet Acknowledged!', ('127.0.0.1', 20001))



Packet number 7 recieved successfully
Round Trip Time for packet 7 is 2.0277419090270996
```

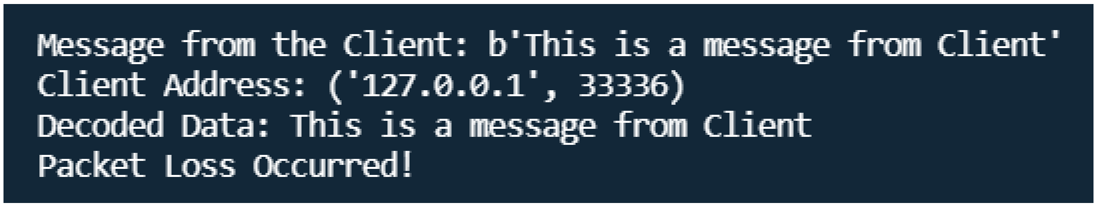


```
Message from the Client: b'This is a message from Client'
Client Address: ('127.0.0.1', 33336)
Decoded Data: This is a message from Client
ACK Message: Packet Acknowledged!
```

- ○ Similarly for the case if the packet is lost as shown below(client and server):

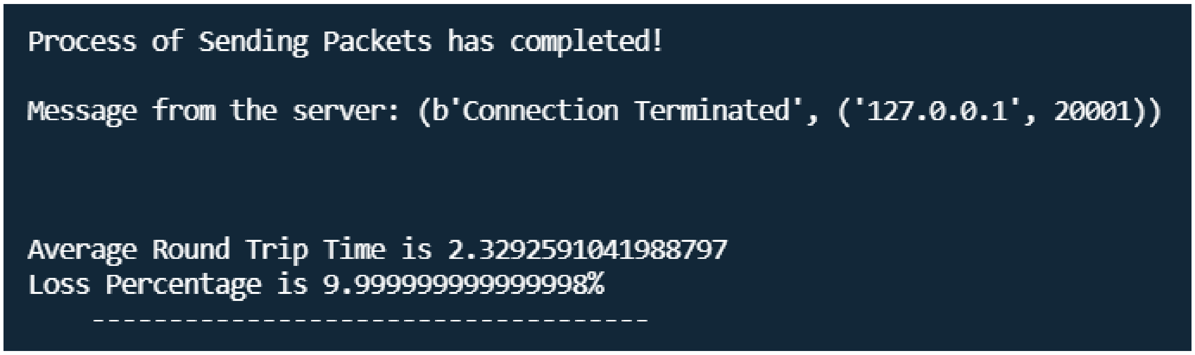


```
                    ***

Sending packet number 1 of 64 bytes...

PACKET LOST!!


                    ***
```



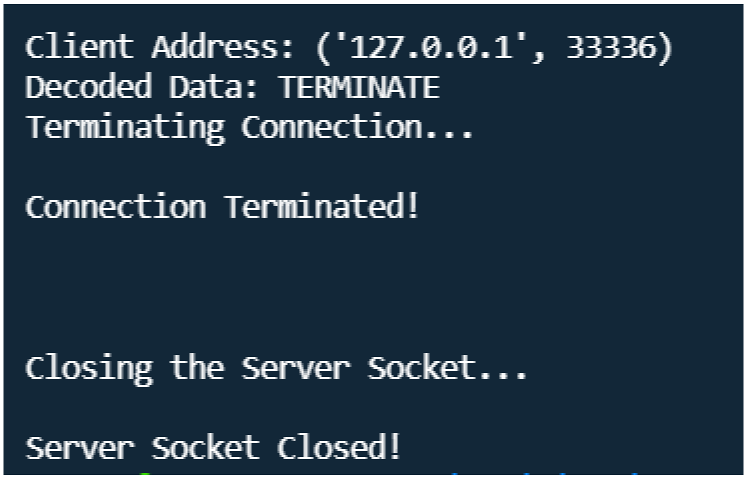
```
Message from the Client: b'This is a message from Client'
Client Address: ('127.0.0.1', 33336)
Decoded Data: This is a message from Client
Packet Loss Occurred!
```

- ○ Average RTT and Loss % is displayed at the end on the client-side:



```
Process of Sending Packets has completed!

Message from the server: (b'Connection Terminated', ('127.0.0.1', 20001))



Average Round Trip Time is 2.3292591041988797
Loss Percentage is 9.999999999999998%
    ------------------------------------
```

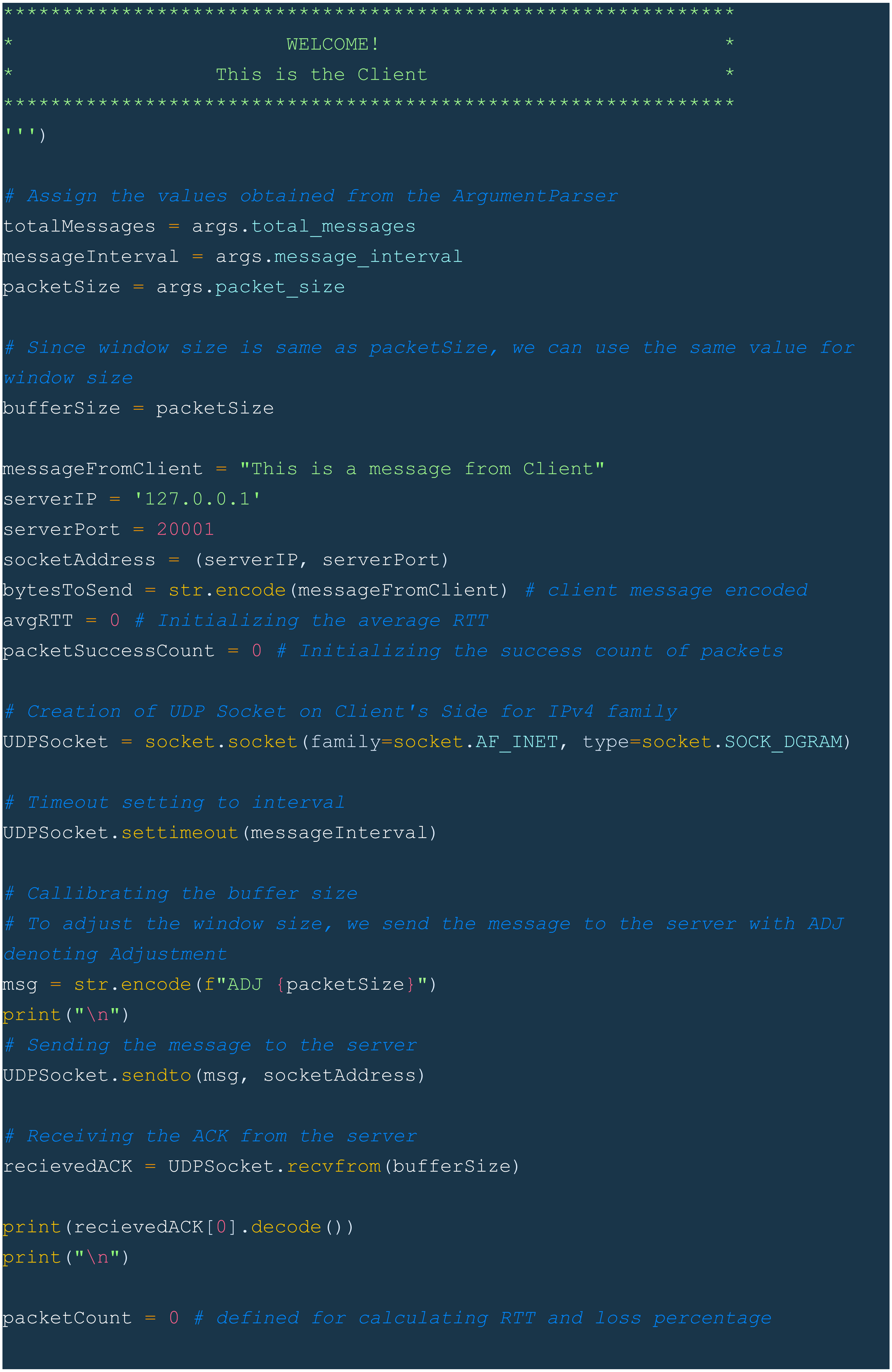
- ○ Termination on the Server-side:



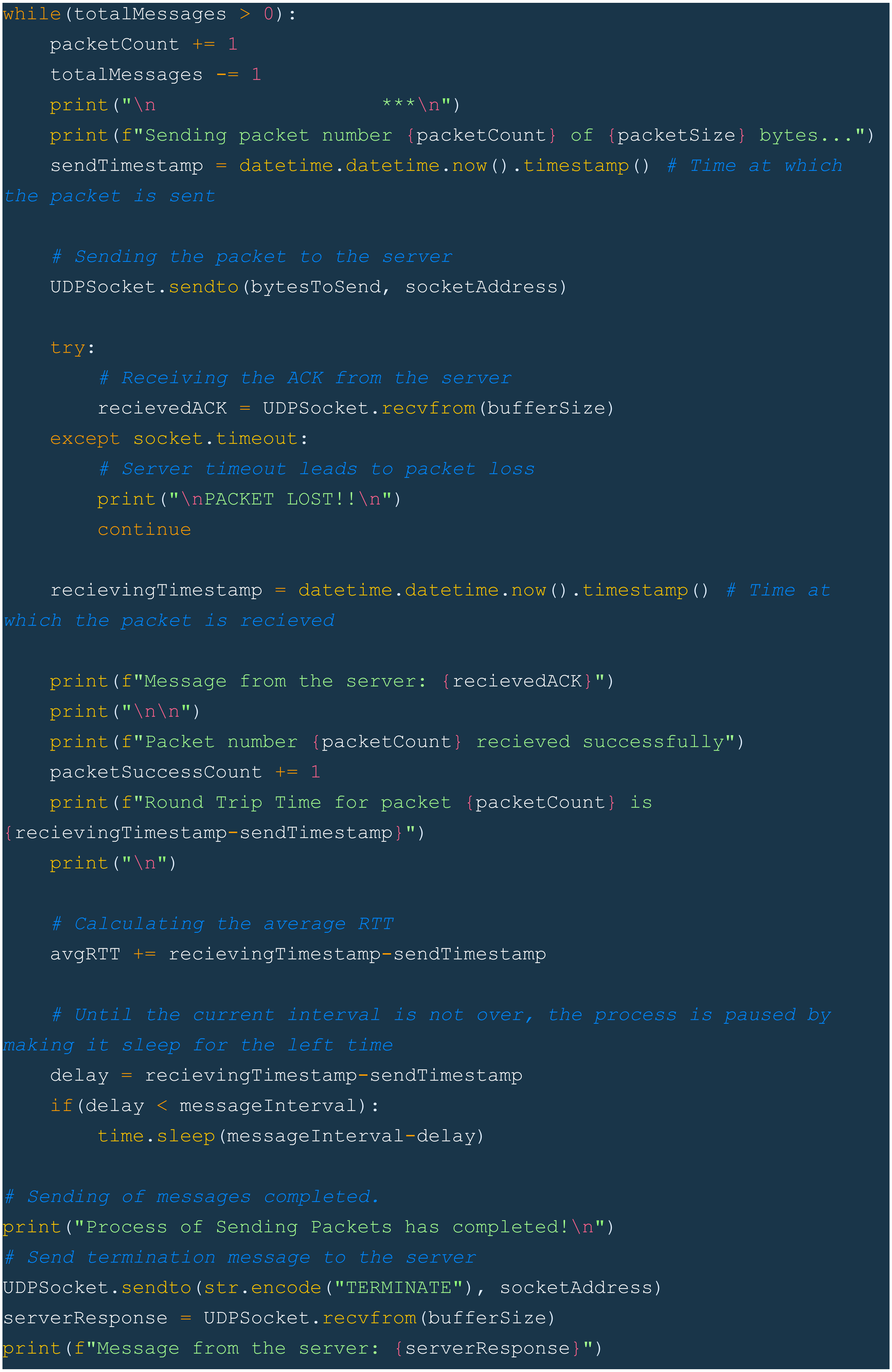
```
Client Address: ('127.0.0.1', 33336)
Decoded Data: TERMINATE
Terminating Connection...

Connection Terminated!



Closing the Server Socket...

Server Socket Closed!
```

- Code:
    - Client Implementation:

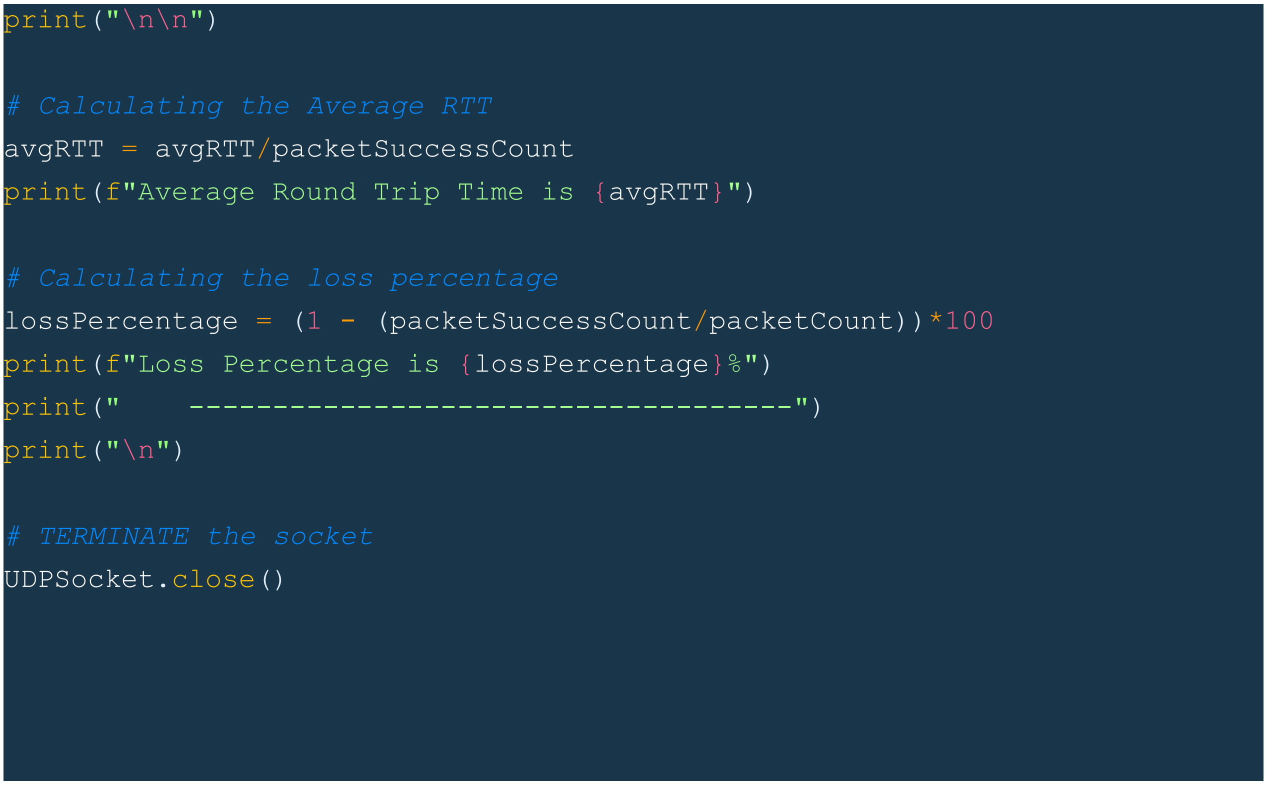
```
# ***PROBLEM STATEMENT***

# Create your own UDP echo client and server application
# to measure round trip time between client and server (similar to “ping” command)

# The client should create a UDP socket and send echo packets to
# server at a given interval, number of echo messages, and given
# packet size (use command line arguments). On reception of the
# packet, server should send the packet back to the client. The
# client on reception of the packet should calculate and display the
# round-trip time. To calculate the round-trip time, you can have
# the timestamp in the packet or/and use some unique identifier
# in the packet. You should also calculate and print the loss
# percentage at the end.


# The code for the client starts here:
import socket
import sys
import time
import datetime
import argparse

# # Input the required parameters using the normal input method
# totalMessages = int(input("Enter the total number of packets: "))
# messageInterval = float(input("Enter the Interval size: "))
# packetSize = int(input("Enter the Packet size: "))

# Input the required parameters using the argparse method
parser = argparse.ArgumentParser()
parser.add_argument("-n", "--total-number-of-packets", help="Enter the total number of packets", type=int, dest="total_messages")
parser.add_argument("-i", "--interval", help="Enter the Interval size", type=float, dest="message_interval")
parser.add_argument("-s", "--size", help="Enter the Packet size", type=int, dest="packet_size")
args = parser.parse_args()

# Begin with a display message
print('''
```



```
**************************************************************
*                       WELCOME!                             *
*                 This is the Client                         *
**************************************************************
''')

# Assign the values obtained from the ArgumentParser
totalMessages = args.total_messages
messageInterval = args.message_interval
packetSize = args.packet_size

# Since window size is same as packetSize, we can use the same value for
window size
bufferSize = packetSize

messageFromClient = "This is a message from Client"
serverIP = '127.0.0.1'
serverPort = 20001
socketAddress = (serverIP, serverPort)
bytesToSend = str.encode(messageFromClient) # client message encoded
avgRTT = 0 # Initializing the average RTT
packetSuccessCount = 0 # Initializing the success count of packets

# Creation of UDP Socket on Client's Side for IPv4 family
UDPSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

# Timeout setting to interval
UDPSocket.settimeout(messageInterval)

# Callibrating the buffer size
# To adjust the window size, we send the message to the server with ADJ
denoting Adjustment
msg = str.encode(f"ADJ {packetSize}")
print("\n")
# Sending the message to the server
UDPSocket.sendto(msg, socketAddress)

# Recieving the ACK from the server
recievedACK = UDPSocket.recvfrom(bufferSize)

print(recievedACK[0].decode())
print("\n")

packetCount = 0 # defined for calculating RTT and loss percentage
```

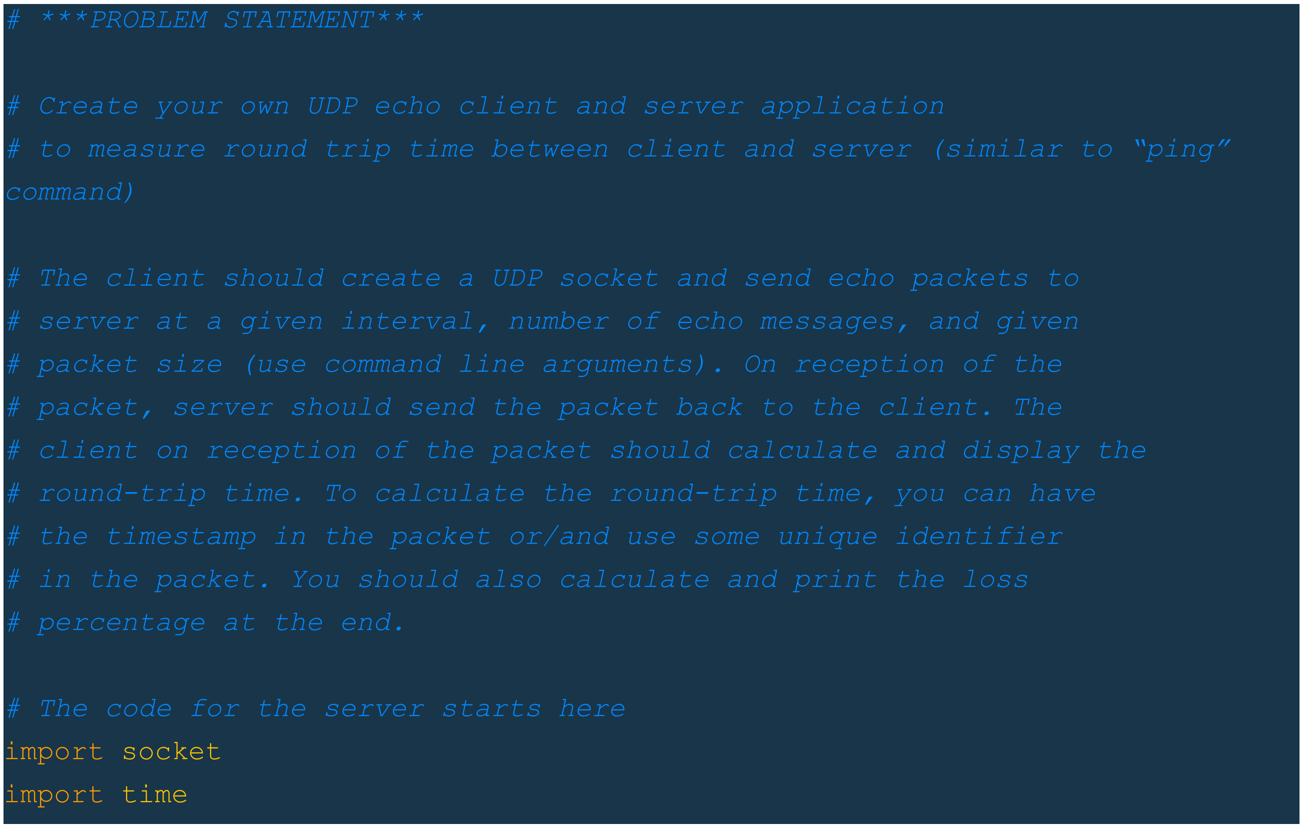


```
while(totalMessages > 0):
    packetCount += 1
    totalMessages -= 1
    print("\n                   ***\n")
    print(f"Sending packet number {packetCount} of {packetSize} bytes...")
    sendTimestamp = datetime.datetime.now().timestamp() # Time at which
the packet is sent

    # Sending the packet to the server
    UDPSocket.sendto(bytesToSend, socketAddress)

    try:
        # Receiving the ACK from the server
        recievedACK = UDPSocket.recvfrom(bufferSize)
    except socket.timeout:
        # Server timeout leads to packet loss
        print("\nPACKET LOST!!\n")
        continue

    recievingTimestamp = datetime.datetime.now().timestamp() # Time at
which the packet is recieved

    print(f"Message from the server: {recievedACK}")
    print("\n\n")
    print(f"Packet number {packetCount} recieved successfully")
    packetSuccessCount += 1
    print(f"Round Trip Time for packet {packetCount} is
{recievingTimestamp-sendTimestamp}")
    print("\n")

    # Calculating the average RTT
    avgRTT += recievingTimestamp-sendTimestamp

    # Until the current interval is not over, the process is paused by
making it sleep for the left time
    delay = recievingTimestamp-sendTimestamp
    if(delay < messageInterval):
        time.sleep(messageInterval-delay)

# Sending of messages completed.
print("Process of Sending Packets has completed!\n")
# Send termination message to the server
UDPSocket.sendto(str.encode("TERMINATE"), socketAddress)
serverResponse = UDPSocket.recvfrom(bufferSize)
print(f"Message from the server: {serverResponse}")
```



```
print("\n\n")

# Calculating the Average RTT
avgRTT = avgRTT/packetSuccessCount
print(f"Average Round Trip Time is {avgRTT}")

# Calculating the loss percentage
lossPercentage = (1 - (packetSuccessCount/packetCount))*100
print(f"Loss Percentage is {lossPercentage}%")
print("    ------------------------------------")
print("\n")

# TERMINATE the socket
UDPSocket.close()
```

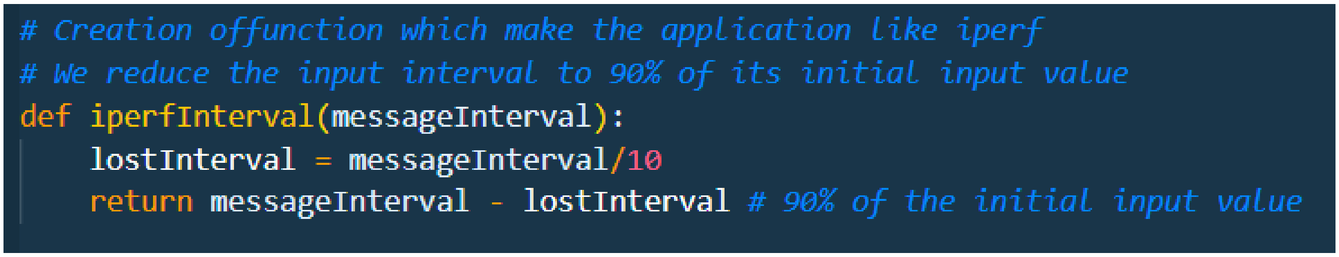
- ○ Server Implementation:



```
# ***PROBLEM STATEMENT***

# Create your own UDP echo client and server application
# to measure round trip time between client and server (similar to "ping"
command)

# The client should create a UDP socket and send echo packets to
# server at a given interval, number of echo messages, and given
# packet size (use command line arguments). On reception of the
# packet, server should send the packet back to the client. The
# client on reception of the packet should calculate and display the
# round-trip time. To calculate the round-trip time, you can have
# the timestamp in the packet or/and use some unique identifier
# in the packet. You should also calculate and print the loss
# percentage at the end.

# The code for the server starts here
import socket
import time
import random
from datetime import datetime

# Start with a display message
print('''
**************************************************************
*                       WELCOME!                             *
*                 This is the server                         *
**************************************************************
''')

serverIP = '127.0.0.1'
serverPort = 20001
socketAddress = (serverIP, serverPort)
serverPublicIP = socket.gethostbyname(socket.gethostname())

bufferSize = 1024

ACKmessage = "Packet Acknowledged!"
encodedACKmessage = ACKmessage.encode()

# Creation of UDP Socket on Server's Side for IPv4 family
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind the socket to the address
serverSocket.bind(socketAddress)

print(f"UDP Server is up and running on address {serverPublicIP} :
{serverPort}")

serverHostname = socket.gethostname()
print(f"Server Hostname: {serverHostname}")

# Listen for incoming connections
while True:
    # Get client requests
    clientRequestData, clientAddress = serverSocket.recvfrom(bufferSize)
    print("\n")
    print(f"Message from the Client: {clientRequestData}")
    print(f"Client Address: {clientAddress}")

    decodedData = clientRequestData.decode()
    print(f"Decoded Data: {decodedData}")

    # If the decoded message required the adjustment of window size
    if decodedData.split()[0] == "ADJ":
        print("Adjusting Window Size...")
        bufferSize = int(decodedData.split(" ")[1])
        print()
        serverSocket.sendto(str.encode(f"Adjusted Window Size to
{bufferSize}"), clientAddress)
        print(f"Adjusted Window size to {bufferSize}")
        print("\n")
        continue

    # If the decoded message requests Termination of Connection
    if decodedData == "TERMINATE":
        print("Terminating Connection...")
        serverSocket.sendto(str.encode("Connection Terminated"),
clientAddress)
        print("\nConnection Terminated!\n")
        break


    # Introducing Packet Delay
    time.sleep(random.randint(1, 5000) / 1000)

    # Introducing Packet Loss with Loss Probability of 0.1
    if random.randint(1, 10) <= 1:
        print("Packet Loss Occurred!\n")
        continue

    # Sending reply to the client with an Acknowledgement message
    serverSocket.sendto(encodedACKmessage, clientAddress)
    print(f"ACK Message: {ACKmessage}")


# Close the socket with a message
print("\n\nClosing the Server Socket...")
serverSocket.close()
print("\nServer Socket Closed!")
```

2. The source links: client.py, server.py, video

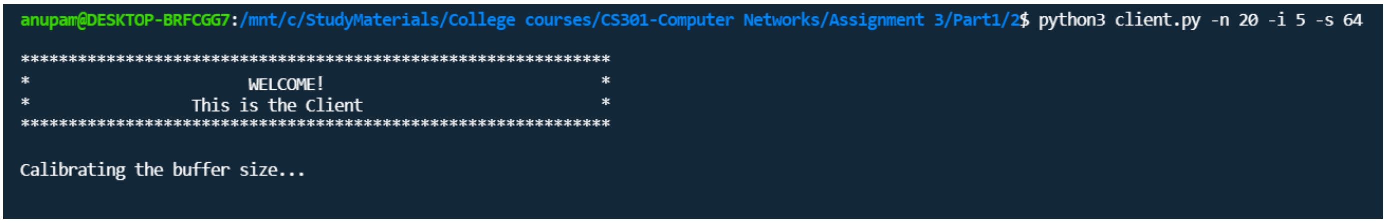
- Process Overview:
    - This is also a UDP echo client-server same as Part 1.1
    - To make it like an iperf application, we have reduced the interval between the two consecutive echo packets by the use of the function as shown:



```
# Creation offunction which make the application like iperf
# We reduce the input interval to 90% of its initial input value
def iperfInterval(messageInterval):
    lostInterval = messageInterval/10
    return messageInterval - lostInterval # 90% of the initial input value
```

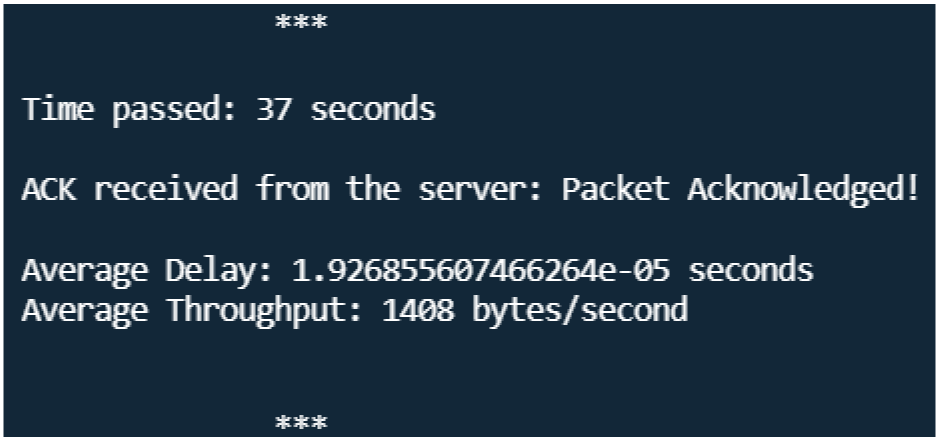
- We have also calculated the average throughput and average delay for each second and plotted it using Matplotlib.

- How To Run and Explanation:
    - We run it in the same way it was done for Part 1.1 by entering the inputs using the command line arguments as shown below:



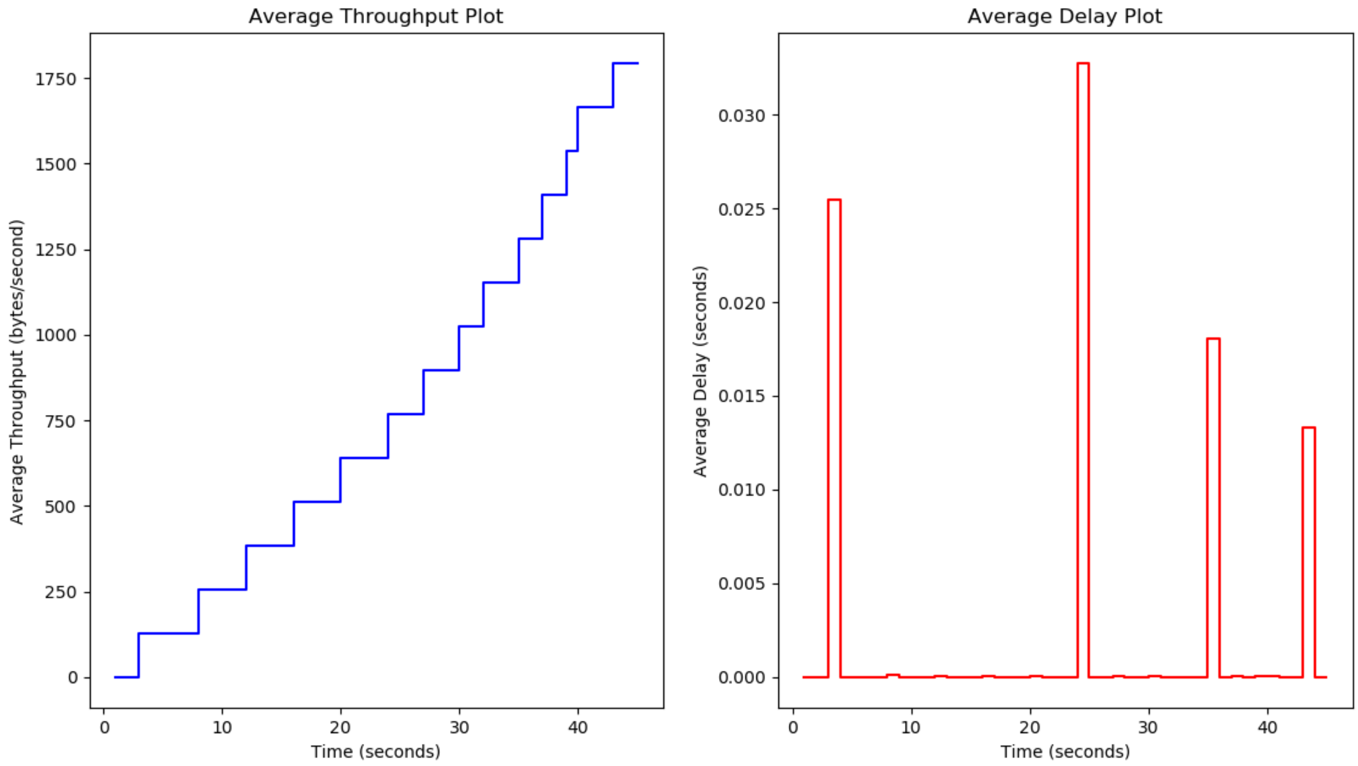
```
anupam@DESKTOP-BRFCGG7:/mnt/c/StudyMaterials/College courses/CS301-Computer Networks/Assignment 3/Part1/2$ python3 client.py -n 20 -i 5 -s 64

**************************************************************
*                      WELCOME!                              *
*                 This is the Client                         *
**************************************************************

Calibrating the buffer size...
```

- For every packet that is successfully sent, we display the Acknowledgement from the server and the average delay and average throughput along with it.

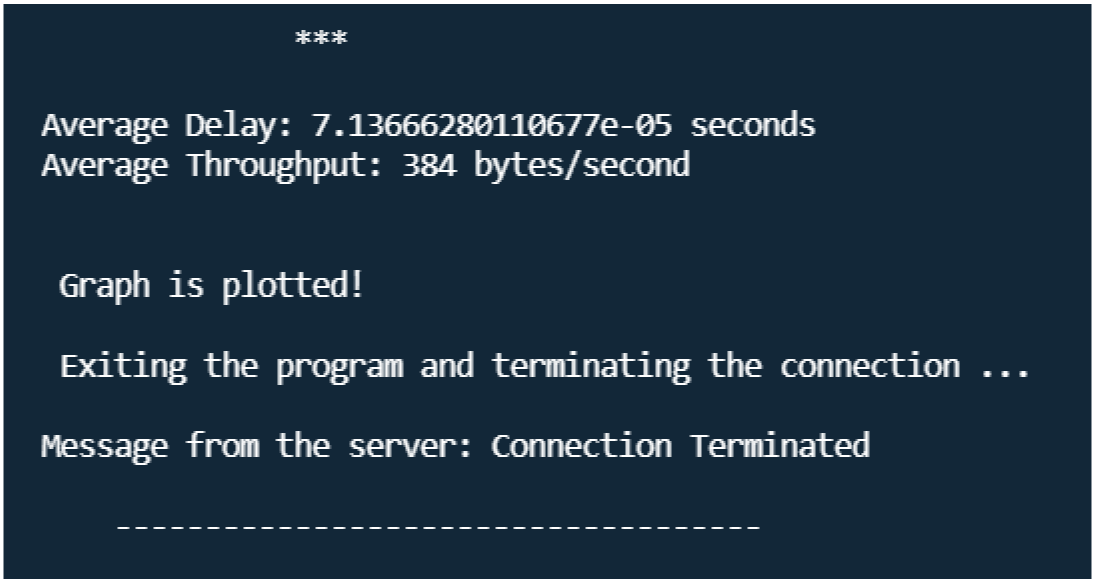


```
                    ***

Time passed: 37 seconds

ACK received from the server: Packet Acknowledged!

Average Delay: 1.926855607466264e-05 seconds
Average Throughput: 1408 bytes/second


                    ***
```

- The server behaves the same way as it did in Part 1.1
- In the end, I have used Matplotlib to plot the graph for average throughput and average delay. One of the graphs obtained is shown below:

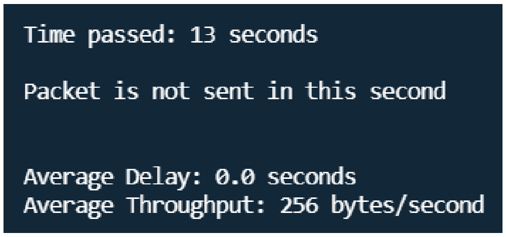


- ○ Termination is shown below (client):

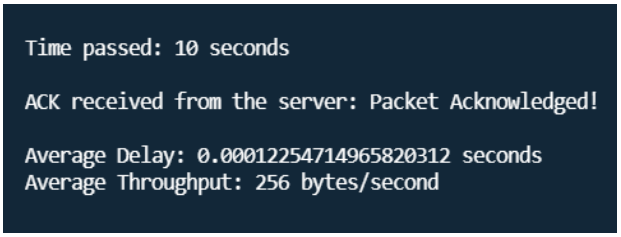


```
                ***

Average Delay: 7.13666280110677e-05 seconds
Average Throughput: 384 bytes/second


 Graph is plotted!

 Exiting the program and terminating the connection ...

Message from the server: Connection Terminated

    ------------------------------------
```

- ○ We are sending the average throughput and average delay each second, so it may not send a packet every second. It displays a suitable message for all the seconds.



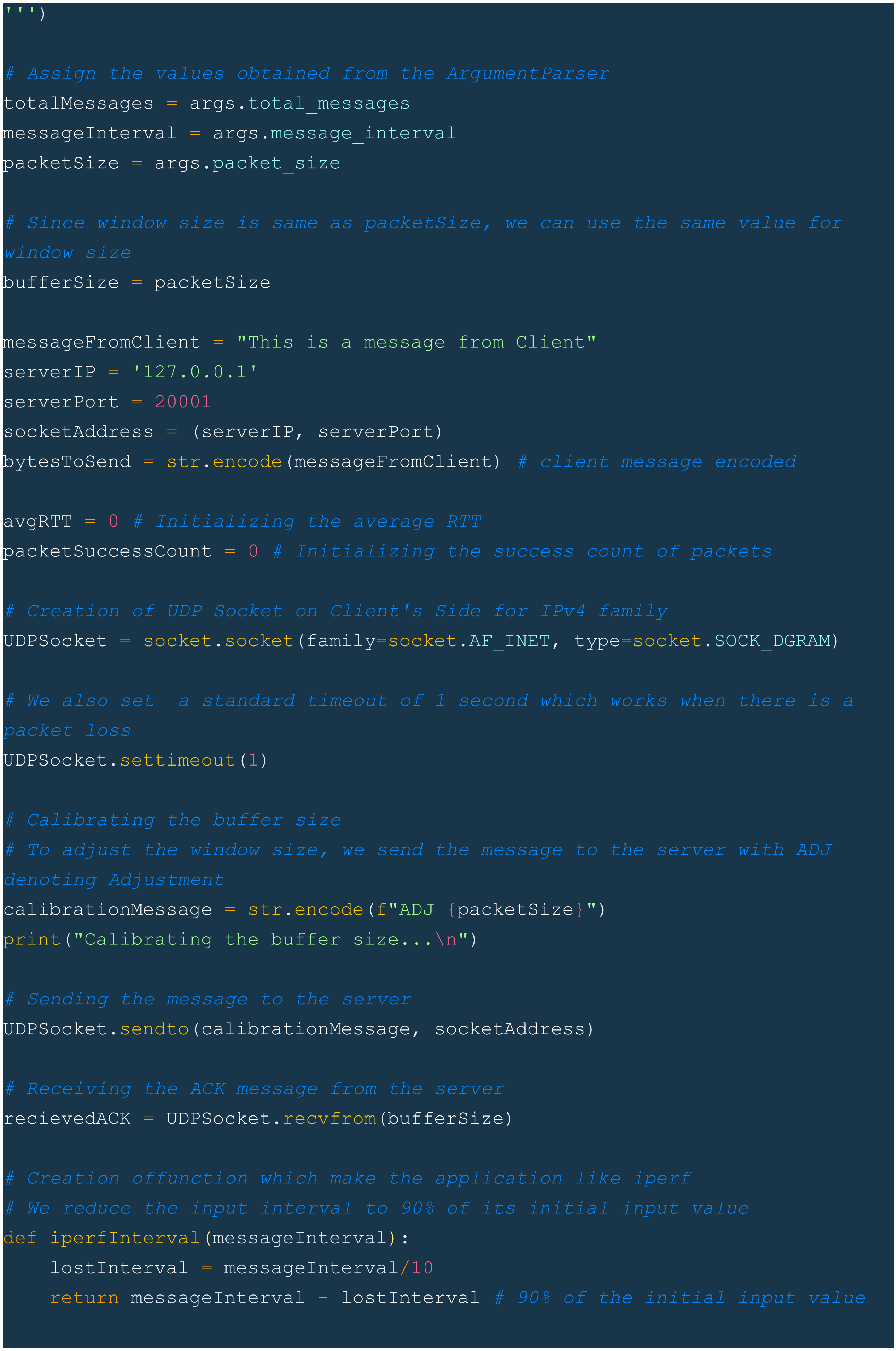
```
Time passed: 13 seconds

Packet is not sent in this second


Average Delay: 0.0 seconds
Average Throughput: 256 bytes/second
```



```
Time passed: 10 seconds

ACK received from the server: Packet Acknowledged!

Average Delay: 0.00012254714965820312 seconds
Average Throughput: 256 bytes/second
```

- Code:
  - Client Implementation:

```
# ***PROBLEM STATEMENT***

# Create an iperf like application using the above
# developed echo client and server program. Reduce the interval
# between two consecutive UDP echo packets generated by client
# to increase the number of echo packets sent from client for a
# given packet size. Calculate the throughput and average delay
# observed every one second. Plot the observed throughput and
# average delay vs time (1 second interval).


# The code for the client starts here
import socket
import sys
import time
import datetime
import argparse
import matplotlib.pyplot as matplotplt


# # Input the required parameters using the normal input method
# totalMessages = int(input("Enter the total number of packets: "))
# messageInterval = float(input("Enter the Interval size: "))
# packetSize = int(input("Enter the Packet size: "))

# Input the required parameters using the argparse method
parser = argparse.ArgumentParser()
parser.add_argument("-n", "--total-number-of-packets", help="Enter the total
number of packets", type=int, dest="total_messages")
parser.add_argument("-i", "--interval", help="Enter the Interval size",
type=float, dest="message_interval")
parser.add_argument("-s", "--size", help="Enter the Packet size", type=int,
dest="packet_size")
args = parser.parse_args()

# Begin with a display message
print('''
**************************************************************
*                       WELCOME!                             *
*                 This is the Client                         *
**************************************************************
```



```
''')

# Assign the values obtained from the ArgumentParser
totalMessages = args.total_messages
messageInterval = args.message_interval
packetSize = args.packet_size

# Since window size is same as packetSize, we can use the same value for
window size
bufferSize = packetSize

messageFromClient = "This is a message from Client"
serverIP = '127.0.0.1'
serverPort = 20001
socketAddress = (serverIP, serverPort)
bytesToSend = str.encode(messageFromClient) # client message encoded

avgRTT = 0 # Initializing the average RTT
packetSuccessCount = 0 # Initializing the success count of packets

# Creation of UDP Socket on Client's Side for IPv4 family
UDPSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

# We also set  a standard timeout of 1 second which works when there is a
packet loss
UDPSocket.settimeout(1)

# Calibrating the buffer size
# To adjust the window size, we send the message to the server with ADJ
denoting Adjustment
calibrationMessage = str.encode(f"ADJ {packetSize}")
print("Calibrating the buffer size...\n")

# Sending the message to the server
UDPSocket.sendto(calibrationMessage, socketAddress)

# Receiving the ACK message from the server
recievedACK = UDPSocket.recvfrom(bufferSize)

# Creation offunction which make the application like iperf
# We reduce the input interval to 90% of its initial input value
def iperfInterval(messageInterval):
    lostInterval = messageInterval/10
    return messageInterval - lostInterval # 90% of the initial input value
```

```
# For plotting the graph of average throughput, we create an array to store
average throughput values for every second
AVERAGEThroughput = []

# For plotting the graph of average throughput, we create an array to store
average delay values for every second
AVERAGEDelay = []

# We also create an array to store the time values for the graph
timeValues = []

secondsPassed = 0 # Initializing the seconds passed
delayDiff = 0 # Initializing the delay difference for storing the delay time
before transmission of a packet in the next second

# Loop begins here for sending the messages to the server iteratively and
increase the time
while totalMessages > 0:

    secondsPassed += 1 # Incrementing the seconds passed
    print("\n              ***\n")
    print(f"Time passed: {secondsPassed} seconds")

    # Variable for storing average delay time
    averageDelayTime = 0

    startTimeStamp = datetime.datetime.now().timestamp()  # Getting the
current time stamp for beginning of the second

    # As long as we don't reach 1 second from the stratTimeStamp, we keep
sending the message
    while datetime.datetime.now().timestamp() - startTimeStamp <=1:

        # If delay differnce exceeds 1, we  sleep for the second and don't
send the message
        if delayDiff > 1:
             goToSleep = 1
             delayDiff -= 1
             print("\nPacket is not sent in this second\n")

        else:

            goToSleep = delayDiff # If delay difference is less than 1, we
send the message
```

```
        # Sleep till the dalay is over
        time.sleep(goToSleep)

        if totalMessages == 0:
            break

        # Stop the transfer for the second if we skiped the whole second
        if goToSleep == 1:
            break

        # Send message to the server
        clientMessage = "Message from the Client"
        UDPSocket.sendto(str.encode(clientMessage), socketAddress)

        # Initialize a variable for storing the sending Timestamp of the
echo message
        sendingTimeStamp = datetime.datetime.now().timestamp()

        #Message is sent so, decrease the total number of messages
        totalMessages -= 1

        # Receive the ACK message from the server
        try:
            recievedACK = UDPSocket.recvfrom(bufferSize)
            recievedACK = recievedACK[0]
            recievedACK = recievedACK.decode()
            print(f"\nACK received from the server: {recievedACK}")

        # If the packet is not recieved from the server
        # When packet loss occurs causing socket timeout
        except socket.timeout:
            print("\nPACKET LOST!!\n")
            continue # Move on to the next packet

        # Store the recieving timestamp of the ACK message
        recievingTimeStamp = datetime.datetime.now().timestamp()

        # Calculate the RTT
        RTT = recievingTimeStamp - sendingTimeStamp

        # Packet was successfully sent so, increment the packet success
count
        packetSuccessCount += 1
        # Average Dealy occurs due to RTT so, increament the average delay between
        averageDelayTime += RTT

        # Define sleepTime for the case when packet arrives earlier than the interval
        # We need to sleep for the interval left to send the next packet
        sleepTime = max(messageInterval - RTT,0)

        # Every iteration, we decrese the interval according to the function we defined earlier to make it like iperfIterval
        messageInterval = iperfInterval(messageInterval)

        passedTime = datetime.datetime.now().timestamp() - startTimeStamp # Calculate the passed time

        # we wait until the interpacket message interval is completed
        # If time isleft, sleep for the remaining time
        if passedTime > 1:
            delayDiff = goToSleep # Store the remaining delay to carry to next second
            break

        elif passedTime + sleepTime > 1 and passedTime <= 1:
            goToSleep = 1-passedTime # Store the sleep value to the rest of the second
            delayDiff = sleepTime - (1 - passedTime) # Store the remaining delay to carry to next second

        else:
            goToSleep = sleepTime # Store the sleep value to the rest of the second

        #Sleep until the delay is complete
        time.sleep(goToSleep)

    # Calculation of the average delay time
    if packetSuccessCount != 0:
        averageDelayTime = averageDelayTime/packetSuccessCount

    else:
        averageDelayTime = 0 # If no packet was successfully sent, average delay is 0

    # Calculation of the average throughput
    # We calculate the average throughput by dividing the total data sent by the total number of seconds
    # SInce we are calculating the average throughput for every second, we don't divide by 1 everytime
    # Since the packet is transfered from client ot server and then back to client, we multiply it by 2
    averageThroughput = (packetSuccessCount*bufferSize)*2

    print(f"\nAverage Delay: {averageDelayTime} seconds")
    print(f"Average Throughput: {averageThroughput} bytes/second\n")

    # Appending the average delay and throughput values to the array
    AVERAGEDelay.append(averageDelayTime)
    AVERAGEThroughput.append(averageThroughput)

    # Since the value of average delay and average throughput remains same for one second, we append both the values twice
    AVERAGEDelay.append(averageDelayTime)
    AVERAGEThroughput.append(averageThroughput)

    timeValues.append(secondsPassed)
    timeValues.append(secondsPassed + 0.9999999999999999) # this time value stays same till the end of the second

# The graph of average throughput and average delay is plotted here
# We create two subgraphs side by side

# 1. Sub Plot for the graph of average throughput
matplotplt.subplot(1,2,1)

# Average throughput is on Y-axis and time is on X-axis
matplotplt.plot(timeValues, AVERAGEThroughput, color = 'blue', label = 'Average Throughput')
matplotplt.xlabel('Time (seconds)')
matplotplt.ylabel('Average Throughput (bytes/second)')
matplotplt.title('Average Throughput Plot')

# 2. Sub Plot for the graph of average delay
matplotplt.subplot(1,2,2)

# Average Delay is on Y-axis and time is on X-axis
matplotplt.plot(timeValues, AVERAGEDelay, color = 'red', label = 'Average Delay')
matplotplt.xlabel('Time (seconds)')
matplotplt.ylabel('Average Delay (seconds)')
matplotplt.title('Average Delay Plot')

matplotplt.show()

print('\n Graph is plotted!')
print('\n Exiting the program and terminating the connection ...\n')

# Send the message to the server to terminate the connection
UDPSocket.sendto(str.encode("TERMINATE"), socketAddress)
serverResponse = UDPSocket.recvfrom(bufferSize)
print(f"Message from the server: Connection Terminated")
print("\n    ------------------------------------")
print("\n")

# TERMINATE the socket
UDPSocket.close()
```

- ○ Server Implementation:

```
# ***PROBLEM STATEMENT***

# Create an iperf like application using the above
# developed echo client and server program. Reduce the interval
# between two consecutive UDP echo packets generated by client
# to increase the number of echo packets sent from client for a
# given packet size. Calculate the throughput and average delay
# observed every one second. Plot the observed throughput and
# average delay vs time (1 second interval).


# The code for the server starts here
import socket
import time
import random
from datetime import datetime

# Start with a display message
print('''
**************************************************************
*                       WELCOME!                             *
*                 This is the server                         *
**************************************************************
''')

serverIP = '127.0.0.1'
serverPort = 20001
socketAddress = (serverIP, serverPort)
serverPublicIP = socket.gethostbyname(socket.gethostname())

bufferSize = 1024

ACKmessage = "Packet Acknowledged!"
encodedACKmessage = ACKmessage.encode()

# Creation of UDP Socket on Client's Side for IPv4 family
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind the socket to the address
serverSocket.bind(socketAddress)

print(f"UDP Server is up and running on address {serverPublicIP} :
{serverPort}")

serverHostname = socket.gethostname()
print(f"Server Hostname: {serverHostname}")

# Listen for incoming connections
while True:
    # Get client requests
    clientRequestData, clientAddress = serverSocket.recvfrom(bufferSize)
    print("\n")
    print(f"Message from the Client: {clientRequestData}")
    print(f"Client Address: {clientAddress}")

    decodedData = clientRequestData.decode()
    print(f"Decoded Data: {decodedData}")

    # If the decoded message required the adjustment of window size
    if decodedData.split()[0] == "ADJ":
        print("Adjusting Window Size...")
        bufferSize = int(decodedData.split(" ")[1])
        print()
        serverSocket.sendto(str.encode(f"Adjusted Window Size to
{bufferSize}"), clientAddress)
        print(f"Adjusted Window size to {bufferSize}")
        print("\n")
        continue

    # If the decoded message requests Termination of Connection
    if decodedData == "TERMINATE":
        print("Terminating Connection...")
        serverSocket.sendto(str.encode("Connection Terminated"),
clientAddress)
        print("\nConnection Terminated!\n")
        break


    # Introducing Packet Delay
    time.sleep(random.randint(1, 5000) / 1000)

    # Introducing Packet Loss with Loss Probability of 0.1
    if random.randint(1, 10) <= 1:
        print("Packet Loss Occurred!\n")
        continue

    # Sending reply to the client with an Acknowledgement message
    serverSocket.sendto(encodedACKmessage, clientAddress)
    print(f"ACK Message: {ACKmessage}")


# Close the socket with a message
print("\n\nClosing the Server Socket...")
serverSocket.close()
print("\nServer Socket Closed!")
```

PART 2: Extend Echo Client/Server and create your own client-server application

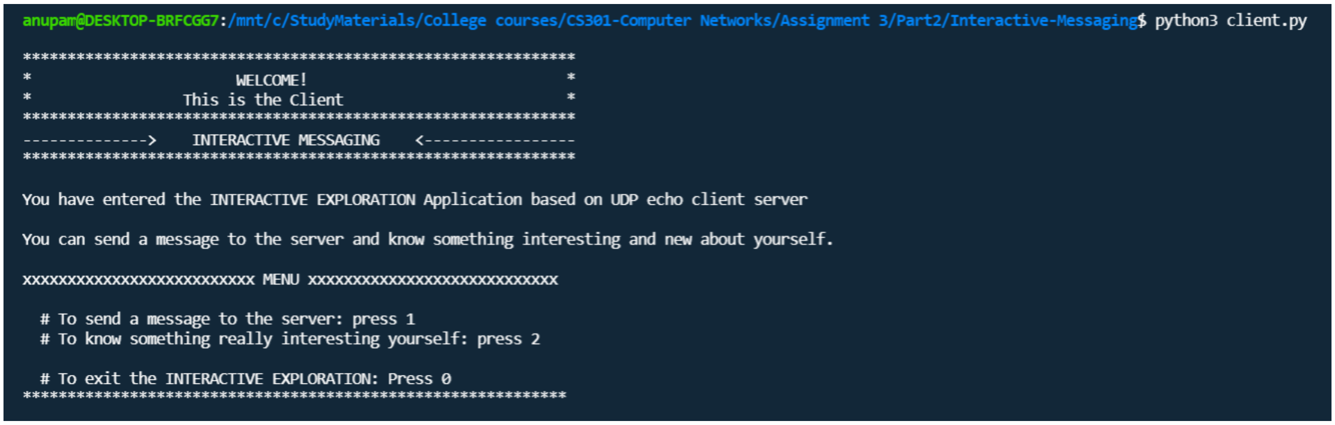
3. The source links: video

- Features Added:
    1. Interactive Messaging
    2. File Viewer

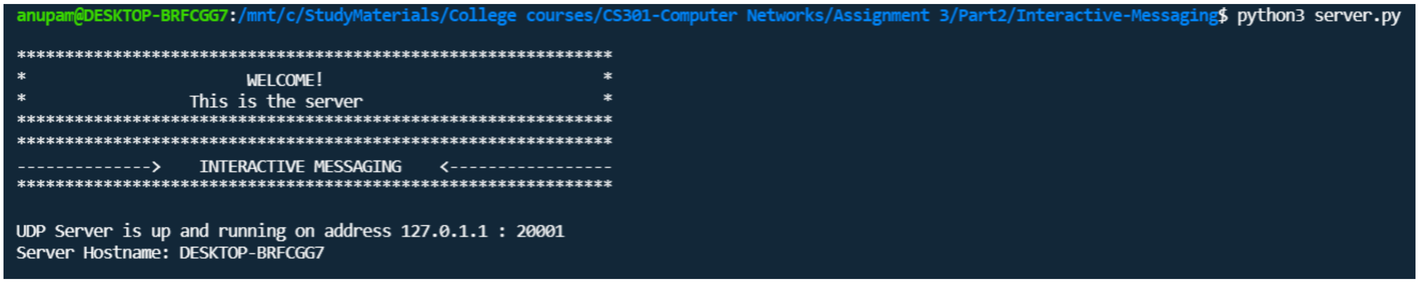
1. Interactive Messaging (client.py, server.py)

- Feature Overview:
    - This is a UDP Echo client-server application that lets the client send the server messages and receive messages from the server.
    - This application also has the feature of taking the date of birth from the user, sending this data to the server, and receiving interesting things based on the date of birth of the client.
    - This is a menu-driven application that iteratively runs and asks for input and interacts with the server to output meaningful results on the client-side.

- How To Run and Explanation:
    - This can simply be run by running the code in server followed by the client by the use of the following commands:
        - Server: python3 server.py
        - Client: python3 client.py

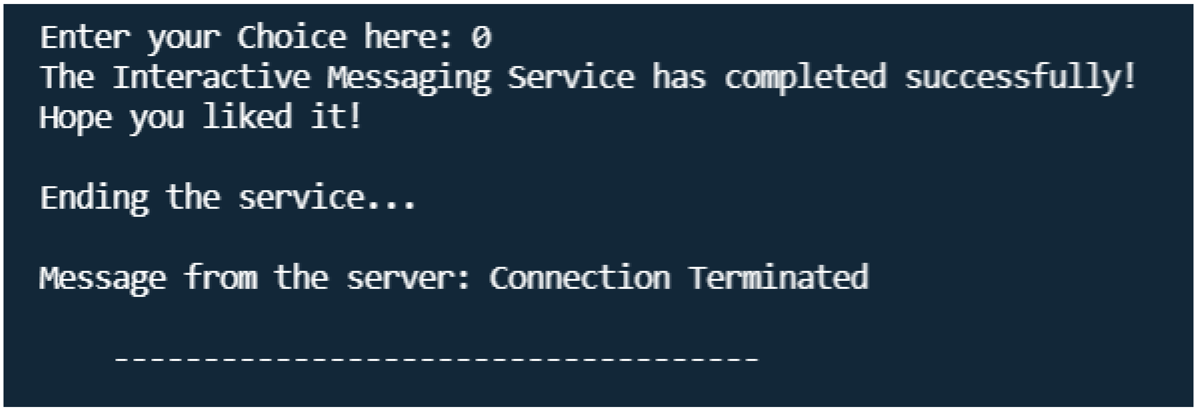


```
anupam@DESKTOP-BRFCGG7:/mnt/c/StudyMaterials/College courses/CS301-Computer Networks/Assignment 3/Part2/Interactive-Messaging$ python3 client.py

**************************************************************
*                      WELCOME!                              *
*                This is the Client                          *
**************************************************************
-------------->    INTERACTIVE MESSAGING    <-----------------
**************************************************************

You have entered the INTERACTIVE EXPLORATION Application based on UDP echo client server

You can send a message to the server and know something interesting and new about yourself.

xxxxxxxxxxxxxxxxxxxxxxxxxx MENU xxxxxxxxxxxxxxxxxxxxxxxxxxxxx

  # To send a message to the server: press 1
  # To know something really interesting yourself: press 2

  # To exit the INTERACTIVE EXPLORATION: Press 0
*************************************************************
```

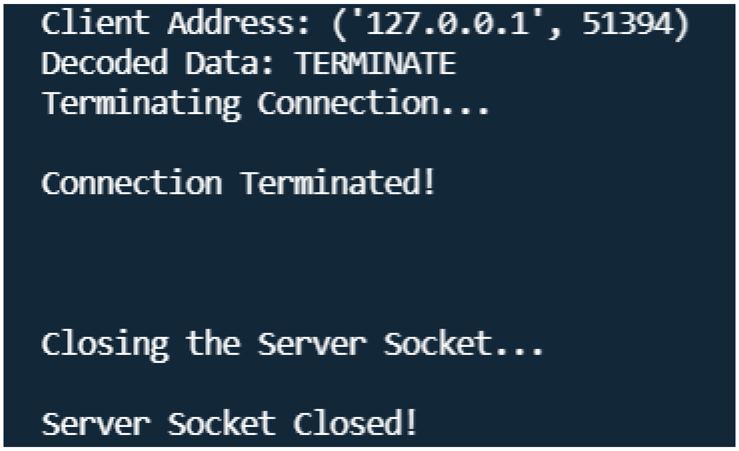


```
anupam@DESKTOP-BRFCGG7:/mnt/c/StudyMaterials/College courses/CS301-Computer Networks/Assignment 3/Part2/Interactive-Messaging$ python3 server.py

**************************************************************
*                       WELCOME!                             *
*                 This is the server                         *
**************************************************************
**************************************************************
-------------->    INTERACTIVE MESSAGING    <-----------------
**************************************************************

UDP Server is up and running on address 127.0.1.1 : 20001
Server Hostname: DESKTOP-BRFCGG7
```

- Termination (client, server):



```
Enter your Choice here: 0
The Interactive Messaging Service has completed successfully!
Hope you liked it!

Ending the service...

Message from the server: Connection Terminated

    -----------------------------------
```



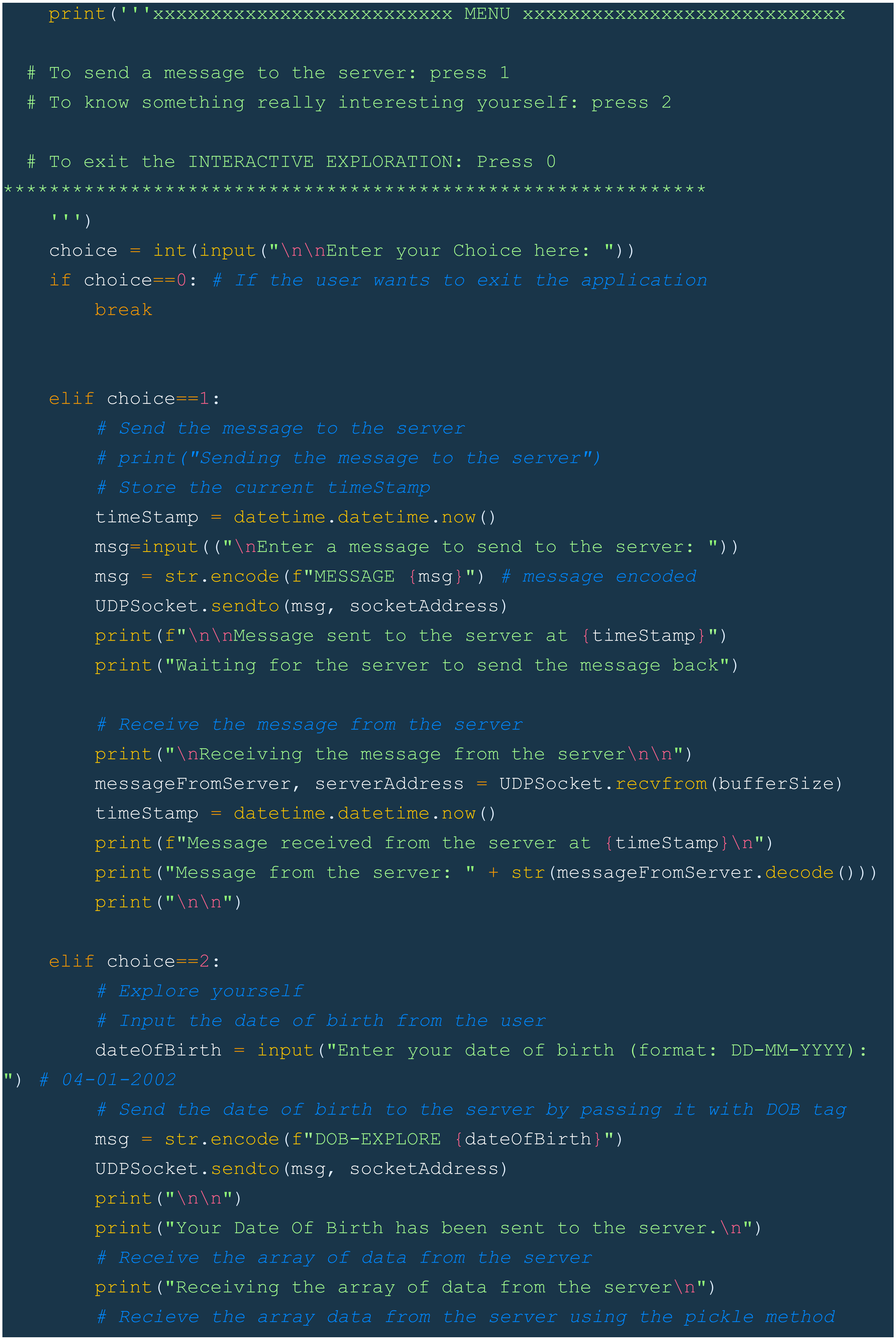
```
Client Address: ('127.0.0.1', 51394)
Decoded Data: TERMINATE
Terminating Connection...

Connection Terminated!



Closing the Server Socket...

Server Socket Closed!
```

- Further Explanation is available in the video explanation and code comments

- Code:

  - Client Implementation:

```
# ***PROBLEM STATEMENT***

# Add any two features to Echo Client/Server and demonstrate
# them. In the report, you must describe the new features with their
# benefit.

# This is one of the features that I have incorporated
# INTERACTIVE MESSAGING in Echo client server


# The code for the client starts here:
import socket
import datetime
import pickle



# Begin with a display message
print('''
**************************************************************
*                       WELCOME!                             *
*                 This is the Client                         *
**************************************************************
-------------->    INTERACTIVE MESSAGING    <-----------------
**************************************************************
''')


# Initialize the buffer size to 1024
bufferSize = 1024

messageFromClient = "This is a message from Client"
serverIP = '127.0.0.1'
serverPort = 20001
socketAddress = (serverIP, serverPort)
bytesToSend = str.encode(messageFromClient) # client message encoded

# Creation of UDP Socket on Client's Side for IPv4 family
UDPSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

# Timeout setting to interval
UDPSocket.settimeout(1024)

# packetCount = 0

print("You have entered the INTERACTIVE EXPLORATION Application based on UDP
echo client server\n")
print("You can send a message to the server and know something interesting
and new about yourself.\n")

# Get in the infinite loop until the user asks to leave
while True:
```



```
    print('''xxxxxxxxxxxxxxxxxxxxxxxxxx MENU xxxxxxxxxxxxxxxxxxxxxxxxxxxx

  # To send a message to the server: press 1
  # To know something really interesting yourself: press 2

  # To exit the INTERACTIVE EXPLORATION: Press 0
*************************************************************
    ''')
    choice = int(input("\n\nEnter your Choice here: "))
    if choice==0: # If the user wants to exit the application
        break


    elif choice==1:
        # Send the message to the server
        # print("Sending the message to the server")
        # Store the current timeStamp
        timeStamp = datetime.datetime.now()
        msg=input(("\nEnter a message to send to the server: "))
        msg = str.encode(f"MESSAGE {msg}") # message encoded
        UDPSocket.sendto(msg, socketAddress)
        print(f"\n\nMessage sent to the server at {timeStamp}")
        print("Waiting for the server to send the message back")

        # Receive the message from the server
        print("\nReceiving the message from the server\n\n")
        messageFromServer, serverAddress = UDPSocket.recvfrom(bufferSize)
        timeStamp = datetime.datetime.now()
        print(f"Message received from the server at {timeStamp}\n")
        print("Message from the server: " + str(messageFromServer.decode()))
        print("\n\n")

    elif choice==2:
        # Explore yourself
        # Input the date of birth from the user
        dateOfBirth = input("Enter your date of birth (format: DD-MM-YYYY):
") # 04-01-2002
        # Send the date of birth to the server by passing it with DOB tag
        msg = str.encode(f"DOB-EXPLORE {dateOfBirth}")
        UDPSocket.sendto(msg, socketAddress)
        print("\n\n")
        print("Your Date Of Birth has been sent to the server.\n")
        # Receive the array of data from the server
        print("Receiving the array of data from the server\n")
        # Recieve the array data from the server using the pickle method
        dataRecieved, serverAddress = UDPSocket.recvfrom(bufferSize)
        dataFromServer = pickle.loads(dataRecieved) # deserialize the data
stream using the loads method
        # dataFromServer, serverAddress = UDPSocket.recvfrom(bufferSize)
        age = dataFromServer[0]
        greeting = dataFromServer[1]
        astrologicalSign = dataFromServer[2]
        personality = dataFromServer[3]
        print(f"\nThe message from the server is:\n")
        print(f"{greeting} client!\n\nHere are the informations about you
which you might not have known:-\nYour Age is {age}\nYour Astrological Sign
is {astrologicalSign}\n{personality}\n\n")


    else: # If the user has entered an invalid choice
        print("\nYou have entered an invalid input!\n")
        print("\nPlease enter a valid choice\nRedirecting to the main
page...\n")
        continue


# Sending of messages completed.
print("The Interactive Messaging Service has completed successfully!\nHope
you liked it!\n")

print("Ending the service...\n")
# Send termination message to the server
UDPSocket.sendto(str.encode("TERMINATE"), socketAddress)
serverResponse = UDPSocket.recvfrom(bufferSize)
print(f"Message from the server: Connection Terminated")

print("\n    ------------------------------------\n")

# TERMINATE the socket
UDPSocket.close()
```

- ○ Server Implementation:

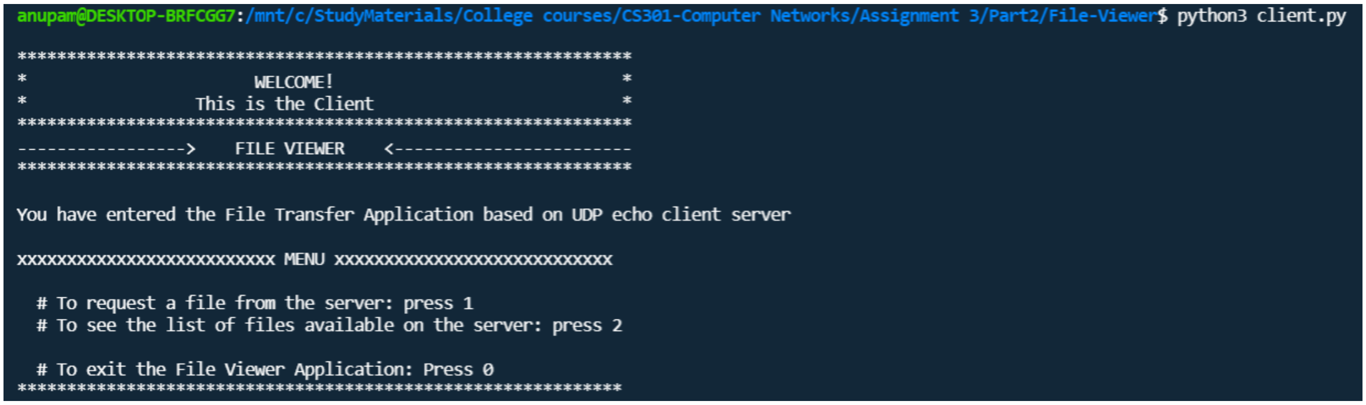
```python
# ***PROBLEM STATEMENT***

# Add any two features to Echo Client/Server and demonstrate
# them. In the report, you must describe the new features with their
# benefit.

# This is one of the features that I have incorporated
# INTERACTIVE MESSAGING in Echo client server

# The code for the server starts here
import socket

# The pickle module implements binary protocols for serializing and
de-serializing a Python object structure.
import pickle
import random
import datetime

# Start with a display message
print('''
**************************************************************
*                       WELCOME!                             *
*                 This is the server                         *
**************************************************************
**************************************************************
-------------->    INTERACTIVE MESSAGING    <-----------------
**************************************************************
''')

serverIP = '127.0.0.1'
serverPort = 20001
socketAddress = (serverIP, serverPort)
serverPublicIP = socket.gethostbyname(socket.gethostname())

bufferSize = 1024

ACKmessage = "Packet Acknowledged!"
encodedACKmessage = ACKmessage.encode()

# Creation of UDP Socket on Client's Side for IPv4 family
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind the socket to the address
```

```
serverSocket.bind(socketAddress)

print(f"UDP Server is up and running on address {serverPublicIP} :
{serverPort}")

serverHostname = socket.gethostname()
print(f"Server Hostname: {serverHostname}")

# Listen for incoming connections
while True:
    # Get client requests
    clientRequestData, clientAddress = serverSocket.recvfrom(bufferSize)
    print("\n")
    print(f"Message from the Client: {clientRequestData}")
    print(f"Client Address: {clientAddress}")

    decodedData = clientRequestData.decode()
    print(f"Decoded Data: {decodedData}")

    # If the decoded message required the adjustment of window size
    if decodedData.split()[0] == "ADJ":
        print("Adjusting Window Size...")
        bufferSize = int(decodedData.split(" ")[1])
        print()
        serverSocket.sendto(str.encode(f"Adjusted Window Size to
{bufferSize}"), clientAddress)
        print(f"Adjusted Window size to {bufferSize}")
        print("\n")
        continue

    # If the decoded message requests Termination of Connection
    if decodedData == "TERMINATE":
        print("Terminating Connection...")
        serverSocket.sendto(str.encode("Connection Terminated"),
clientAddress)
        print("\nConnection Terminated!\n")
        break

    # If the decoded message requests the server to send a random message to
the client when decodedData begins with MESSAGE
    if decodedData.split()[0] == "MESSAGE":
        # Recieve the message from the client
        message = decodedData.split()[1]
        # store the current timestamp
        timestamp = datetime.datetime.now()
        print(f"\nMessage from the Client: {message}\nReceived at 
{timestamp}\n")
        print("\nSending Random Message...")
        # generate a random message comprising greeting words to form a 
message
        randomMessage = random.choice(["Hello", "Hi", "Hey", "Howdy", 
"Greetings", "Good Day", "Good Morning", "Good Evening", "Good Afternoon", 
"Good Night"])
        # send the random message to the client
        serverSocket.sendto(str.encode(f"Your Message has been recieved\n 
Here is a Message for you: {randomMessage}"), clientAddress)
        # update the timestamp
        timestamp = datetime.datetime.now()
        print(f"Random Message: {randomMessage}")
        print(f"\nMessage sent at {timestamp}")
        print("\n")
        continue


    # If the decoded message requests the server for age based on date of 
birth when the decoded message begins with DOB
    if decodedData.split()[0] == "DOB-EXPLORE":
        # Recieve the date of birth from the client
        dateOfBirth = decodedData.split()[1] #04-01-2002
        # store the current timestamp
        timestamp = datetime.datetime.now()
        print(f"\nDate of Birth: {dateOfBirth}\nReceived at {timestamp}\n")
        print("\nSending Age...")

        # calculate the age in years
        age = datetime.datetime.now().year - int(dateOfBirth.split("-")[2]) # 
[04,01,2002]
        # Print the age of the client
        print(f"\nAge of the Client: {age}\n")
        timestamp = datetime.datetime.now()
        # Generate a greeting message based on current time like Good 
Morning, Good Afternoon, Good Evening, Good Night
        greetingTime = timestamp.hour # get the current hour
        if greetingTime >= 0 and greetingTime < 12:
            greeting = "Good Morning"
        elif greetingTime >= 12 and greetingTime < 16:
            greeting = "Good Afternoon"
        elif greetingTime >= 16 and greetingTime < 20:
            greeting = "Good Evening"
        else:
            greeting = "Good Night"

        # Get astrological sign based on the date of birth
        dateOfBirth = dateOfBirth.split("-") # split the date of birth into
day, month and year
        day = int(dateOfBirth[0])
        month = int(dateOfBirth[1])
        year = int(dateOfBirth[2])
        if month == 1 or month == 2: # January and February
            month = 12
            year = year - 1 # decrement the year
        else:
            month = month - 1
        # Calculate the day of the week
        dayOfTheWeek = (day + (((13 * month) - 1) / 5) + year + (year / 4) +
(6 * (year / 100)) + (year / 400)) % 7
        # Get the astrological sign based on the day of the week
        if dayOfTheWeek == 0:
            astrologicalSign = "Capricorn"
        elif dayOfTheWeek == 1:
            astrologicalSign = "Aquarius"
        elif dayOfTheWeek == 2:
            astrologicalSign = "Pisces"
        elif dayOfTheWeek == 3:
            astrologicalSign = "Aries"
        elif dayOfTheWeek == 4:
            astrologicalSign = "Taurus"
        elif dayOfTheWeek == 5:
            astrologicalSign = "Gemini"
        else:
            astrologicalSign = "Cancer"

        # Find the personality based on the date of birth and the
astrological sign
        if day >= 21:
            personality = "You are an Achiever"
        elif day >= 19:
            personality = "You are a Socializer"
        elif day >= 17:
            personality = "You are a Thinker"
        elif day >= 15:
            personality = "You are an Explorer"
        elif day >= 13:
            personality = "You are an Entertainer"
        elif day >= 11:
            personality = "You are an Analyst"
        elif day >= 9:
            personality = "You are a Leader"
        elif day >= 7:
            personality = "You are an Analyst"
        elif day >= 5:
            personality = "You are an Entertainer"
        elif day >= 3:
            personality = "You are a Thinker"
        else:
            personality = "You are a Socializer"
        # Send the age and personality to the client
        # create an array and all the value of age, greeting, astrological
sign and personality
        clientInfo = [age, greeting, astrologicalSign, personality]
        # send the array to the client
        serverSocket.sendto(pickle.dumps(clientInfo), clientAddress) #
searilize the array and send it to the client
        # serverSocket.sendto(str.encode(f"{greeting} client!\n\n Here are
the informations about you which you might not have known:-\nYour Age is
{age}\nYour Astrological Sign is {astrologicalSign}\n\n {personality}"),
clientAddress)
        print(f"\nMessage sent at {timestamp}")
        print("\n")
        timestamp = datetime.datetime.now()
        print(f"\nGreeting: {greeting}\nAge: {age}\nAstrological Sign:
{astrologicalSign}\nPersonality: {personality}\n")
        print(f"\nMessage sent at {timestamp}")
        print("\n")
        continue



    # Sending reply to the client with an Acknowledgement message
    serverSocket.sendto(encodedACKmessage, clientAddress)
    print(f"ACK Message: {ACKmessage}")


# Close the socket with a message
print("\n\nClosing the Server Socket...")
serverSocket.close()
print("\nServer Socket Closed!")
```

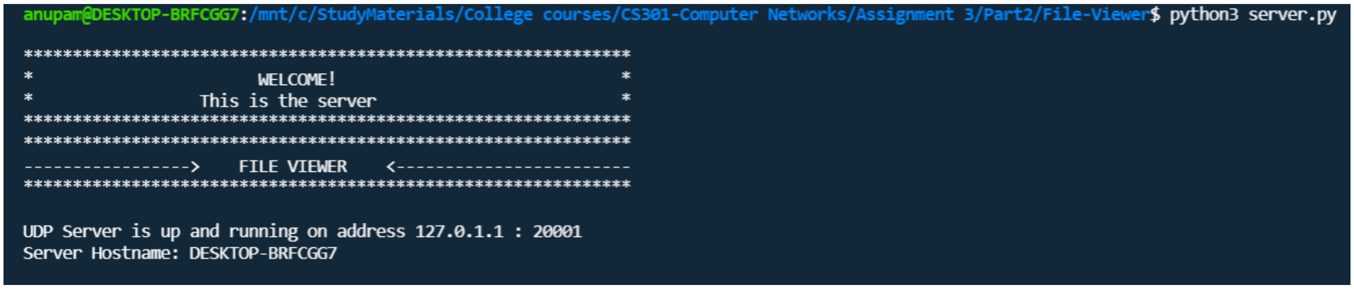
2. File Viewer (client.py, server.py)

- Feature Overview:
    - This is a UDP Echo client-server application that lets the client see the list of files available on the server-side and also view the files available in the server.
    - This application takes messages from the user on the client-side and returns output according to the request by interacting with the server.
    - This is a menu-driven application that iteratively runs and asks for input and interacts with the server to output meaningful results on the client-side.

- How To Run and Explanation:
    - This can simply be run by running the code in server followed by the client by the use of the following commands:
        - Server: python3 server.py
        - Client: python3 client.py

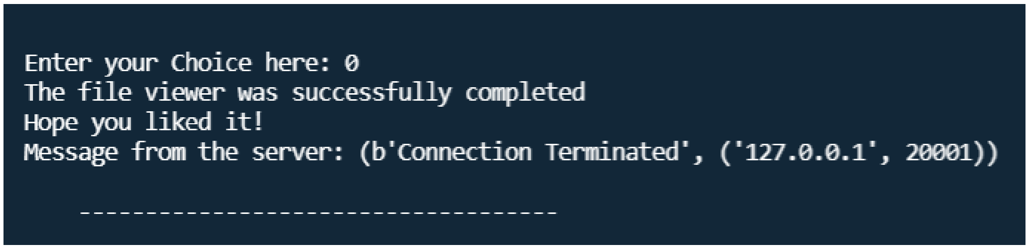


```
anupam@DESKTOP-BRFCGG7:/mnt/c/StudyMaterials/College courses/CS301-Computer Networks/Assignment 3/Part2/File-Viewer$ python3 client.py

***************************************************************
*                       WELCOME!                              *
*                 This is the Client                          *
***************************************************************
----------------->    FILE VIEWER    <------------------------
***************************************************************

You have entered the File Transfer Application based on UDP echo client server

xxxxxxxxxxxxxxxxxxxxxxxxxx MENU xxxxxxxxxxxxxxxxxxxxxxxxxxxx

  # To request a file from the server: press 1
  # To see the list of files available on the server: press 2

  # To exit the File Viewer Application: Press 0
**************************************************************
```

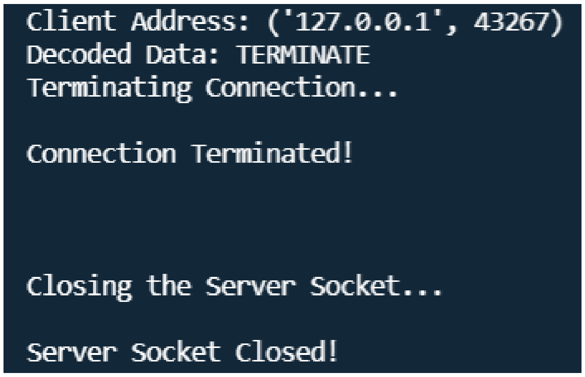


```
anupam@DESKTOP-BRFCGG7:/mnt/c/StudyMaterials/College courses/CS301-Computer Networks/Assignment 3/Part2/File-Viewer$ python3 server.py

***************************************************************
*                       WELCOME!                              *
*                 This is the server                          *
***************************************************************
***************************************************************
----------------->    FILE VIEWER    <------------------------
***************************************************************

UDP Server is up and running on address 127.0.1.1 : 20001
Server Hostname: DESKTOP-BRFCGG7
```

    - Termination (client, server):



```
Enter your Choice here: 0
The file viewer was successfully completed
Hope you liked it!
Message from the server: (b'Connection Terminated', ('127.0.0.1', 20001))

    ------------------------------------
```

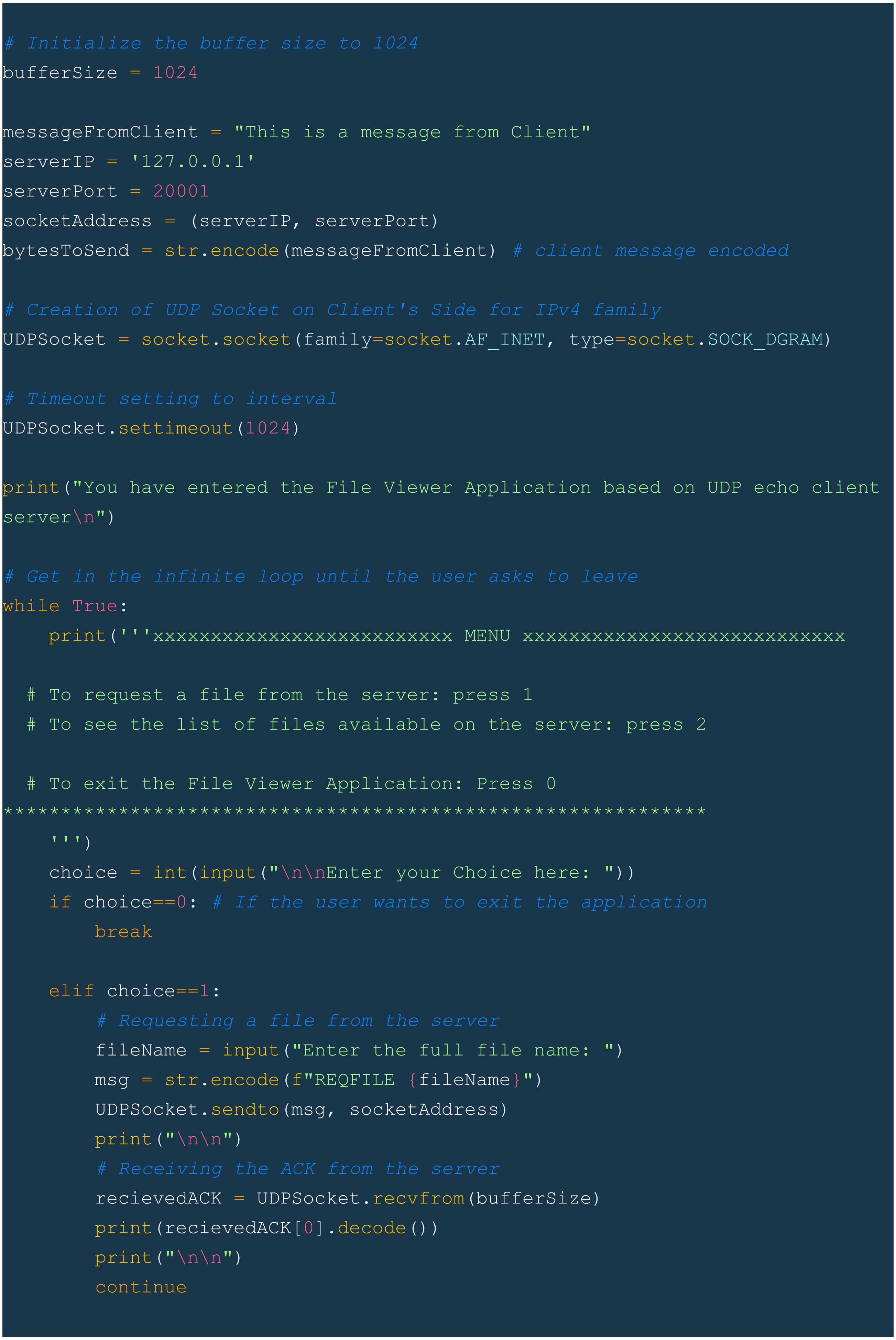


```
Client Address: ('127.0.0.1', 43267)
Decoded Data: TERMINATE
Terminating Connection...

Connection Terminated!



Closing the Server Socket...

Server Socket Closed!
```

- Further Explanation is available in the video explanation and code comments

- Code:
  - Client Implementation:

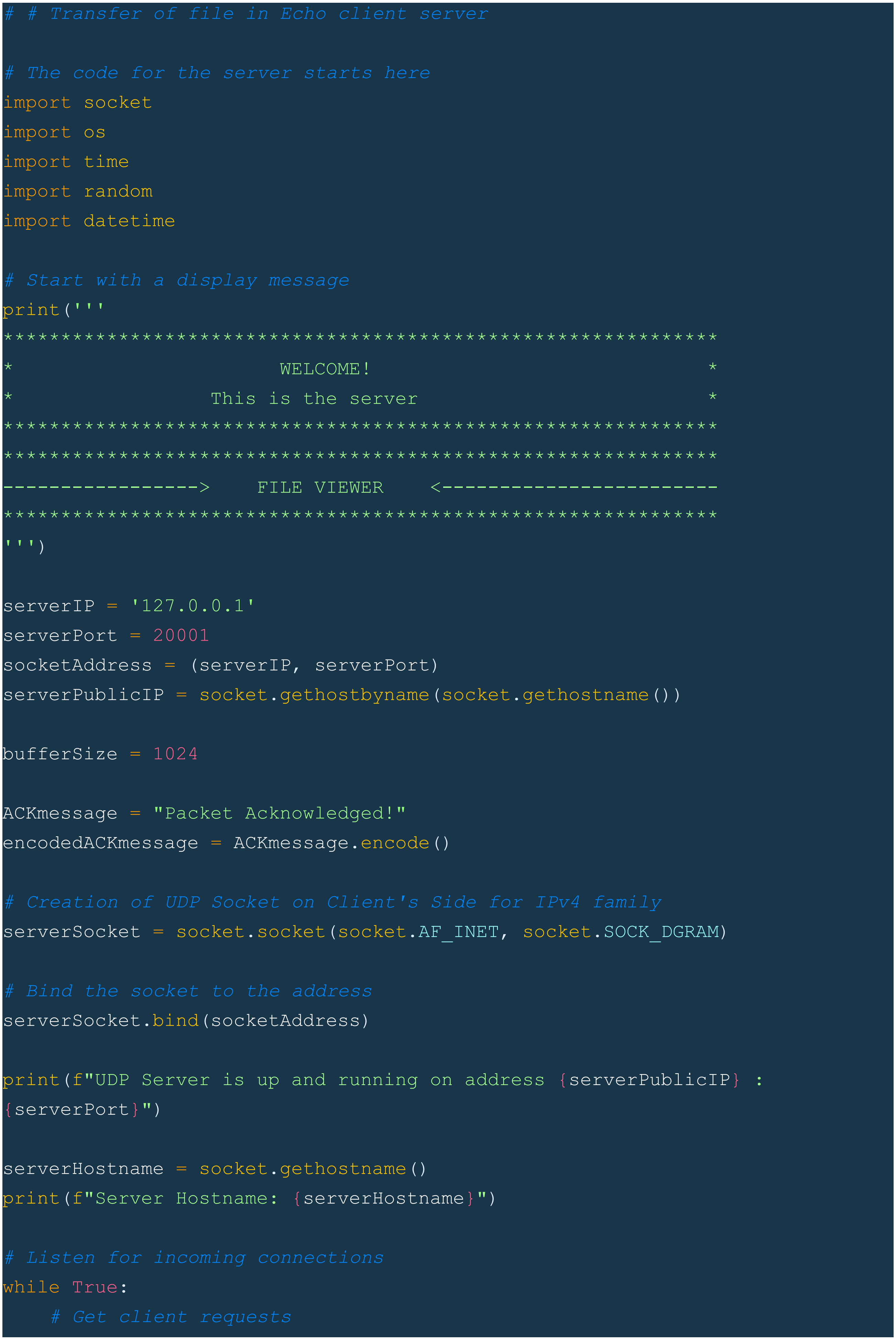
```
# ***PROBLEM STATEMENT***

# Add any two features to Echo Client/Server and demonstrate
# them. In the report, you must describe the new features with their
# benefit.

# This is on of the features that I have incorporated
# Transfer of file in Echo client server


# The code for the client starts here:
import socket




# Begin with a display message
print('''
****************************************************************
*                        WELCOME!                              *
*                  This is the Client                          *
****************************************************************
------------------>    FILE VIEWER    <-------------------------
****************************************************************
''')
```



```
# Initialize the buffer size to 1024
bufferSize = 1024

messageFromClient = "This is a message from Client"
serverIP = '127.0.0.1'
serverPort = 20001
socketAddress = (serverIP, serverPort)
bytesToSend = str.encode(messageFromClient) # client message encoded

# Creation of UDP Socket on Client's Side for IPv4 family
UDPSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

# Timeout setting to interval
UDPSocket.settimeout(1024)

print("You have entered the File Viewer Application based on UDP echo client
server\n")

# Get in the infinite loop until the user asks to leave
while True:
    print('''xxxxxxxxxxxxxxxxxxxxxxxxxx MENU xxxxxxxxxxxxxxxxxxxxxxxxxxxx

  # To request a file from the server: press 1
  # To see the list of files available on the server: press 2

  # To exit the File Viewer Application: Press 0
*************************************************************
    ''')
    choice = int(input("\n\nEnter your Choice here: "))
    if choice==0: # If the user wants to exit the application
        break

    elif choice==1:
        # Requesting a file from the server
        fileName = input("Enter the full file name: ")
        msg = str.encode(f"REQFILE {fileName}")
        UDPSocket.sendto(msg, socketAddress)
        print("\n\n")
        # Receiving the ACK from the server
        recievedACK = UDPSocket.recvfrom(bufferSize)
        print(recievedACK[0].decode())
        print("\n\n")
        continue
    elif choice==2:
        # Requesting the list of files from the server
        msg = str.encode(f"LIST")
        UDPSocket.sendto(msg, socketAddress)
        print("\n\n")
        # Receiving the ACK from the server
        recievedACK = UDPSocket.recvfrom(bufferSize)
        print(recievedACK[0].decode())
        print("\n\n")
        continue

    else: # If the user has entered an invalid choice
        print("\nYou have entered an invalid input!\n")
        print("\nPlease enter a valid choice\nRedirecting to the main page...\n")
        continue


# Sending of messages completed.
print("The file viewer was successfully completed\nHope you liked it!")

# Send termination message to the server
UDPSocket.sendto(str.encode("TERMINATE"), socketAddress)
serverResponse = UDPSocket.recvfrom(bufferSize)
print(f"Message from the server: {serverResponse}")

print("\n    ------------------------------------\n")

# TERMINATE the socket
UDPSocket.close()
```

- Server Implementation:

```
# ***PROBLEM STATEMENT***

# Add any two features to Echo Client/Server and demonstrate
# them. In the report, you must describe the new features with their
# benefit.

# This is on of the features that I have incorporated
```



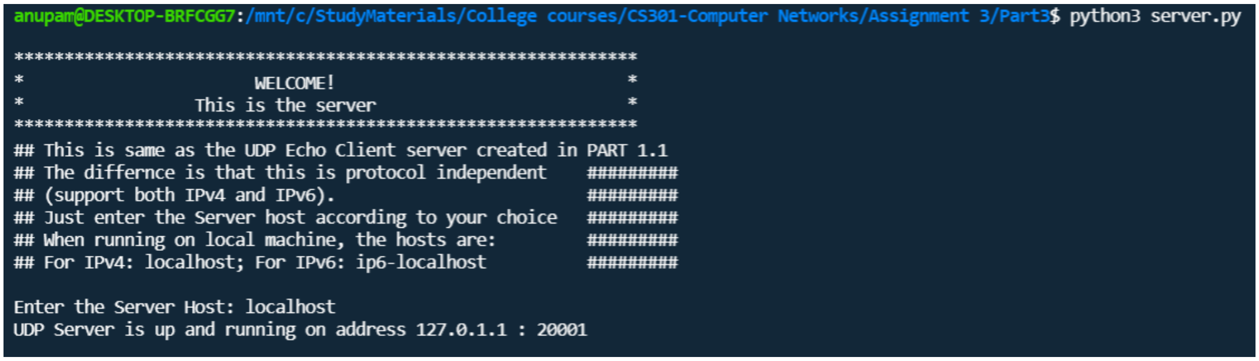
```
# # Transfer of file in Echo client server

# The code for the server starts here
import socket
import os
import time
import random
import datetime

# Start with a display message
print('''
**************************************************************
*                       WELCOME!                             *
*                 This is the server                         *
**************************************************************
**************************************************************
------------------>    FILE VIEWER    <------------------------
**************************************************************
''')

serverIP = '127.0.0.1'
serverPort = 20001
socketAddress = (serverIP, serverPort)
serverPublicIP = socket.gethostbyname(socket.gethostname())

bufferSize = 1024

ACKmessage = "Packet Acknowledged!"
encodedACKmessage = ACKmessage.encode()

# Creation of UDP Socket on Client's Side for IPv4 family
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind the socket to the address
serverSocket.bind(socketAddress)

print(f"UDP Server is up and running on address {serverPublicIP} :
{serverPort}")

serverHostname = socket.gethostname()
print(f"Server Hostname: {serverHostname}")

# Listen for incoming connections
while True:
    # Get client requests
    clientRequestData, clientAddress = serverSocket.recvfrom(bufferSize)
    print("\n")
    print(f"Message from the Client: {clientRequestData}")
    print(f"Client Address: {clientAddress}")

    decodedData = clientRequestData.decode()
    print(f"Decoded Data: {decodedData}")

    # If the decoded message required the adjustment of window size
    if decodedData.split()[0] == "ADJ":
        print("Adjusting Window Size...")
        bufferSize = int(decodedData.split(" ")[1])
        print()
        serverSocket.sendto(str.encode(f"Adjusted Window Size to 
{bufferSize}"), clientAddress)
        print(f"Adjusted Window size to {bufferSize}")
        print("\n")
        continue

    # If the decoded message requests Termination of Connection
    if decodedData == "TERMINATE":
        print("Terminating Connection...")
        serverSocket.sendto(str.encode("Connection Terminated"), 
clientAddress)
        print("\nConnection Terminated!\n")
        break

    # If the decoded message requests the server to send a list of files 
available in the server
    if decodedData == "LIST":
        print("Requesting the list of files from the server...")
        # Get the list of files available in the server
        listOfFiles = [f for f in os.listdir('.') if os.path.isfile(f)]

        # Remove the unnecessary or private files from the list
        listOfFiles.remove('server.py') # Remove the server.py file from the 
list
        listOfFiles.remove('client.py') # Remove the client.py file from the 
list

        print(f"Files in the Server: {listOfFiles}")
        # Send the list of files available in the server to the client
        serverSocket.sendto(str.encode(f"Files available in the Server: 
{listOfFiles}"), clientAddress)
        print("\n")
        continue



    # If the decoded message requests the server for file viewing
    if decodedData.split()[0] == "REQFILE":
        print("Sending File...")
        nameOfFile = decodedData.split()[1] # Get the name of the file requested by the client
        file = open(nameOfFile, "rb")  # Open the file in binary mode(r-read, b-binary) binary mode is useful for reading images and other files
        dataInFile = file.read() # Read the data from the file
        file.close() # Close the file
        serverSocket.sendto(dataInFile, clientAddress) # Send the data to the client
        print(f"File Sent: {nameOfFile}") # Print the name of the file sent
        print("\n")
        continue



    # Sending reply to the client with an Acknowledgement message
    serverSocket.sendto(encodedACKmessage, clientAddress)
    print(f"ACK Message: {ACKmessage}")


# Close the socket with a message
print("\n\nClosing the Server Socket...")
serverSocket.close()
print("\nServer Socket Closed!")
```

PART 3: Making Echo Client/Server “protocol Independent”

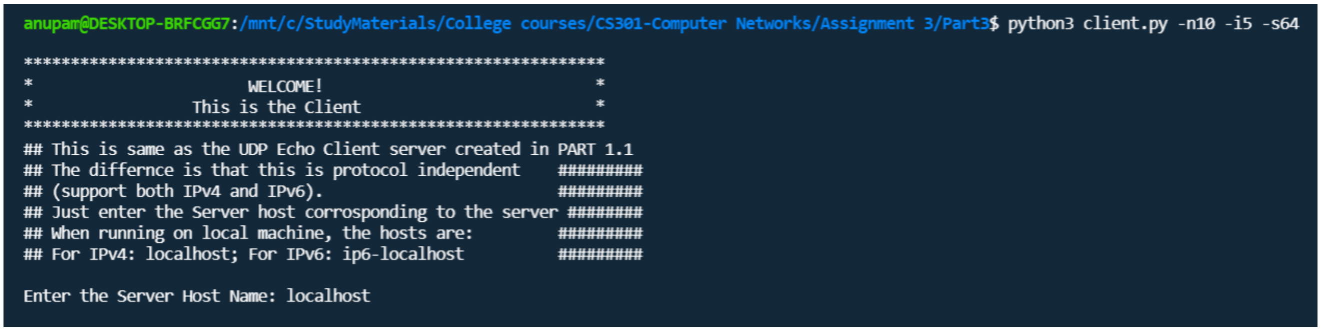
4. The source links: client.py, server.py, video

- Process Overview:
  - I have created the same UDP echo client-server as we have seen in Part 1.1
  - The difference in this is that this is protocol-independent(works for both IPv4 and IPv6).
  - The other parts are the same as Part 1.1 so, I have calculated and displayed the Average RTT and Average Loss at the end.
  - To make the server protocol independent, I have used the **getaddrinfo()** function which takes in the hostname and returns a list of tuples which is used to obtain its family and type. (Further explanation in video and code)

- How To Run and Explanation:
  - Since the base code is the same as Part 1.1, the input-taking process is the same as that.
  - The additional input here is that we have to enter the server host in server and client as shown below (first run the server then, the client):

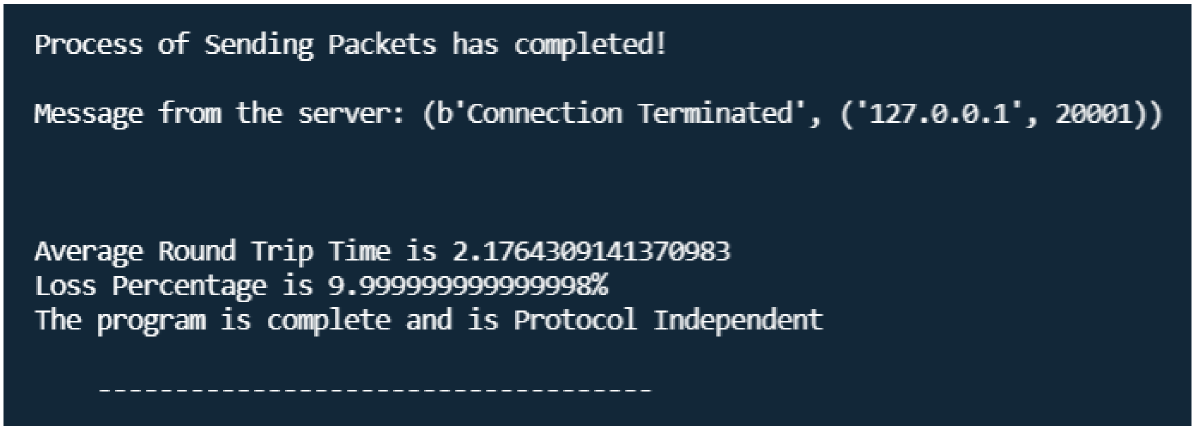


```
anupam@DESKTOP-BRFCGG7:/mnt/c/StudyMaterials/College courses/CS301-Computer Networks/Assignment 3/Part3$ python3 server.py

**************************************************************
*                       WELCOME!                             *
*                 This is the server                         *
**************************************************************
## This is same as the UDP Echo Client server created in PART 1.1
## The differnce is that this is protocol independent    #########
## (support both IPv4 and IPv6).                         #########
## Just enter the Server host according to your choice   #########
## When running on local machine, the hosts are:         #########
## For IPv4: localhost; For IPv6: ip6-localhost          #########

Enter the Server Host: localhost
UDP Server is up and running on address 127.0.1.1 : 20001
```

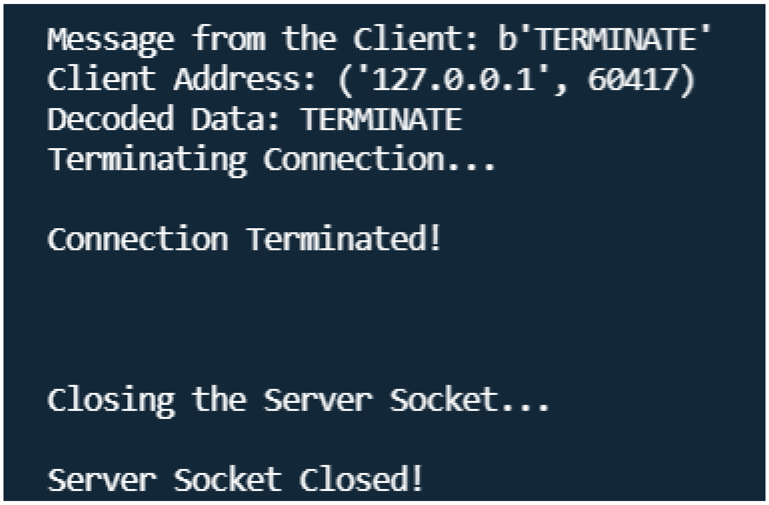


```
anupam@DESKTOP-BRFCGG7:/mnt/c/StudyMaterials/College courses/CS301-Computer Networks/Assignment 3/Part3$ python3 client.py -n10 -i5 -s64

**************************************************************
*                       WELCOME!                             *
*                 This is the Client                         *
**************************************************************
## This is same as the UDP Echo Client server created in PART 1.1
## The differnce is that this is protocol independent    #########
## (support both IPv4 and IPv6).                         #########
## Just enter the Server host corrosponding to the server ########
## When running on local machine, the hosts are:         #########
## For IPv4: localhost; For IPv6: ip6-localhost          #########

Enter the Server Host Name: localhost
```

  - The other parts are the same as Part 1.1 and work the same as that.
  - The termination is shown below (client and server):



```
Process of Sending Packets has completed!

Message from the server: (b'Connection Terminated', ('127.0.0.1', 20001))



Average Round Trip Time is 2.1764309141370983
Loss Percentage is 9.999999999999998%
The program is complete and is Protocol Independent

    ------------------------------------
```

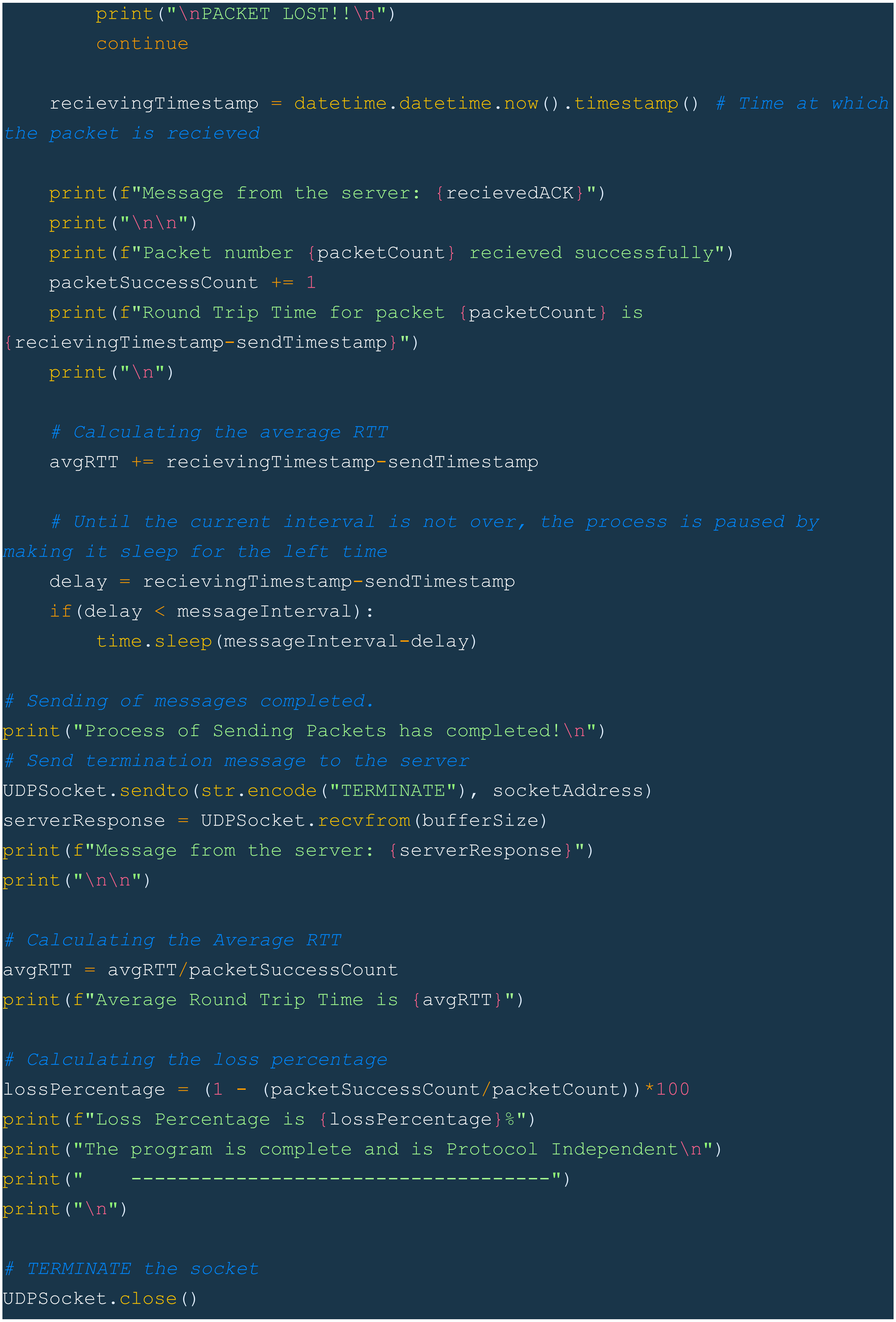


```
Message from the Client: b'TERMINATE'
Client Address: ('127.0.0.1', 60417)
Decoded Data: TERMINATE
Terminating Connection...

Connection Terminated!



Closing the Server Socket...

Server Socket Closed!
```

- Code:
    - Client Implementation:

```
# ***PROBLEM STATEMENT***

# Revise echo client and server to be protocol independent
# (support both IPv4 and IPv6).



# The code for the client starts here:
import socket
import sys
import time
import datetime
import argparse

# # Input the required parameters using the normal input method
# totalMessages = int(input("Enter the total number of packets: "))
# messageInterval = float(input("Enter the Interval size: "))
# packetSize = int(input("Enter the Packet size: "))

# Input the required parameters using the argparse method
parser = argparse.ArgumentParser()
parser.add_argument("-n", "--total-number-of-packets", help="Enter the total number of packets", type=int, dest="total_messages")
parser.add_argument("-i", "--interval", help="Enter the Interval size", type=float, dest="message_interval")
parser.add_argument("-s", "--size", help="Enter the Packet size", type=int, dest="packet_size")
args = parser.parse_args()

# Begin with a display message
print('''
**************************************************************
*                       WELCOME!                            *
*                 This is the Client                         *
**************************************************************
## This is same as the UDP Echo Client server created in PART 1.1
## The differnce is that this is protocol independent    #########
## (support both IPv4 and IPv6).                         #########
## Just enter the Server host corrosponding to the server ########
## When running on local machine, the hosts are:         #########
## For IPv4: localhost; For IPv6: ip6-localhost          #########
''')

# Assign the values obtained from the ArgumentParser
totalMessages = args.total_messages
messageInterval = args.message_interval
packetSize = args.packet_size

# Since window size is same as packetSize, we can use the same value for window size
bufferSize = packetSize

serverHost = input("Enter the Server Host Name: ")

messageFromClient = "This is a message from Client"
# serverIP = '127.0.0.1'
serverPort = 20001
bytesToSend = str.encode(messageFromClient) # client message encoded
avgRTT = 0 # Initializing the average RTT
packetSuccessCount = 0 # Initializing the success count of packets
```

```
socketAddrInfo = socket.getaddrinfo(serverHost, serverPort,
socket.AF_UNSPEC, socket.SOCK_DGRAM)
firstInfo = socketAddrInfo[0] #First tuple

# Creation of UDP Socket on Client's Side for the user defined family and
protocol
UDPSocket = socket.socket(family=firstInfo[0], type=firstInfo[1])

socketAddress = (serverHost, serverPort)

# Timeout setting to interval
UDPSocket.settimeout(messageInterval)

# Callibrating the buffer size
# To adjust the window size, we send the message to the server with ADJ
denoting Adjustment
msg = str.encode(f"ADJ {packetSize}")
print("\n")
# Sending the message to the server
UDPSocket.sendto(msg, socketAddress)

# Receiving the ACK from the server
recievedACK = UDPSocket.recvfrom(bufferSize)

print(recievedACK[0].decode())
print("\n\n")

packetCount = 0 # defined for calculating RTT and loss percentage

while(totalMessages > 0):
    packetCount += 1
    totalMessages -= 1
    print("\n                   ***\n")
    print(f"Sending packet number {packetCount} of {packetSize} bytes...")
    sendTimestamp = datetime.datetime.now().timestamp() # Time at which the
packet is sent

    # Sending the packet to the server
    UDPSocket.sendto(bytesToSend, socketAddress)

    try:
        # Receiving the ACK from the server
        recievedACK = UDPSocket.recvfrom(bufferSize)
    except socket.timeout:
        # Server timeout leads to packet loss
```



```
        print("\nPACKET LOST!!\n")
        continue

    recievingTimestamp = datetime.datetime.now().timestamp() # Time at which the packet is recieved

    print(f"Message from the server: {recievedACK}")
    print("\n\n")
    print(f"Packet number {packetCount} recieved successfully")
    packetSuccessCount += 1
    print(f"Round Trip Time for packet {packetCount} is {recievingTimestamp-sendTimestamp}")
    print("\n")

    # Calculating the average RTT
    avgRTT += recievingTimestamp-sendTimestamp

    # Until the current interval is not over, the process is paused by making it sleep for the left time
    delay = recievingTimestamp-sendTimestamp
    if(delay < messageInterval):
        time.sleep(messageInterval-delay)

# Sending of messages completed.
print("Process of Sending Packets has completed!\n")
# Send termination message to the server
UDPSocket.sendto(str.encode("TERMINATE"), socketAddress)
serverResponse = UDPSocket.recvfrom(bufferSize)
print(f"Message from the server: {serverResponse}")
print("\n\n")

# Calculating the Average RTT
avgRTT = avgRTT/packetSuccessCount
print(f"Average Round Trip Time is {avgRTT}")

# Calculating the loss percentage
lossPercentage = (1 - (packetSuccessCount/packetCount))*100
print(f"Loss Percentage is {lossPercentage}%")
print("The program is complete and is Protocol Independent\n")
print("    ------------------------------------")
print("\n")

# TERMINATE the socket
UDPSocket.close()
```

- Server Implementation:

```
# ***PROBLEM STATEMENT***

# Revise echo client and server to be protocol independent
# (support both IPv4 and IPv6).


# The code for the server starts here
import socket
import time
import random
from datetime import datetime

# Start with a display message
print('''
**************************************************************
*                       WELCOME!                             *
*                 This is the server                         *
**************************************************************
## This is same as the UDP Echo Client server created in PART 1.1
## The differnce is that this is protocol independent    #########
## (support both IPv4 and IPv6).                         #########
## Just enter the Server host according to your choice   #########
## When running on local machine, the hosts are:         #########
## For IPv4: localhost; For IPv6: ip6-localhost          #########
''')

# serverIP = '127.0.0.1'
serverPort = 20001
serverPublicIP = socket.gethostbyname(socket.gethostname())

bufferSize = 1024

# Taking the server Host from the user which depicts whether the protocol is IPv4 or IPv6
serverHost = input("Enter the Server Host: ")


ACKmessage = "Packet Acknowledged!"
encodedACKmessage = ACKmessage.encode()
```

```
# We need to make the echo client server protocol independent
# So, instead of creating the socket for a fixed family
# We will create the socket based on the Server Host Name entered by the
used
# And then bind to the correct family based on the client's request

# For doing this, we use the getaddrinfo() function which return a list of
tuples that contain information about the socket
# The synatx for it is: socket.getaddrinfo(host, port, family, type)
socketAddrInfo = socket.getaddrinfo(serverHost, serverPort,
socket.AF_UNSPEC, socket.SOCK_DGRAM)

# We take the first tuple from the list of tuples in socketAddrInfo
firstInfo = socketAddrInfo[0] #First tuple

# The first tuple obtained from the list of tuples in socketAddrInfo
contains the following information
# (family, type, proto, canonname, sockaddr)
# So, we can use the first tuple to create the UDPsocket

# Creation of UDP Socket on Client's Side based on the family obtained from
the socketAddrInfo
serverSocket = socket.socket(family=firstInfo[0], type=firstInfo[1])
socketAddress = (serverHost, serverPort)

# Bind the socket to the address
serverSocket.bind(socketAddress)

print(f"UDP Server is up and running on address {serverPublicIP} :
{serverPort}")


# Listen for incoming connections
while True:
    # Get client requests
    clientRequestData, clientAddress = serverSocket.recvfrom(bufferSize)
    print("\n")
    print(f"Message from the Client: {clientRequestData}")
    print(f"Client Address: {clientAddress}")

    decodedData = clientRequestData.decode()
    print(f"Decoded Data: {decodedData}")

    # If the decoded message required the adjustment of window size
    if decodedData.split()[0] == "ADJ":
        print("Adjusting Window Size...")
        bufferSize = int(decodedData.split(" ")[1])
        print()
        serverSocket.sendto(str.encode(f"Adjusted Window Size to
{bufferSize}"), clientAddress)
        print(f"Adjusted Window size to {bufferSize}")
        print("\n")
        continue

    # If the decoded message requests Termination of Connection
    if decodedData == "TERMINATE":
        print("Terminating Connection...")
        serverSocket.sendto(str.encode("Connection Terminated"),
clientAddress)
        print("\nConnection Terminated!\n")
        break

    # Introducing Packet Delay
    time.sleep(random.randint(1, 5000) / 1000)

    # Introducing Packet Loss with Loss Probability of 0.1
    if random.randint(1, 10) <= 1:
        print("Packet Loss Occurred!\n")
        continue

    # Sending reply to the client with an Acknowledgement message
    serverSocket.sendto(encodedACKmessage, clientAddress)
    print(f"ACK Message: {ACKmessage}")


# Close the socket with a message
print("\n\nClosing the Server Socket...")
serverSocket.close()
print("\nServer Socket Closed!")
```

NOTE: The drive links of all the source code have been attached in the answers to the respective questions using Hyperlink. These files can also be found in the zipped file. The video link for all the parts is given at the beginning of the report as well as at the starting of each part.