Computer network

ASSIGNMENT-3

NAME:-SHAIK RAHUL email:shaikrahul@iitbhilai.ac.in
ID :-12041350

Part 1: Client/Server Programming to transfer content of a File.

We Create our own TCP client and server application to transfer content of a file.

Server.py

```
import socket
import os
import time
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
def packet_send(link,frame):
   link.send(bytes(frame,"utf-8"))
   print("==>>",frame)


def packet_recived(link):
   frame = link.recv(1024).decode()
   print("<<==",frame)
   return frame



def server_socket():
   IP = "localhost" # default localhost ipv4
   PORT = 5678
   soc = socket.socket()
   soc.bind((IP, PORT)) # binding ip and port
   soc.listen(10)
   print("tcp server ready and listing to client")
   while True:
```

```
        link , addr = soc.accept()
        print("Client Connected    :",addr)
        frame = packet_recived(link)
        valid = os.path.exists(f"./server_folder/{frame}")

        if valid:
            with open(f"./server_folder/{frame}","r") as f:
                file_data = f.read().split(" ")
            packet_send(link,file_data[0])

            while True:
                frame = packet_recived(link)
                time.sleep(1)
                if frame == "BREAK":
                    packet_send(link,"BREAK")
                    break

                inde = int(frame.split("$")[1])

                packet_send(link,file_data[inde])
                if file_data[inde] == "EOF":
                    break


        else:
            packet_send(link,"404:File-not-Found")

        link.close()
        print("Client Disconnect and close :",addr)


if __name__ == '__main__':
    server_socket()
```

client.py

```python
import socket
import time
#================================================================================
name_of_file = input("file name (file_1.txt) :")
if name_of_file == "":
   name_of_file = "file_1.txt"

def client():
   IP = "localhost"
   PORT = 5678
   soc = socket.socket()
   soc.connect((IP, PORT))
   msg = name_of_file
   soc.send(msg.encode())
   recv_info = []
   while True:
       frame = soc.recv(1024).decode()
       time.sleep(2)
       if frame in ["404 file is not found"]:
           print(f"invalid file : File '{name_of_file}' not found")
           break
       recv_info.append(frame)
       if frame == "EOF":
           break
       else:
           soc.send(bytes(f"WORD_${len(recv_info)}","utf-8"))
   soc.close()
   if len(recv_info) > 0:
       content = " ".join(recv_info)
       with open(f"./client_folder/{name_of_file}","w") as f:
           f.write(content)
       print("Data Received :",content)

if __name__ == '__main__':

   client()
```

Explanation of code:
Server.py

Create a TCP socket.

- **Bind the IP address and PORT to the server socket.**
- **Listening for the clients.**
- **Accept the connection from the client.**
- **Receive the filename from the client and create a text file. the server should look for the file in the local directory if the file is not there it should send back a message 404 file not found.**
- **After receiving my name the client create a local file. The process continuous until the client receive the keyword Eof.**
- **Close the connection.**

Client.py

1, create a socket for the client .

2, connect to the server.

3, Read the content inside the text file.

4, send the words to the server. Receive the response from the server.

5, close the connection.

How to run the code:

1. **To run the server.py just type in terminal “python3 server.py”**
2. **Open another terminal run client.py just type in terminal “python3 client.py” .Enter the file name”file_1.txt” or “file_2.txt” .**

Part-2:Client/Server Programming for network analysis

Question-1:

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

Server.py

```
import socket

import time

IP = "127.0.0.1"

Port = 5065

BiteSize = 1024
```

```
try:



   udp_socket = socket.socket(

       family=socket.AF_INET, type=socket.SOCK_DGRAM)

    udp_socket.bind((IP, Port))

   print("udp server is created and listening")

except:

   print("Socket Not Creted due to some error")









while(True):

   client_msg, client_add = udp_socket.recvfrom(BiteSize)

   if client_msg:



       cMsg = client_msg.decode('utf-8')

       client_IP = client_add

       print("message:",cMsg, "Ip:", client_IP)



       time.sleep(1)



       udp_socket.sendto(client_msg, client_add)
```

Client.py

```
import socket

import time

#=============================================================================

name_of_file = input("file name (file_1.txt) :")

if name_of_file == "":

   name_of_file = "file_1.txt"



def client():

   IP = "localhost"

   PORT = 5678

   soc = socket.socket()

   soc.connect((IP, PORT))

   msg = name_of_file

   soc.send(msg.encode())

   recv_info = []

   while True:

       frame = soc.recv(1024).decode()

       time.sleep(2)

       if frame in ["404 file is not found"]:

           print(f"invalid file : File '{name_of_file}' not found")

           break
```

```
        recv_info.append(frame)

        if frame == "EOF":

            break

        else:

            soc.send(bytes(f"WORD_${len(recv_info)}","utf-8"))

    soc.close()

    if len(recv_info) > 0:

        content = " ".join(recv_info)

        with open(f"./client_folder/{name_of_file}","w") as f:

            f.write(content)

        print("Data Received :",content)



if __name__ == '__main__':



    client()
```

Explanation of code:

Server.py:

1)socket() function is used for the creating a socket. Socket module and specify the type as udp using DGRAM

2, BINDING THE IP ADDRESS AND THE PORT NUMBER USING BIND () FUNCTION.

3, udp server wait for the client.

4, if we receive something from that client using receivefrom() decode it and send back the message to the client using sendto() function.

Client.py:

1, we give the comment line arguments send, time travel, and package size to take the number of messages.

2, creative Udp socket using the socket () from the socket model and specify the type as udp using DGRAM .

3, in send function clients send the given number of messages to the server by creating the random messages by the package size specified.

4,in receive () function receive the data from the server and decode it.

5, decode function is useful for the conversion of binary into strings.

6, using p_thread we can enjoy that all the package is sent and received

7, to calculate the throughput we send the messages with a small time interval gap.

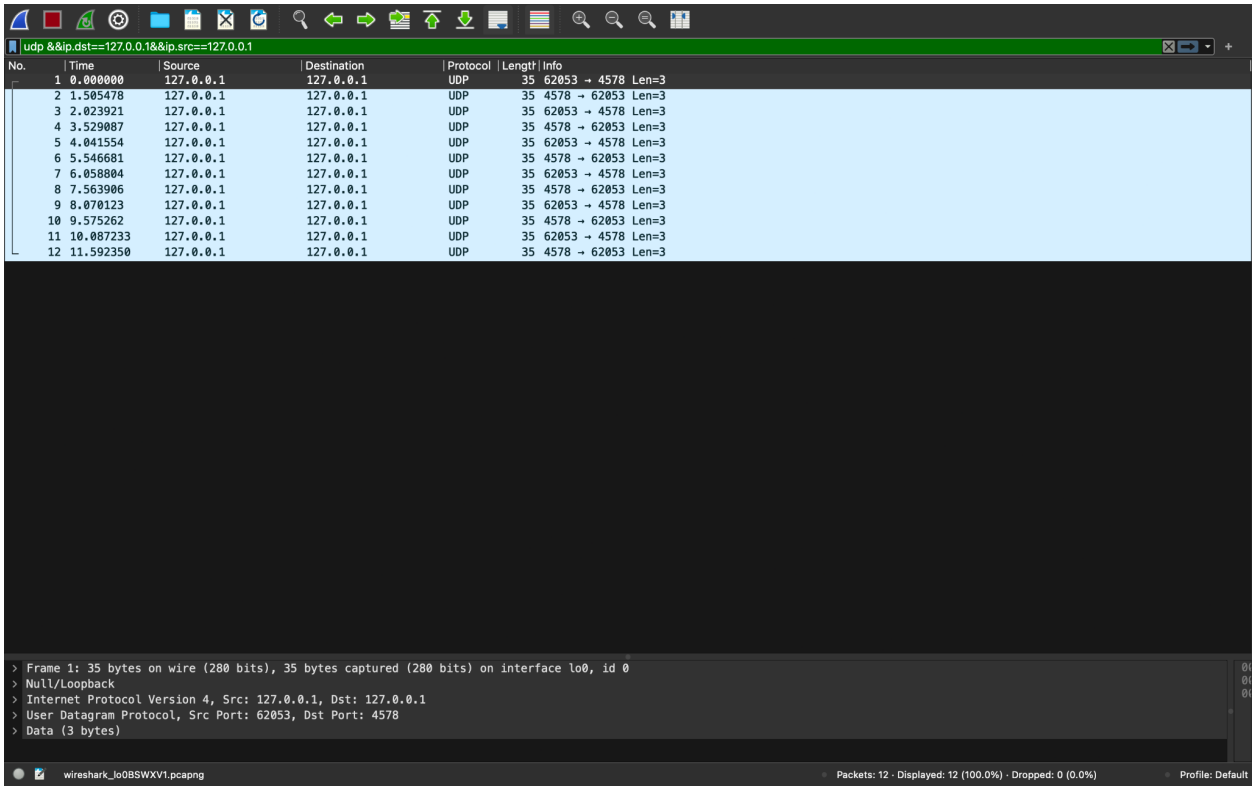
8, we calculate the number of packets sent in a second and append it to the packets list.

Screenshots of running:

```
(p_3_10_6) shaikrahul@shaikrahu part-2-question-1 % python3 server.py
udp server is created and listening
message: 9L8 Ip: ('127.0.0.1', 62053)
message: BID Ip: ('127.0.0.1', 62053)
message: GXL Ip: ('127.0.0.1', 62053)
message: NKZ Ip: ('127.0.0.1', 62053)
message: D5X Ip: ('127.0.0.1', 62053)
message: CX2 Ip: ('127.0.0.1', 62053)
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
(p_3_10_6) shaikrahul@shaikrahu part-2-question-1 % python3 client.py 6 2 3
Internet_protocal: 127.0.0.1
port: 4578
message no:1 sent. message:9L8
message no:1 received, message: 9L8
message no:2 sent. message:BID
message no:2 received, message: BID
message no:3 sent. message:GXL
message no:3 received, message: GXL
message no:4 sent. message:NKZ
message no:4 received, message: NKZ
message no:5 sent. message:D5X
message no:5 received, message: D5X
message no:6 sent. message:CX2
message no:6 received, message: CX2
RoundTripTime
loss percentage=0.0
The Round TrIP Time for message1: 1.4992997646331787
The Round TrIP Time for message2: 1.505296230316162
The Round TrIP Time for message3: 1.5054821968078613
The Round TrIP Time for message4: 1.5114591121673584
The Round TrIP Time for message5: 1.5056641101837158
The Round TrIP Time for message6: 1.511491060256958
(p_3_10_6) shaikrahul@shaikrahu part-2-question-1 %
```



Measurement	Captured	Displayed	Marked
Packets	12	12 (100.0%)	–
Time span, s	11.592	11.592	–
Average pps	1.0	1.0	–
Average packet size, B	35	35	–
Bytes	420	420 (100.0%)	0
Average bytes/s	36	36	–
Average bits/s	289	289	–

When client is running capture the data packets

which sending 12 packets of length 36 bytes data and got 24 packets in wireshark for this process.

In which 12 are sending packets

And 12 are receiving packets.

part-2

Question-2:

Create an iperf like application using the above developed echo client and server program.
Server.py:

```
import socket #importing the socket module
import time
IP = "127.0.0.1" #localhost ipv4 addr_essess
port_no = 5048 # port number = 5048
Buffer_size = 1024 # buffersize inializing

try:
   soc_ket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
   soc_ket.bind((IP, port_no))
   print("udp sever is ready and listening")
except:
   print(" Socket creating error")
```

```
while(True):
   message, addr_ess = soc_ket.recvfrom(Buffer_size) #data packets receive from client
   if message:
       soc_ket.sendto(message, addr_ess)
       #data packets sending to client
```

Client.py:

```
import socket
import sys
import random
import string
import time
#=============================================================
Buffersize = 1024
list_send_time= []
list_recv_timestamp = []
Ip = "localhost"
port_no = 5048
if(len(sys.argv)!=3):
   print("arguments overflow error")
   exit(0)
x =[]
y_throughput =[]
y_average_delay =[]
y_averagedelay = []
#=========================================================
time_interval_len = int(sys.argv[1]) # first argument time interval argument
data_size = int(sys.argv[2]) #second argument packetsize
sock_et = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #udp socket is ready
print("run the plot.plt to get the graphs")
def echoclientsoc():
   t=0
   count=0
   packet_send=0
   time_initialize=time.time()

#=============================================================================================
==========
   while True:
       if(t==time_interval_len):
```

```
            break
        msg_info = ''.join(random.choices(

            string.ascii_uppercase + string.digits, k=data_size)).encode("utf-8")
        sock_et.sendto(msg_info, (Ip, port_no))  #sending msg_info to server
        sent_time = time.time()
        list_send_time.append(sent_time)
        msg_info, _ = sock_et.recvfrom(Buffersize)
        if msg_info:
            packet_recv_time = time.time()  #packet recived time calulating
            count+=packet_recv_time-sent_time
            packet_send+=1


            if(packet_recv_time-time_initialize>=1):
                time_initialize=time.time()

                round_trip_time=count/packet_send
                count=0
                avg_delay=round_trip_time/2
                t+=1
                print(t,(packet_send*data_size)/1000,avg_delay*1000,sep=", ") #printing
the throushput and average delay
                x.append(t)
                y_throughput.append((packet_send*data_size)/1000)
                y_averagedelay.append(avg_delay*1000)
                packet_send=0
            list_recv_timestamp.append(packet_recv_time)
        else:
            continue

if __name__ == "__main__":
   echoclientsoc()


import matplotlib.pyplot as plt
plt.xlabel("time")
plt.ylabel("throughput ")
plt.plot(x,y_throughput)
plt.savefig("throput_graph.png")
plt.clf()
plt.xlabel("time")
```

```
plt.ylabel("averagewg delay ms")
plt.plot(x,y_averagedelay)

plt.savefig("average_delay_graph")
```

How to run code:
run the command "python3 server.py"
open another terminal " python3 client.py 6 2 > rahul.txt"

Code review:
Server.py
1)creating udp socket usong socket().
2,Waiting for the client to send a message after opening the socket.
3,Recvfrom() was used to obtain the data from the client, and decode ().
4)Using sento(), I established an echo client server and sent the reverse data to the client.

Client.py
1,From the command line inputs, I was able to determine the time interval and packet size.
2,In the client, I have created a socket that is comparable to the server's.
3,Using the time lists, I was able to save the beginning and finishing times of the packets.

Calculated the rtt using the time lists and average delay which is rtt/2.

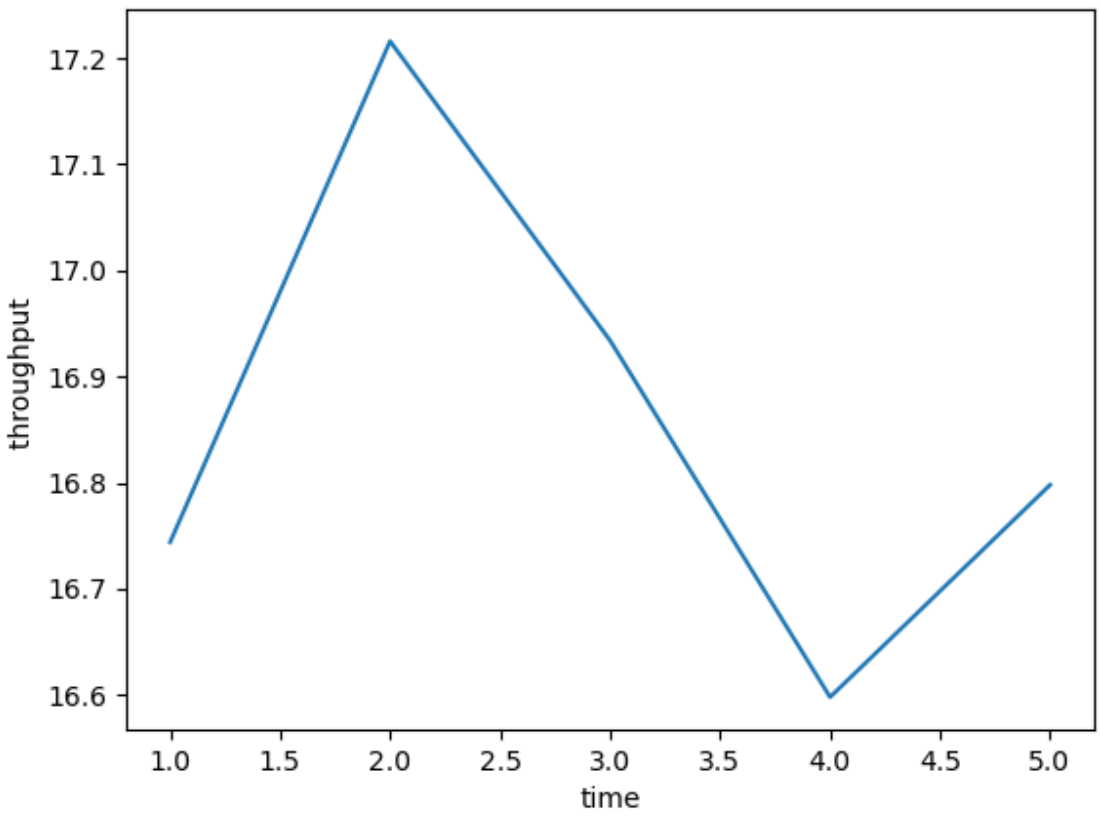
Calculated the throughput by packetsize*no of packets /total time taken .

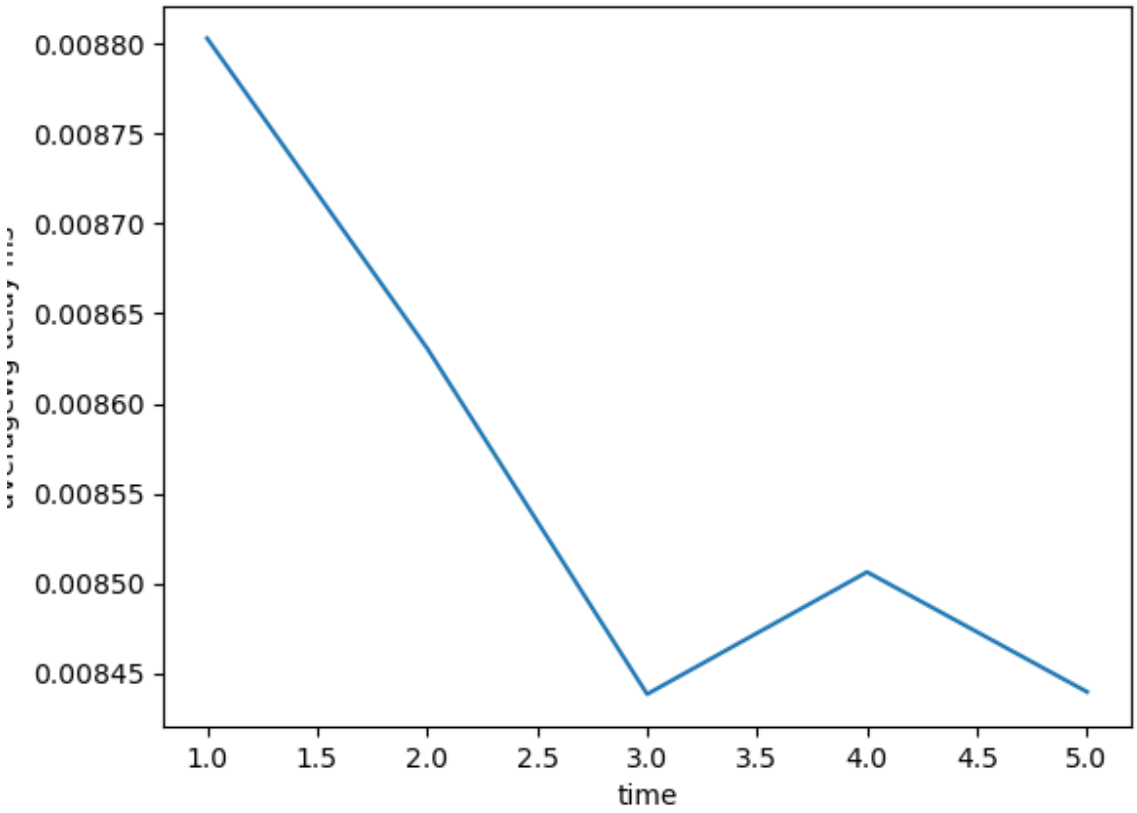
Plot the graphs using it.

Plot the graphs:

```
import matplotlib.pyplot as plt
plt.xlabel("time")
plt.ylabel("throughput ")
plt.plot(x,y_throughput)
plt.savefig("throput_graph.png")
plt.clf()
plt.xlabel("time")
plt.ylabel("averagewg delay ms")
plt.plot(x,y_averagedelay)

plt.savefig("average_delay_graph")
```


17.2
17.1
17.0
16.9
16.8
16.7
16.6
throughput
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
time



Average delay vs time graph
How to run the code:

1,To run the server type “python3 server.py” in the terminal.

2,Next run the client using command “ python3 client.py 6 2 > rahul.txt” and save it in txt file.

3)genarated values into output.txt file.

1, 16.744, 0.008803062101784245

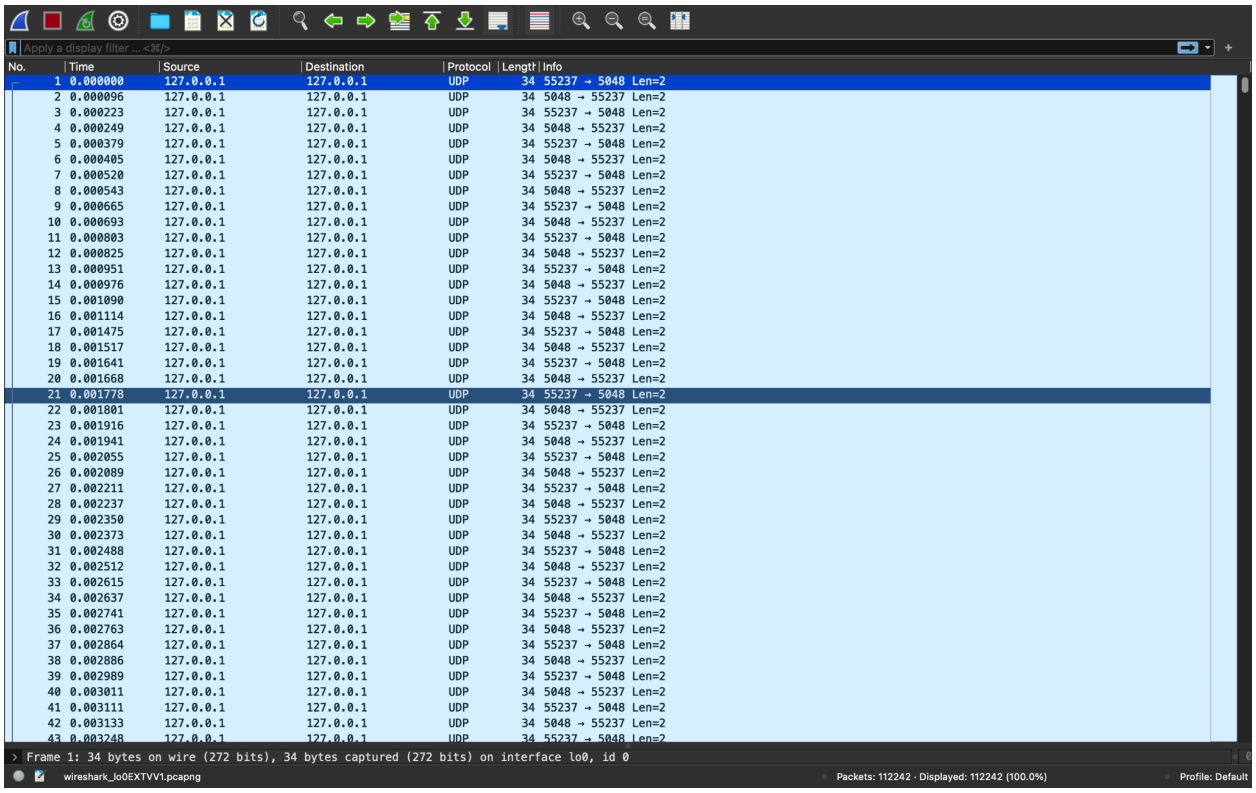
2, 17.216, 0.008630735945081179

3, 16.934, 0.008438404214358236

4, 16.598, 0.00850629168743828

5, 16.798, 0.008439631643203317

Wireshark:



Part-3:

Server.py:

```
import socket


host = ""

print("---give any port which you like")

port = int(input("Port number:"))




soc=socket.socket(socket.AF_INET6, socket.SOCK_STREAM)

soc.setsockopt(socket.IPPROTO_IPV6, socket.IPV6_V6ONLY, 0)# The setsockopt() function provides an application program with the means to control socket behavior. An application program can use setsockopt() to allocate buffer space, control timeouts, or permit socket data broadcasts.

soc.bind((host, port))

soc.listen(3)



while True:

   p, add = soc.accept()



   if p:

       print('Connection established with ', add)

       while True:
```

```
            income_data = p.recv(1024)

            if not income_data:

                break

            p.send(income_data)
```

Client.py:

```
#importing required libraries

import socket



print("choose any internet_protocol 1 or 2:")

internet_protocol = int(input("1 -> IPV4 \n2 -> IPV6 \nEnter the internet_protocol 1
or 2: "))

host = input("Hostname: ")



print("give any port_number number which you like-")

p_number = int(input("Port number: "))



if internet_protocol == 1:

   family = socket.AF_INET # default ipv4







else:

   family = socket.AF_INET6 # default  IPv6
```

```
#=============================================================================================
======================


address = socket.getaddrinfo(host, p_number, family= family)


for i in range(len(address)):
   host = address[i][4][0]




   p = socket.socket(family=family, type=address[i][1])



   p.connect((host,p_number))




   p.close()
   break


family = address[i][0]
type = address[i][1]
host = address[i][4][0]
```

```
#=============================================================================================
==================
```

```
z= socket.socket(family= family, type= type)
```

```
z.connect((host, p_number))
```

```
print("ENTER 'end' to stop close the connection!")
```

```
while True:
```

```
   coming_data = input("give the data to send: ")
```

```
   if coming_data == "end":
```

```
       break
```

```
   z.send(str.encode(coming_data))
```

```
   out_data = z.recv(1024)
```

```
   print(out_data.decode())
```

```
z.close()
```

How to run code code:

First the run the command ”python3 server.py”

Enter port number:4567

Next Run “python 3 client.py”

Select 1 for ipv4

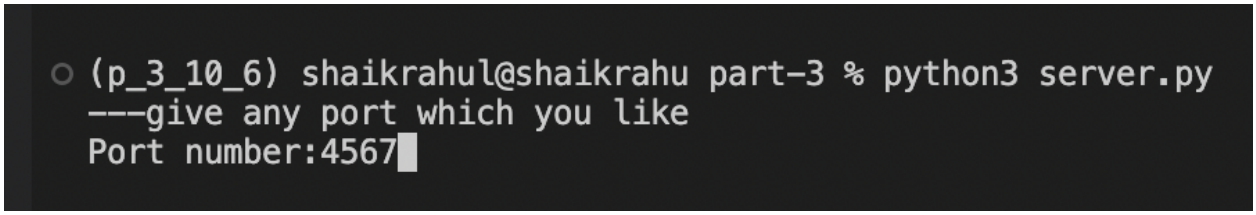
Select 2 for ipv6

Hostname :localhost

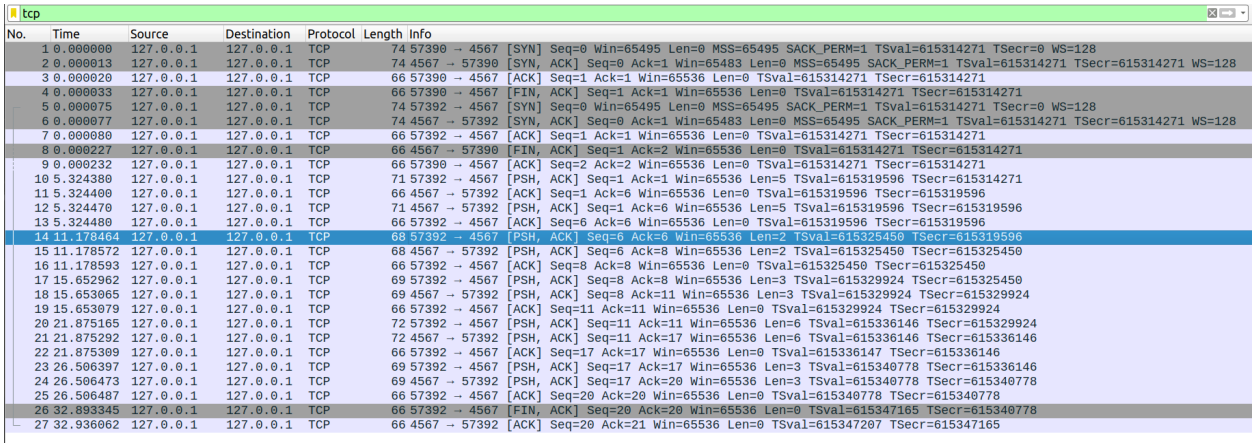
Port number:4567

Enter any messages.

“Quit” to detach the connection



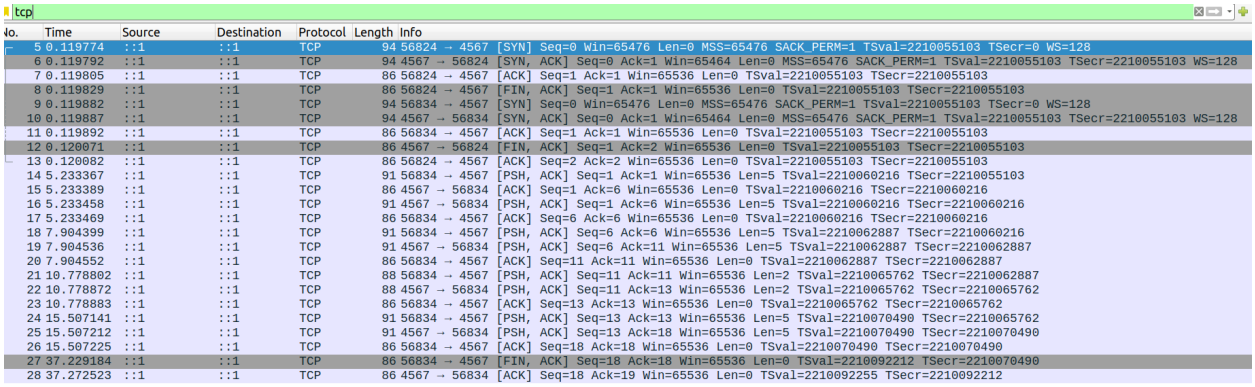
```
selcect the protocol to be use
 1.) type 1 for IPV4
 2.) type 2 for IPV6
 enter the protocol : 1
Hostname: localhost
port number: 4567
ENTER 'quit' to exit!
give the data to send: rahul
rahul
give the data to send: cn
cn
give the data to send: 301
301
give the data to send: course
course
give the data to send: end
end
give the data to send: quit
```

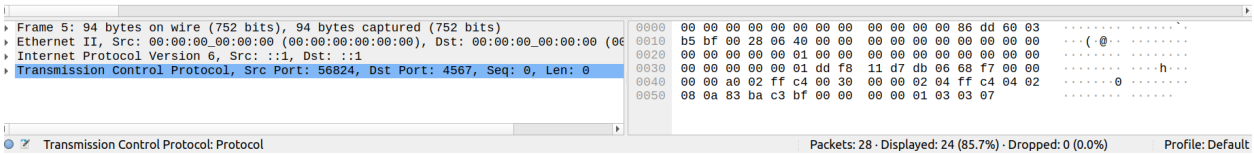


tcp

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	74	57390 → 4567 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=615314271 TSecr=0 WS=128
2	0.000013	127.0.0.1	127.0.0.1	TCP	74	4567 → 57390 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=615314271 TSecr=615314271 WS=128
3	0.000020	127.0.0.1	127.0.0.1	TCP	66	57390 → 4567 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=615314271 TSecr=615314271
4	0.000033	127.0.0.1	127.0.0.1	TCP	66	57390 → 4567 [FIN, ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=615314271 TSecr=615314271
5	0.000075	127.0.0.1	127.0.0.1	TCP	74	57392 → 4567 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=615314271 TSecr=0 WS=128
6	0.000077	127.0.0.1	127.0.0.1	TCP	74	4567 → 57392 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=615314271 TSecr=615314271 WS=128
7	0.000080	127.0.0.1	127.0.0.1	TCP	66	57392 → 4567 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=615314271 TSecr=615314271
8	0.000227	127.0.0.1	127.0.0.1	TCP	66	4567 → 57390 [FIN, ACK] Seq=1 Ack=2 Win=65536 Len=0 TSval=615314271 TSecr=615314271
9	0.000232	127.0.0.1	127.0.0.1	TCP	66	57390 → 4567 [ACK] Seq=2 Ack=2 Win=65536 Len=0 TSval=615314271 TSecr=615314271
10	5.324380	127.0.0.1	127.0.0.1	TCP	71	57392 → 4567 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=5 TSval=615319596 TSecr=615314271
11	5.324400	127.0.0.1	127.0.0.1	TCP	66	4567 → 57392 [ACK] Seq=1 Ack=6 Win=65536 Len=0 TSval=615319596 TSecr=615319596
12	5.324470	127.0.0.1	127.0.0.1	TCP	71	4567 → 57392 [PSH, ACK] Seq=1 Ack=6 Win=65536 Len=5 TSval=615319596 TSecr=615319596
13	5.324480	127.0.0.1	127.0.0.1	TCP	66	57392 → 4567 [ACK] Seq=6 Ack=6 Win=65536 Len=0 TSval=615319596 TSecr=615319596
14	11.178464	127.0.0.1	127.0.0.1	TCP	68	57392 → 4567 [PSH, ACK] Seq=6 Ack=6 Win=65536 Len=2 TSval=615325450 TSecr=615319596
15	11.178572	127.0.0.1	127.0.0.1	TCP	68	4567 → 57392 [PSH, ACK] Seq=6 Ack=8 Win=65536 Len=2 TSval=615325450 TSecr=615325450
16	11.178593	127.0.0.1	127.0.0.1	TCP	66	57392 → 4567 [ACK] Seq=8 Ack=8 Win=65536 Len=0 TSval=615325450 TSecr=615325450
17	15.652962	127.0.0.1	127.0.0.1	TCP	69	57392 → 4567 [PSH, ACK] Seq=8 Ack=8 Win=65536 Len=3 TSval=615329924 TSecr=615325450
18	15.653065	127.0.0.1	127.0.0.1	TCP	69	4567 → 57392 [PSH, ACK] Seq=8 Ack=11 Win=65536 Len=3 TSval=615329924 TSecr=615329924
19	15.653079	127.0.0.1	127.0.0.1	TCP	66	57392 → 4567 [ACK] Seq=11 Ack=11 Win=65536 Len=0 TSval=615329924 TSecr=615329924
20	21.875165	127.0.0.1	127.0.0.1	TCP	72	57392 → 4567 [PSH, ACK] Seq=11 Ack=11 Win=65536 Len=6 TSval=615336146 TSecr=615329924
21	21.875292	127.0.0.1	127.0.0.1	TCP	72	4567 → 57392 [PSH, ACK] Seq=11 Ack=17 Win=65536 Len=6 TSval=615336146 TSecr=615336146
22	21.875309	127.0.0.1	127.0.0.1	TCP	66	57392 → 4567 [ACK] Seq=17 Ack=17 Win=65536 Len=0 TSval=615336147 TSecr=615336146
23	26.506397	127.0.0.1	127.0.0.1	TCP	69	57392 → 4567 [PSH, ACK] Seq=17 Ack=17 Win=65536 Len=3 TSval=615340778 TSecr=615336146
24	26.506473	127.0.0.1	127.0.0.1	TCP	69	4567 → 57392 [PSH, ACK] Seq=17 Ack=20 Win=65536 Len=3 TSval=615340778 TSecr=615340778
25	26.506487	127.0.0.1	127.0.0.1	TCP	66	57392 → 4567 [ACK] Seq=20 Ack=20 Win=65536 Len=0 TSval=615340778 TSecr=615340778
26	32.893345	127.0.0.1	127.0.0.1	TCP	66	57392 → 4567 [FIN, ACK] Seq=20 Ack=20 Win=65536 Len=0 TSval=615347165 TSecr=615340778
27	32.936062	127.0.0.1	127.0.0.1	TCP	66	4567 → 57392 [ACK] Seq=20 Ack=21 Win=65536 Len=0 TSval=615347207 TSecr=615347165

capturing the packets while running the ipv4





While running the ipv6 packets captured by the wireshark

Code explanation:

server.py

1,As soon as the ipv6 protocol server was formed, the ipv4 to ipv6 conversion was specified using the setsockopt() method with ipv6 only set to false.

2,then use the command line to obtain the input port number.

Client.py:

1,Using the command line, we have obtained the input IP address, port number, and protocol.

2,Use the getaddrinfo() function to retrieve a list of tuples including the address, protocol, and port.

3,Create a temporary socket to check the port and IP address.

4,After confirmation, build a socket based on the information obtained and carry out the same operations as an echo client-server application.

Part-4:Image transfer application.

Client.py

```
import socket
from PIL import Image
host = '127.0.0.1'
port = 4567

temp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
temp.connect((host, port))
print("Connecting to the server...\n")


with open('ClientFolder/a.jpg', 'wb') as f:
        print('To save the received image, a new file has been generated.')
        data = temp.recv(100000)
        f.write(data)
        print("collect recived data from the server ")
        f.close()




with open('ClientFolder/a.jpg', 'rb') as f:
        rahul = Image.open(f)
        rahul.show()
```

```
print(' RECEIVED image')
temp.close()
print('successfully executed the program')
```

Server.py:

```
import socket
from PIL import Image
portnumber = 4567
host = '127.0.0.1'
temp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

temp.bind((host, portnumber))
temp.listen(1)

print('Server has ready. search for any clients trying to connect.')
while True:
   connection, addr_ess = temp.accept()
   if connection:


       print(f'Connecting to {addr_ess}...')


       with open('ServerFolder/b.jpg', 'rb') as f:
           connection.sendall(f.read())
           f.close()
           break


print('successfully executed the program')
```

Run the code:
1,”python server.py”
2,open another terminal run “python client.py”

Review of code:
Client.py

1,We first build a socket and attach it to the
2,The data that we receive from the server is first saved into a file that we first create.

Server.py
2,After that, we look for any clients attempting to connect.
3,If we locate the client successfully, "try" runs and delivers the image we specified to the client from the server folder.

Sent image:

Received image: