CS301 Computer Networks
Assignment 3
By Chaitanya Bisht – 12040450

Part 1

For this part, we create a UDP server and a UDP client, the UDP server is hosted at localhost:10001.

The client requests the file by providing the file name and the server checks if the file is available or not. If it is available, the server only sends the file line by line when the client requests the following line. When the server sends EOF, the client knows the end of the file and terminates the connection.

Start the server by executing the following command:

```
$ python3 udp_server.py
```

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_a  master  ⚡
₹ py udp_server.py
Server is listening on port 10001
```

Similarly, on another terminal, we execute the following command to start the client:

```
$ python3 udp_client.py
```

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_a  master  ⚡
₹ py udp_client.py
Enter file name: file.txt
Received Chaitanya Bisht from server
Requested Word#2 to server
Received def from server
Requested Word#3 to server
Received ghi from server
Requested Word#4 to server
Recieved EOF from server
Transfer complete
```

We enter the name of the file we are looking for and the server starts sending the contents. We get Word#1 from server and then we request the Word#2 and so on until we get EOF when the transfer is complete.

During the transer, the server side terminal looks like this

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_a  master  ⚡
₹ py udp_server.py
Server is listening on port 10001
File name requested: file.txt
Sent Word#1 to client
Sent Word#2 to client
Sent Word#3 to client
Sent EOF to client
Transfer complete
```

The contents of the requested file.txt looks like this

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_a  master  ⚡
₹ cat file.txt
Chaitanya Bisht
def
ghi
EOF%
```

The client saves the received file as file_server.txt. The contents of the file_server.txt looks like:

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_a  master  ⚡
₹ cat file_server.txt
Chaitanya Bisht
def
ghi
EOF%
```

Error Handling:

If we give a name of the file that does not exist, we return file not found:

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_a  master  ⚡
₹ py udp_client.py
Enter file name: some_random_file_name
File not found on server
```

The Wireshark capture looks like this

No.	Time	Source	Destination	Protocol	Length	Bytes in flight	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	60		40992 → 10001 Len=18
2	0.000260622	127.0.0.1	127.0.0.1	UDP	58		10001 → 40992 Len=16
3	1.001457907	127.0.0.1	127.0.0.1	UDP	43		40992 → 10001 Len=1
6	3.003351727	127.0.0.1	127.0.0.1	UDP	46		10001 → 40992 Len=4
7	4.004752283	127.0.0.1	127.0.0.1	UDP	43		40992 → 10001 Len=1
16	6.006476382	127.0.0.1	127.0.0.1	UDP	46		10001 → 40992 Len=4
17	7.007356660	127.0.0.1	127.0.0.1	UDP	43		40992 → 10001 Len=1
18	9.009532330	127.0.0.1	127.0.0.1	UDP	45		10001 → 40992 Len=3
31	26.813593930	127.0.0.1	127.0.0.1	UDP	74		53219 → 53219 Len=32
32	26.824650205	127.0.0.1	127.0.0.1	UDP	66		53219 → 53219 Len=24
33	26.825294802	127.0.0.1	127.0.0.1	UDP	74		53219 → 53219 Len=32
34	26.835597250	127.0.0.1	127.0.0.1	UDP	10…		53219 → 53219 Len=960
35	26.835603082	127.0.0.1	127.0.0.1	UDP	442		53219 → 53219 Len=400
36	26.835605182	127.0.0.1	127.0.0.1	UDP	554		53219 → 53219 Len=512
49	86.869698618	127.0.0.1	127.0.0.1	UDP	74		53219 → 53219 Len=32
50	86.882049847	127.0.0.1	127.0.0.1	UDP	66		53219 → 53219 Len=24

Code for udp_client.py

```python
from socket import socket, AF_INET, SOCK_DGRAM
from time import sleep

UDP_client_socket = socket(family = AF_INET, type = SOCK_DGRAM)
UDP_client_socket.settimeout(1)

buffer_size = 1024

file_name = input("Enter file name: ")

UDP_client_socket.sendto(("FILE_NAME:"+file_name).encode(), ("127.0.0.1", 10001))
    You, yesterday • init …

next = 1
f = open(file_name.split('.')[0] + "_server." + file_name.split('.')[1], 'w')
while True:
    msg, addr = UDP_client_socket.recvfrom(buffer_size)
    msg = msg.decode()

    if msg == 'FNF':
        print("File not found on server")
        break
    elif msg == 'EOF':
        f.write('EOF')
        f.close()
        print('Recieved EOF from server')
        print('Transfer complete')
        break
```

```
    f.write(msg)
    msg = msg.strip("\n")
    print(f'Received {msg} from server')
    next += 1
    sleep(1)

    UDP_client_socket.sendto(str(next).encode(), ("127.0.0.1", 10001))
    print(f'Requested Word#{next} to server')


    sleep(2)
```

Code for udp_server.py

```
from socket import AF_INET
from socket import socket
from socket import SOCK_DGRAM
from socket import timeout
from time import sleep


server_IP_address_port = ("127.0.0.1", 10001)
UDP_server_socket = socket(family = AF_INET, type = SOCK_DGRAM)

buffer_size = 1024

UDP_server_socket.bind(server_IP_address_port)

print('Server is listening on port 10001')
ptr = None
f = None
while True:
    msg, addr = UDP_server_socket.recvfrom(buffer_size)
    msg = msg.decode()

    if msg.split(':')[0] == 'FILE_NAME':
        print(f'File name requested: {msg.split(":")[1]}')
        try:
            f = open(msg.split(':')[1], 'r')      You, yesterday • ini
            ptr = 1
        except:
            UDP_server_socket.sendto('FNF'.encode(), addr)
```

```
        except:
            UDP_server_socket.sendto('FNF'.encode(), addr)
            f.close()
            print('File not found on server')
            continue
        ptr = 1
    else:
        ptr = msg

    message = f.readline()
    if message.strip('\n') == 'EOF':
        UDP_server_socket.sendto('EOF'.encode(), addr)
        print('Sent EOF to client')
        print('Transfer complete')
        f.close()
        continue

    UDP_server_socket.sendto(str(message).encode(), addr)
    print(f'Sent Word#{ptr} to client')
    sleep(3)
```

Part 2

Question 1

At both the server and client ends, we first open a standard UDP socket with a localhost port and IP address. After that, we begin transmitting packets across the connection. The server is started up first, followed by the client.

On execution, the client requests user input, which includes the given interval, number of echo messages, and packet size as command-line arguments. We first send out a message from the client to the server to update the buffer size to the packet size because we need to keep the buffer size the same as the packet size. This input variable is also sent to the server as a packet, which updates the buffer size.

Once the buffer size is updated, we can start transmitting packets of fixed buffer size from the client to the server and vice-versa.

When the client sends a packet, it assigns it a sequence number, and the server receives it. It responds to that packet in the form of an acknowledgement with the same sequence number,

which the client captures. The difference between the timestamps of recieve and transmission yields the Round-Trip Time (RTT) of the packet, which is displayed.

Every echo client message must be sent after a predetermined time interval. Because a packet's RTT is typically very short, the previous packet returns before the next packet is sent. To remain consistent in the interpacket interval, we add a delay between the next interval and the RTT of the currently transmitted packet. This ensures that packets are only transmitted after the interpacket interval has expired.

To simulate packet loss, the server randomly drops a packet with a 5% chance.

We can start the server by running the following command:

```
$ python3 udp_server.py
```

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/part_b/q1  master  ⚡
₹ py udp_server.py
Server is listening for client connection
```

We start our client by running the following command:

```
$ python3 udp_client.py
```

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/part_b/q1  master  ⚡
₹ py udp_client.py
Enter the number of echo messages to be sent: 5
Enter the interval: 1
Enter the packet size in bytes: 100
Sending server size of buffer
Buffer size set to  100 bytes
Pinging Server, Seq: 1 of size 100 bytes
Reply from server, Seq:  1, RTT:  0.00035381317138671875 seconds
Pinging Server, Seq: 2 of size 100 bytes
Reply from server, Seq:  2, RTT:  0.00032210350036621094 seconds
Pinging Server, Seq: 3 of size 100 bytes
Reply from server, Seq:  3, RTT:  0.00047397613525390625 seconds
Pinging Server, Seq: 4 of size 100 bytes
Reply from server, Seq:  4, RTT:  0.0004069805145263672 seconds
Pinging Server, Seq: 5 of size 100 bytes
Reply from server, Seq:  5, RTT:  0.0004439353942871094 seconds

Average RTT:  0.0004001617431640625 seconds
Packets Dropped:  0
Packets Success:  5
Loss Percentage:  0.0 %
```

After that we enter the parameters like number of echo messages, interval, packet size. Initially both client and server agree upon a buffer size. Then for each ping message, we calculate the RTT.

On the server size, we see the following output

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_b/q1  master  ⚡
₹ py udp_server.py
Server is listening for client connection
Buffer size set to  100 bytes
Ping from client, Seq:  1
Ping from client, Seq:  2
Ping from client, Seq:  3
Ping from client, Seq:  4
Ping from client, Seq:  5
Client has exited
```

The Wireshark capture of the above looks like this:

No.	Time	Source	Destination	Protocol	Length	Bytes in flight	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	130		53219 → 53219 Len=88
8	39.526712661	127.0.0.1	127.0.0.1	UDP	57		52431 → 10001 Len=15
9	39.526892900	127.0.0.1	127.0.0.1	UDP	57		10001 → 52431 Len=15
10	39.527066257	127.0.0.1	127.0.0.1	UDP	43		52431 → 10001 Len=1
11	39.527182755	127.0.0.1	127.0.0.1	UDP	43		10001 → 52431 Len=1
12	40.528163823	127.0.0.1	127.0.0.1	UDP	43		52431 → 10001 Len=1
13	40.528348895	127.0.0.1	127.0.0.1	UDP	43		10001 → 52431 Len=1
14	41.529274454	127.0.0.1	127.0.0.1	UDP	43		52431 → 10001 Len=1
15	41.529435297	127.0.0.1	127.0.0.1	UDP	43		10001 → 52431 Len=1
16	42.530491199	127.0.0.1	127.0.0.1	UDP	43		52431 → 10001 Len=1
17	42.530727618	127.0.0.1	127.0.0.1	UDP	43		10001 → 52431 Len=1
18	43.531108259	127.0.0.1	127.0.0.1	UDP	43		52431 → 10001 Len=1
19	43.531392496	127.0.0.1	127.0.0.1	UDP	43		10001 → 52431 Len=1
20	44.532172139	127.0.0.1	127.0.0.1	UDP	46		52431 → 10001 Len=4

As we can see the ping messages go back and forth between the client and server.

For a much larger message count(100 in the following example) we can see the losses as well:

```
Pinging Server, Seq: 95 of size 100 bytes
Packet Lost, Seq:  95
Pinging Server, Seq: 96 of size 100 bytes
Reply from server, Seq:  96, RTT:  0.0003910064697265625 seconds
Pinging Server, Seq: 97 of size 100 bytes
Reply from server, Seq:  97, RTT:  0.0003600120544433594 seconds
Pinging Server, Seq: 98 of size 100 bytes
Reply from server, Seq:  98, RTT:  0.0005168914794921875 seconds
Pinging Server, Seq: 99 of size 100 bytes
Reply from server, Seq:  99, RTT:  0.0004818439483642578 seconds
Pinging Server, Seq: 100 of size 100 bytes
Reply from server, Seq:  100, RTT:  0.0004551410675048828 seconds

Average RTT:  0.0004103843201982214 seconds
Packets Dropped:  6
Packets Success:  94
Loss Percentage:  6.0 %
```

Code for udp_client.py

```python
from socket import AF_INET
from socket import socket
from socket import SOCK_DGRAM
from time import sleep
from datetime import datetime



server_IP_address_port = ("127.0.0.1", 10001)
UDP_client_socket = socket(family = AF_INET, type = SOCK_DGRAM)
UDP_client_socket.settimeout(1)
      You, yesterday • init …

msg_count = int(input("Enter the number of echo messages to be sent: "))

interval = float(input("Enter the interval: "))

buffer_size = int(input("Enter the packet size in bytes: "))

print('Sending server size of buffer')

buffer_msg = ("BUFFER_SIZE:" + str(buffer_size)).encode()
UDP_client_socket.sendto(buffer_msg, server_IP_address_port)
ack_msg = UDP_client_socket.recvfrom(buffer_size)

if (ack_msg[0].decode().split(':')[0] == 'BUFFER_SIZE'):
    print('Buffer size set to ', ack_msg[0].decode().split(':')[1], 'bytes')
else:
```

```
    print('[Invalid ACK] Buffer size failed to set')
    exit()


avg_rtt = 0
packets_success = 0


for msg_count in range(1, msg_count + 1):
    print("Pinging Server, Seq:", msg_count, "of size", buffer_size, "bytes")
    send_time = datetime.now().timestamp()
    UDP_client_socket.sendto(str(msg_count).encode(), server_IP_address_port)
    try:
        msg = UDP_client_socket.recvfrom(buffer_size)
        print('Reply from server, Seq: ', msg[0].decode(), end='')
    except:
        print('Packet Lost, Seq: ', msg_count)
        continue

    packets_success += 1

    recieve_time = datetime.now().timestamp()

    rtt = recieve_time - send_time
    print(', RTT: ', rtt, 'seconds')
    avg_rtt += rtt
```

Code for udp_server.py

```python
from socket import socket, AF_INET, SOCK_DGRAM      You, 14 hours ago • commit …
from time import sleep
from random import randint

UDP_server_socket = socket(family = AF_INET, type = SOCK_DGRAM)
UDP_server_socket.bind(("127.0.0.1", 10001))

buffer_size = 1024

print('Server is listening for client connection')

while True:
    msg, addr = UDP_server_socket.recvfrom(buffer_size)
    msg = msg.decode()

    if (msg.split(':')[0] == 'BUFFER_SIZE'):
        buffer_size = int(msg.split(':')[1])
        print('Buffer size set to ', buffer_size, 'bytes')
        UDP_server_socket.sendto((f"BUFFER_SIZE:{buffer_size}").encode(), addr)
        continue
    elif (msg == 'exit'):
        print('Client has exited')
        continue

    # To simulate packet loss
    if (randint(1, 100) <= 5):
        print('Packet Dropped by the Server: ', msg)
        continue
```

```python
    print('Ping from client, Seq: ', msg)
    UDP_server_socket.sendto(msg.encode(), addr)
```

Question 2

We start our server by running the following command:

```
$ python3 iperf_server.py
```

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/part_b/q2  master  ⚡
₹ py iperf_server.py
Listening on port 10001
```

We then start our client by executing the following command:

```
$ python3 iperf_client.py
```

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/part_b/q2  master  ⚡
₹ py iperf_client.py
Enter the number of messages: 300
Enter the interval (in seconds): 1
Enter the packet size (in bytes): 100
Second 1 :
Reply from Server, Seq: 1
Second 2 :
Reply from Server, Seq: 2
Reply from Server, Seq: 3
Second 3 :
Reply from Server, Seq: 4
Second 4 :
Reply from Server, Seq: 5
Second 5 :
Reply from Server, Seq: 6
Reply from Server, Seq: 7
Second 6 :
Reply from Server, Seq: 8
Reply from Server, Seq: 9
```

…

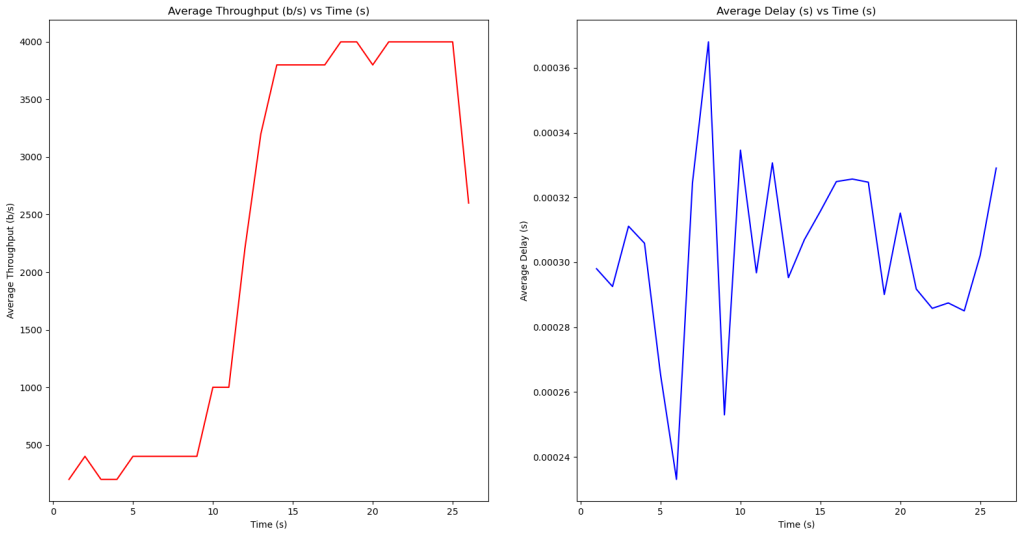
```
Second 26 :
Reply from Server, Seq: 288
Reply from Server, Seq: 289
Reply from Server, Seq: 290
Reply from Server, Seq: 291
Reply from Server, Seq: 292
Reply from Server, Seq: 293
Reply from Server, Seq: 294
Reply from Server, Seq: 295
Reply from Server, Seq: 296
Reply from Server, Seq: 297
Reply from Server, Seq: 298
Reply from Server, Seq: 299
Reply from Server, Seq: 300
```

On the server side, we have the following output:

…

```
Ping from client, Seq: 285
Ping from client, Seq: 286
Ping from client, Seq: 287
Ping from client, Seq: 288
Ping from client, Seq: 289
Ping from client, Seq: 290
Ping from client, Seq: 291
Ping from client, Seq: 292
Ping from client, Seq: 293
Ping from client, Seq: 294
Ping from client, Seq: 295
Ping from client, Seq: 296
Ping from client, Seq: 297
Ping from client, Seq: 298
Ping from client, Seq: 299
Ping from client, Seq: 300
Ping from client, Seq: exit
Client has exited
```

Using Matplotlib, we plot our graph as shown below:



On the left we have the **Average Throughput VS Time** and on the right we have **Average Delay vs Time.**

We are calculating and plotting the effective throughput and average delay per second. We also introduce a mechanism for reducing the inter-packet interval in order to increase the rate at which packets are sent. We reduce the interpacket interval by 10% every second to reduce it.

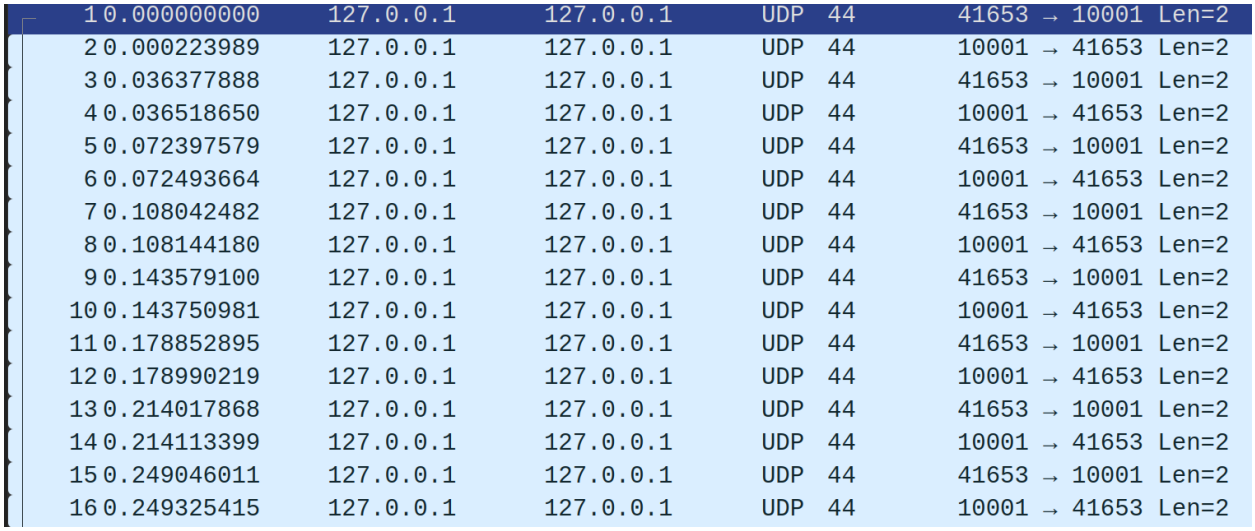
We use the total successful packets transmitted in one second approach to calculate effective throughput and average delay per second. We send the messages iteratively until all of them have been sent. In this iteration, we try to send as many messages as possible in one second.

We use the total successful packets transmitted in one second approach to calculate effective throughput and average delay per second. We send the messages iteratively until all of them have been sent. In this iteration, we try to send as many messages as possible in one second.

To accomplish this, we wait for the delay to complete in the current second before transmitting the packet. If the interval is very long (say, 5 seconds), we must sometimes wait the entire second for the delay and, in some cases, multiple seconds before sending the next packet.

Only packets with an RTT are considered when calculating effective throughput and average delay. Packets with dropped acknowledgements will not be counted as throughput, and their average delay cannot be estimated.

The Wireshark capture of the above experiment looks like this



1	0.000000000	127.0.0.1	127.0.0.1	UDP	44	41653 → 10001 Len=2
2	0.000223989	127.0.0.1	127.0.0.1	UDP	44	10001 → 41653 Len=2
3	0.036377888	127.0.0.1	127.0.0.1	UDP	44	41653 → 10001 Len=2
4	0.036518650	127.0.0.1	127.0.0.1	UDP	44	10001 → 41653 Len=2
5	0.072397579	127.0.0.1	127.0.0.1	UDP	44	41653 → 10001 Len=2
6	0.072493664	127.0.0.1	127.0.0.1	UDP	44	10001 → 41653 Len=2
7	0.108042482	127.0.0.1	127.0.0.1	UDP	44	41653 → 10001 Len=2
8	0.108144180	127.0.0.1	127.0.0.1	UDP	44	10001 → 41653 Len=2
9	0.143579100	127.0.0.1	127.0.0.1	UDP	44	41653 → 10001 Len=2
10	0.143750981	127.0.0.1	127.0.0.1	UDP	44	10001 → 41653 Len=2
11	0.178852895	127.0.0.1	127.0.0.1	UDP	44	41653 → 10001 Len=2
12	0.178990219	127.0.0.1	127.0.0.1	UDP	44	10001 → 41653 Len=2
13	0.214017868	127.0.0.1	127.0.0.1	UDP	44	41653 → 10001 Len=2
14	0.214113399	127.0.0.1	127.0.0.1	UDP	44	10001 → 41653 Len=2
15	0.249046011	127.0.0.1	127.0.0.1	UDP	44	41653 → 10001 Len=2
16	0.249325415	127.0.0.1	127.0.0.1	UDP	44	10001 → 41653 Len=2

As you can see the interval between the two packets reduce as the time goes on.

Code for iperf_client.py

```python
from datetime import datetime as dt      You, yesterday • init …
from decimal import Decimal
import matplotlib.pyplot as plt
from time import sleep
from socket import AF_INET, socket, SOCK_DGRAM, timeout


# Get parameters from user
msg_total = int(input("Enter the number of messages: "))
interval = Decimal(input("Enter the interval (in seconds): "))
buffer_size = int(input("Enter the packet size (in bytes): "))

# Create UDP socket
UDP_client_socket = socket(family = AF_INET, type = SOCK_DGRAM)
UDP_client_socket.settimeout(1)

# Send buffer size to server
set_buffer = ("B :" + str(buffer_size)).encode()
UDP_client_socket.sendto(set_buffer, ("127.0.0.1", 10001))
ack_msg = UDP_client_socket.recvfrom(buffer_size)

# Define variables
avg_throughput_lst = []
avg_delay_lst = []
time_lst = []
avg_rtt = 0
count_seconds = 0
packets_success = 0
```

```
packets_success = 0
delta = 0
seq = 1

while msg_total > 0:
    count_seconds += 1
    print("Second", count_seconds, ":")

    avg_delay = 0
    packets_success = 0

    startSecond = Decimal(dt.now().timestamp())

    while Decimal(Decimal(dt.now().timestamp()) - Decimal(startSecond)) <= 1:

        # Check if the value of delta
        if delta > 1:
            tosleep = 1
            delta -= 1
        else: tosleep = Decimal(delta)

        sleep(float(tosleep))

        # if message finished, break or if sleep is 1, break
        if msg_total == 0 or tosleep == 1: break

        UDP_client_socket.sendto(str(seq).encode(), ("127.0.0.1", 10001))
        send_time_stamp = Decimal(dt.now().timestamp())
```

```
        msg_total -= 1

        # Receive message from server
        try:
            server_msg = UDP_client_socket.recvfrom(buffer_size)
            print("Reply from Server, Seq:", server_msg[0].decode())
        except timeout:
            continue

        # Calculate RTT, delay and throughput
        rtt = Decimal(Decimal(dt.now().timestamp()) - send_time_stamp)
        avg_delay += Decimal(rtt)
        sleep_time = Decimal(max(interval - rtt, 0))
        seq += 1
        past_time = Decimal(Decimal(dt.now().timestamp()) - startSecond)
        interval = interval * Decimal(0.9)
        packets_success += 1

        if past_time > 1:
            delta = Decimal(tosleep)
            break
        elif past_time + sleep_time > 1 and past_time <= 1:
            tosleep = Decimal(1 - past_time)
            delta = Decimal(sleep_time - tosleep)
        else:
            tosleep = Decimal(sleep_time)

        sleep(float(tosleep))


    if packets_success == 0: avg_delay = 0
    else: avg_delay = avg_delay / packets_success
    avg_throughput = packets_success * buffer_size * 2

    avg_delay_lst.append(avg_delay)
    avg_throughput_lst.append(avg_throughput)
    time_lst.append(count_seconds)

UDP_client_socket.sendto('exit'.encode(), ("127.0.0.1", 10001))
```

```python
# Plot for Average Throughput vs Time
plt.subplot(1, 2, 1)
plt.plot(time_lst, avg_throughput_lst, 'red')
plt.xlabel('Time (s)')
plt.ylabel('Average Throughput (b/s)')
plt.title('Average Throughput (b/s) vs Time (s)')


# Plot for Average Delay vs Time
plt.subplot(1, 2, 2)
plt.plot(time_lst, avg_delay_lst, 'blue')
plt.xlabel('Time (s)')
plt.ylabel('Average Delay (s)')
plt.title('Average Delay (s) vs Time (s)')

plt.show()
```

Code for iperf_server.py

```python
from socket import socket, AF_INET, SOCK_DGRAM      You, yesterday • improve …

buffer_size = 1024
UDP_server_socket = socket(family = AF_INET, type = SOCK_DGRAM)
UDP_server_socket.bind(("127.0.0.1", 10001))

print("Listening on port", 10001)

while True:
    msg, addr = UDP_server_socket.recvfrom(buffer_size)
    client_msg = msg.decode()
    print("Ping from client, Seq:", client_msg)

    if client_msg[0] == 'B':
        buffer_size = int(client_msg.split(':')[1])
        UDP_server_socket.sendto((f"Buffer Size set to {buffer_size}").encode(), addr)

    elif client_msg == 'exit': print("Client has exited")
    else: UDP_server_socket.sendto(msg, addr)
```

Part 3

We start our server by running the following command:

```
$ python3 server.py
```

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_c  master  ⚡
₹ py server.py                                                                                          ♻
Enter host [ip6-localhost, localhost]: localhost
Enter port [8000]: 3000
Socket: (<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('127.0.0.1', 3000))
```

We enter the host as localhost (IPv4) and enter port as 3000.

We see our socket details in the output

Similarly, we start our client by running the following command:

```
$ python3 client.py
```

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_c  master  ⚡
₹ py client.py                                                                                          ♻
Enter host [ip6-localhost, localhost]: localhost
Enter port [8000]: 3000
Enter message: hello
Socket: (<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('127.0.0.1', 3000))
Sending message: hello
Reply from Server: hello
```

We enter the same details (localhost, 3000) in the input and send a message to the server. If our details are correct and the server is up, we get the same message in the response.

On the server side, we see the following output:

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_c  master  ⚡
₹ py server.py                                                                                          ♻
Enter host [ip6-localhost, localhost]: localhost
Enter port [8000]: 3000
Socket: (<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('127.0.0.1', 3000))
Connected to: 127.0.0.1:54208
Received message: hello
Disconnected from: 127.0.0.1:54208
```

Now, lets try with IPv6

Starting server

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_c  master  ⚡
₹ py server.py
Enter host [ip6-localhost, localhost]: ip6-localhost
Enter port [8000]: 3000
Socket: (<AddressFamily.AF_INET6: 10>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('::1', 3000, 0, 0))
```

Starting the client

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_c  master  ⚡
₹ py client.py
Enter host [ip6-localhost, localhost]: ip6-localhost
Enter port [8000]: 3000
Enter message: hello
Socket: (<AddressFamily.AF_INET6: 10>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('::1', 3000, 0, 0))
Sending message: hello
Reply from Server: hello
```

We can see that the IPv6 protocol is also working. We send our message and get the same response from the server which means that our configuration is correct.

We will essentially use the functionality of AF UNSPEC to make the echo client-server protocol independent. When the getaddrinfo socket function is called to establish a socket, this prevents the protocol from distinguishing between IPv4 and IPv6 addressing because it can accept both IPv4 and IPv6 addresses.

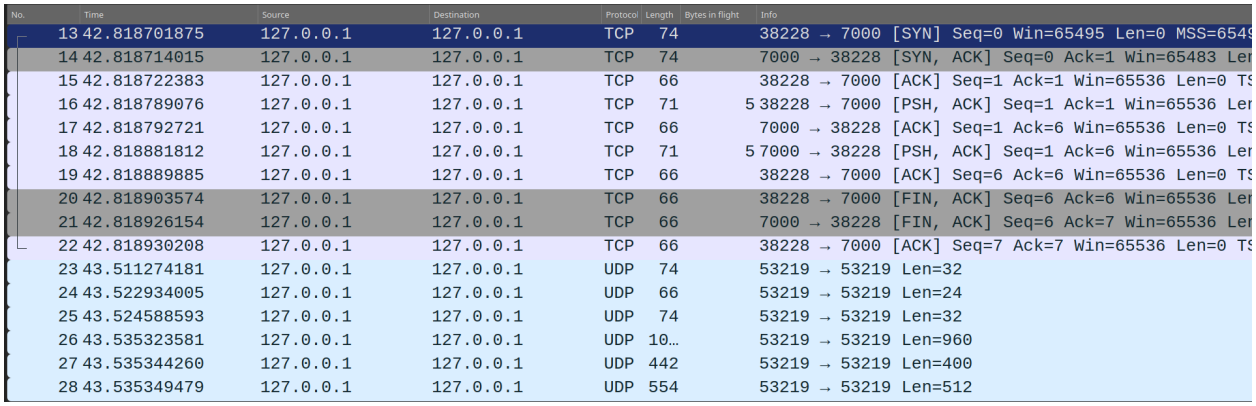
Another flaw is the length of the IPv6 address. Rather than classifying it as an IPv4 address length, we store it in a variable that can hold any length greater than IPv6. As a result, storing the variable with an IPv6 address will not be a problem.

getaddrinfo converts the specified hostname to an address. The AF UNSPEC will allow translation of both addresses if the input is ip6-localhost or localhost.

We can establish sockets and create the necessary socket address to bind to because we can translate both types of addresses.

We only need to transfer data once the sockets are established. We can establish connections on both sides by using both types of addressing.

The Wireshark capture of the above question looks like this:



No.	Time	Source	Destination	Protocol	Length	Bytes in flight	Info
13	42.818701875	127.0.0.1	127.0.0.1	TCP	74		38228 → 7000 [SYN] Seq=0 Win=65495 Len=0 MSS=654
14	42.818714015	127.0.0.1	127.0.0.1	TCP	74		7000 → 38228 [SYN, ACK] Seq=0 Ack=1 Win=65483 Le
15	42.818722383	127.0.0.1	127.0.0.1	TCP	66		38228 → 7000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TS
16	42.818789076	127.0.0.1	127.0.0.1	TCP	71	5	38228 → 7000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Le
17	42.818792721	127.0.0.1	127.0.0.1	TCP	66		7000 → 38228 [ACK] Seq=1 Ack=6 Win=65536 Len=0 TS
18	42.818881812	127.0.0.1	127.0.0.1	TCP	71	5	7000 → 38228 [PSH, ACK] Seq=1 Ack=6 Win=65536 Le
19	42.818889885	127.0.0.1	127.0.0.1	TCP	66		38228 → 7000 [ACK] Seq=6 Ack=6 Win=65536 Len=0 TS
20	42.818903574	127.0.0.1	127.0.0.1	TCP	66		38228 → 7000 [FIN, ACK] Seq=6 Ack=6 Win=65536 Le
21	42.818926154	127.0.0.1	127.0.0.1	TCP	66		7000 → 38228 [FIN, ACK] Seq=6 Ack=7 Win=65536 Le
22	42.818930208	127.0.0.1	127.0.0.1	TCP	66		38228 → 7000 [ACK] Seq=7 Ack=7 Win=65536 Len=0 TS
23	43.511274181	127.0.0.1	127.0.0.1	UDP	74		53219 → 53219 Len=32
24	43.522934005	127.0.0.1	127.0.0.1	UDP	66		53219 → 53219 Len=24
25	43.524588593	127.0.0.1	127.0.0.1	UDP	74		53219 → 53219 Len=32
26	43.535323581	127.0.0.1	127.0.0.1	UDP	10…		53219 → 53219 Len=960
27	43.535344260	127.0.0.1	127.0.0.1	UDP	442		53219 → 53219 Len=400
28	43.535349479	127.0.0.1	127.0.0.1	UDP	554		53219 → 53219 Len=512

Code for client.py

```python
from socket import AF_UNSPEC
from socket import getaddrinfo
from socket import SOCK_STREAM
from socket import socket

host = input('Enter host [ip6-localhost, localhost]: ')
port = input('Enter port [8000]: ')
message = input('Enter message: ')

if port == '': port = 8000      You, yesterday • init
if host == '': host = 'ip6-localhost'


server_socket = None

for addr_family, socket_type, protocol, cn, socket_address in getaddrinfo(host, port, AF_UNSPEC, SOCK_STREAM):
    try:
        server_socket = socket(addr_family, socket_type, protocol)
        server_socket.connect(socket_address)
        print(f'Socket: {addr_family, socket_type, protocol, cn, socket_address}')
        break
    except:
        print('Unable to create socket.')
        exit()

print(f'Sending message: {message}')

server_socket.send(message.encode())
```

```python
print(f'Sending message: {message}')

server_socket.send(message.encode())

response = server_socket.recv(1024)

print(f'Reply from Server: {response.decode()}')

server_socket.close()
```

Code for server.py

```python
from socket import SOCK_STREAM, socket, AF_UNSPEC, getaddrinfo      You, yesterday • init

host = input('Enter host [ip6-localhost, localhost]: ')
port = input('Enter port [8000]: ') # 8000

if host == '': host = 'ip6-localhost'
if port == '': port = 8000

server_socket = None

for addr_family, socket_type, protocol, cn, socket_address in getaddrinfo(host, port, AF_UNSPEC, SOCK_STREAM):
    try:
        server_socket = socket(addr_family, socket_type, protocol)
        server_socket.bind(socket_address)
        server_socket.listen(1)
        print(f'Socket: {addr_family, socket_type, protocol, cn, socket_address}')
        break
    except:
        print('Unable to create socket.')
        exit()

client, address = server_socket.accept()

print(f'Connected to: {address[0]}:{address[1]}')

while 1:
    message = client.recv(1024)
    if message:
```

```python
while 1:
    message = client.recv(1024)
    if message:
        print(f'Received message: {message.decode()}')
        client.send(message)
    else:
        print(f'Disconnected from: {address[0]}:{address[1]}')
        client.close()
        break
```

Part 4

For this part, I have created a chat application using sockets.

There is a central server and multiple clients connect to it. Any message sent by any clients is sent to all other clients and server acts as a mediator between two clients.

The server allocates 1 thread per user connected to the server. The thread closes when the user leaves the chat. For this program, we allow 100 concurrent users, however we can change it according to our needs

Also the chat application is secured by Secure Socket Layer (SSL). We apply SSL on top of our socket to ensure that no one snooping on the network can read our message. Note that this is

server-side encryption and not End-to-End encryption(E2EE) like we see on chatting applications like WhatsApp, iMessage, etc.

SSL stands for Secure Sockets Layer and refers to a protocol for encrypting, securing, and authenticating Internet communications. Although SSL was replaced by a more recent protocol called TLS (Transport Layer Security) some time ago, the term "SSL" is still widely used to refer to this technology.

SSL/TLS is most commonly used to secure communications between a client and a server, but it can also be used to secure email, VoIP, and other communications over insecure networks.

These are the fundamental principles to grasp in order to comprehend how SSL/TLS works:

- The secure communication process starts with a TLS handshake, in which the two communicating parties establish a secure connection and exchange the public key.
- The two parties generate session keys during the TLS handshake, and the session keys encrypt and decrypt all communications after the TLS handshake.
- Each new session uses a different session key to encrypt communications.
- TLS ensures that the party on the server side, or the website with which the user is interacting, is who they claim to be.
- TLS also ensures that data has not been tampered with because it includes a message authentication code (MAC) with transmissions.

For SSL, we need two things key and certificate.

Our certificate looks like this:

```
-----BEGIN CERTIFICATE-----
MIIDtzCCAp+gAwIBAgIUHL5KUcBMKTrMCn+V7BsyPhm2yHYwDQYJKoZIhvcNAQEL
BQAwazELMAkGA1UEBhMCSU4xCzAJBgNVBAgMAkRMMQ4wDAYDVQQHDAVEZWxoaTET
MBEGA1UECgwKSUlUIEJoaWxhaTEqMCgGCSqGSIb3DQEJARYbY2hhaXRhbnlhLmJp
c2h0MTBAZ21haWwuY29tMB4XDTIyMTEyMTE3NTcyNFoXDTIzMTEyMTE3NTcyNFow
azELMAkGA1UEBhMCSU4xCzAJBgNVBAgMAkRMMQ4wDAYDVQQHDAVEZWxoaTETMBEG
A1UECgwKSUlUIEJoaWxhaTEqMCgGCSqGSIb3DQEJARYbY2hhaXRhbnlhLmJpc2h0
MTBAZ21haWwuY29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAyF6+
ZH1ScS7rod3xdEJ5n+IWIqkNVD6vylnFcXUZhZ6m7gAzYr/fFqArEKLJe2rj/ptD
PH1P+3gFzOZhg6pr0ar/amUHt4uOn5SZiUPnWtyh8lSQNpDcjq47Qk4yN4ILkeJy
nL6sCpDw2EEDOwk5U8T5U9D542gbShmP4Sty690wpAJbSTrEB27xNqO7DbWXp3Gb
ac5s5WBo19Giu7jQXj/HAlIs1dhBp5bq4iJlmSP87rkNBKJ1C6TQztl2cq2vslR3
HwzqyaHptBysMkbkC+lkfSxo6KOv9C+6tjLYLOOcgyEj3AcTcsbGGfdW6wNbK5gD
NDy95qzuTKciCnwFiwIDAQABo1MwUTAdBgNVHQ4EFgQUcoLSrQeQIQri2kQ4YQAQ
6K2ibDEwHwYDVR0jBBgwFoAUcoLSrQeQIQri2kQ4YQAQ6K2ibDEwDwYDVR0TAQH/
BAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAxsnScONmwraCcjh0Q+i4TImEWUl6
8Kb98ScnPe12Wz91gTtFxPvwoPrnRDnxP6Up1ZlCNLwzRsZGuGz028DFVeFNERP9
7Ow9rhzbRF7ISqTlVVaaRcWik7LUkNiOOA4e0qFP2ng2B29SGOO/LHL7+oj1LHTy
UqnJcueJz0Y2oY1iWYOguhfIzyY5WUgFkuwsCAO+jd8l1j2CsQ4iPloBF2xMteaK
cR7BymGKgDRMHwbgJOJkhqM0495ywIyWxk7yQhJo2vZwB6nU1GVytqgHtLpz+oem
FtMIk1wZWQaVKVWscdaKjX3gubohvEGE6qNM3atAZoNZxxXLi7gaMT3Mqg==
-----END CERTIFICATE-----
```

Subject Name

C (Country):	IN
ST (State):	DL
L (Locality):	Delhi
O (Organization):	IIT Bhilai
EMAIL (Email Address):	chaitanya.bisht10@gmail.com

Issuer Name

C (Country):	IN
ST (State):	DL
L (Locality):	Delhi
O (Organization):	IIT Bhilai
EMAIL (Email Address):	chaitanya.bisht10@gmail.com

Issued Certificate

Version:	3
Serial Number:	1C BE 4A 51 C0 4C 29 3A CC 0A 7F 95 EC 1B 32 3E 19 B6 C8 76
Not Valid Before:	2022-11-21
Not Valid After:	2023-11-21

Certificate Fingerprints

SHA1:	C4 70 8B 2D D3 8A 5F 72 3A 1B 92 FF 21 36 0A C1 C7 DB 02 22
MD5:	7B 7E 1D D5 97 5A 89 DA 03 49 EC 6C 78 48 A7 89

And our key looks like this:

```
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQDIXr5kfVJxLuuh
3fF0Qnmf4hYiqQ1UPq/KWcVxdRmFnqbuADNiv98WoCsQosl7auP+m0M8fU/7eAXM
5mGDqmvRqv9qZQe3i46flJmJQ+da3KHyVJA2kNyOrjtCTjI3gguR4nKcvqwKkPDY
QQM7CTlTxPlT0PnjaBtKGY/hK3Lr3TCkAltJOsQHbvE2o7sNtZencZtpzmzlYGjX
0aK7uNBeP8cCUizV2EGnluriImWZI/zuuQ0EonULpNDO2XZyra+yVHcfDOrJoem0
HKwyRuQL6WR9LGjoo6/0L7q2Mtgs45yDISPcBxNyxsYZ91brA1srmAM0PL3mrO5M
pyIKfAWLAgMBAAECggEAWKALb5A89GIQCCcOguFaQX0zVD5Y7m/RulttqQrl8IOl
izTchaufZbcPhUqzeprjYHzVRv6Yb359dk1aSW4KRZFwauNn/+fC/FOye5Pk03wu
R2ze4ru9SoDHJkNHE+U579ts3Wa62uuUE9rhrm2chTBlJY0T0mF8XJCk4rAgozEI
jDdOn+Qg7jnKNtywHM1AU5pM1aaMh4vzC+TJ3uqiSBR6Z3JoMuykZx76uRdc54Hd
ibtjoneIpT52Pp9/PqlSAcZg++DSj2eEF+zVE7RvAN8x1ep4YRJ5/i9aCbdAqAKE
tFsoyibcbQmkzx8t68fZy7gtFmilEFeJc0H1iHF++QKBgQDWr3NgX9CgcHoshJ16
i2qTn93IQUF/o1GCEAXkW8PXSb9IObAK+gXnoCVM8rZ1piRrOkyMxHFSReaQgOmJ
k49Ar/89nDTsZeMhlrnHn7SQGJRr2SIuaT0rysgq6dw6mnxjdX0AYv7GRLLCF/fH
rP+/oNVdHsXrqTZrwQ9nTGIDCQKBgQDu7gvw+qfZlpr3g6o2ILPy/+JZh0pQfbts
AyGMEKNCPes7Slv81aVBdTOPY6oBHluqvIuBOBQtPDTewazFxOgfnTJh7PsqT4DF
p+BeiKP+SIbQVVT2KPODZpuCqlJYvxxQJyScqcoKG0DiiNPHDFSoMmFj7VhyJ5dS
BeP/zO8E8wKBgQDC/p1nHeAauc7efJ2k3tiwuXXyHANn8WlpHzCa2BarhtpsuakC
b1HJrpXyrEm51mwh0NQSz6K2PbNL8SstwI9PDAxjY2xWg7ynl4WqoY6ZixnUU2ZO
jVv4WLky8TMNL6BnniEppsEOSPuse6SdAq9Q2L6zU+4lPI6nlLVu99vWMQKBgCNV
mPZ+WyxWlXxdQGLY20Poi7TpyRe2QG1s2R05qKs3NP6DtK7jeZkAmCtwdcfMkYni
```

The server can be started using the following command

```
$ python3 server.py 127.0.0.1 3000
```

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_d  master  ⚡
₹ py server.py 127.0.0.1 3000
Server started on port 3000
```

Now we setup two clients named Chaitanya and Aarav. Ensure that we enter the same IP and the port number

```
$ python3 client.py localhost 3000 Chaitanya
$ python3 client.py localhost 3000 Aarav
```

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_d  master  ⚡
₹ py client.py localhost 3000 Chaitanya
```

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_d  master  ⚡
₹ py client.py localhost 3000 Aarav
```

As we can see on the server side, two sockets are connected to server

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_d  master  ⚡
₹ py server.py 127.0.0.1 3000
Server started on port 3000
<127.0.0.1:57810> joined the chat
<127.0.0.1:56262> joined the chat
```

Let's send a message to Aarav from Chaitanya

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_d  master  ⚡
₹ py client.py localhost 3000 Chaitanya
<127.0.0.1:56262> has joined the chat
<You>: Hi Aarav!
```

Now let's see what we see on Aarav's screen

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_d  master  ⚡
₹ py client.py localhost 3000 Aarav
<Chaitanya> Hi Aarav!
```

Now Aarav will reply to Chaitanya

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_d  master  ⚡
₹ py client.py localhost 3000 Aarav
<Chaitanya> Hi Aarav!
<You>: Hi Chaitanya! Nice to meet you
```

Now we see the reply on Chaitanya's screen

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/pa
rt_d  master  ⚡
₹ py client.py localhost 3000 Chaitanya
<127.0.0.1:56262> has joined the chat
<You>: Hi Aarav!
<Aarav> Hi Chaitanya! Nice to meet you
```

We can exit the chat application by sending \exit command

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/part_d  master  ⚡
₹ py client.py localhost 3000 Aarav
<Chaitanya> Hello
<You>: Hi!
\exit
```

And we can see the connection terminate at the server as well

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/part_d  master  ⚡
₹ py server.py 127.0.0.1 3000
Server started on port 3000
<127.0.0.1:49148> joined the chat
<127.0.0.1:49158> joined the chat
<Chaitanya> Hello
<Aarav> Hi!
<127.0.0.1:49158> has left the chat
```

The Wireshark capture of the application looks like this

No.	Time	Source	Destination	Protocol	Length	Bytes in flight	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74		51734 → 3000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PE
2	0.000013213	127.0.0.1	127.0.0.1	TCP	74		3000 → 51734 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65
3	0.000020794	127.0.0.1	127.0.0.1	TCP	66		51734 → 3000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=41324
4	0.000142708	127.0.0.1	127.0.0.1	TLS…	583	517	Client Hello
5	0.000146279	127.0.0.1	127.0.0.1	TCP	66		3000 → 51734 [ACK] Seq=1 Ack=518 Win=65024 Len=0 TSval=413
6	0.001168856	127.0.0.1	127.0.0.1	TLS…	1577	1511	Server Hello, Change Cipher Spec, Application Data, Applic
7	0.001175758	127.0.0.1	127.0.0.1	TCP	66		51734 → 3000 [ACK] Seq=518 Ack=1512 Win=64256 Len=0 TSval=
8	0.001772384	127.0.0.1	127.0.0.1	TLS…	146	80	Change Cipher Spec, Application Data
9	0.001828943	127.0.0.1	127.0.0.1	TLS…	321	255	Application Data
10	0.049547494	127.0.0.1	127.0.0.1	TCP	66		51734 → 3000 [ACK] Seq=598 Ack=1767 Win=65536 Len=0 TSval=
11	0.049566768	127.0.0.1	127.0.0.1	TLS…	321	255	Application Data

As we can see, there is the TCP handshake and the TLS handshake. This verifies that the messages being sent are sent through a secure channel and no one can eavesdrop on our messages.

Bonus Addition:

I also added a file share functionality to the chat application, where you could enter a file name present in your local machine and it would be sent other clients.

We send the file using \send command and enter the file name to send the file. Note that the file transfer happens on the same socket which means that it is also sent using SSL.

Let's send a file named hello.txt, these are the contents of the file hello.txt:

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/part_4  f
ile_transfer  ⚡
₹ cat hello.txt
Hello There!
This is a sample file
By Chaitanya
CS301
Computer Networks%
```

Now let's send it to the client by entering \send hello.txt

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/part_4  f
ile_transfer  ⚡
₹ py client.py localhost 3000 Chaitanya
<127.0.0.1:50748> has joined the chat
<You>: hello
<Aarav> Hi send me the file please!
\send
hello.txt
```

On the receiver side we see the following output:

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/part_4  f
ile_transfer  ⚡
₹ py client.py localhost 3000 Aarav
<Chaitanya> hello
<You>: Hi send me the file please!
File received hello_download.txt
```

Now let's see the contents of the downloaded file: hello_download.txt

```
[chaitanya@chait360] /media/chaitanya/mydata/IIT/Semester 5/CS301 Computer Networks/Assignments/Assignment 3/part_4  f
ile_transfer  ⚡
₹ cat hello_download.txt
Hello There!
This is a sample file
By Chaitanya
CS301
Computer Networks%
```

As we can see, the contents of the file are the same

Code for client.py

```
from socket import socket, AF_INET, SOCK_STREAM
from select import select
import sys
import ssl
import warnings; warnings.filterwarnings("ignore")

client_socket = socket(AF_INET, SOCK_STREAM)

client_socket = ssl.wrap_socket(
    client_socket, server_side=False, keyfile="host.key", certfile="host.cert"
)

if len(sys.argv) != 4:
    print ("Correct usage: script, IP address, port number, your name")
    exit()

ip_addr = str(sys.argv[1])
port = int(sys.argv[2])
name = str(sys.argv[3])

client_socket.connect((ip_addr, port))

while True:

    sockets_list = [sys.stdin, client_socket]
    r, w, e = select(sockets_list,[],[])
```

```
for sock in r:
    if sock == client_socket:
        message = sock.recv(2048)
        if (message.decode().split(':')[0] == 'SOF'):
            file_name = message.decode().split(':')[1]
            file_name = file_name.split('.')[0] + '_download.' + file_name.split('.')[1]
            f = open(file_name, 'wb')
            while (True):
                message = sock.recv(2048)
                if (message.decode() == 'EOF'):
                    f.close()
                    break
                f.write(message)
            sys.stdout.write(f'File received {file_name}\n')
        else:
            print(message.decode(), end='')
    else:
        message = sys.stdin.readline()
        if (message.strip('\n') == "\exit"):
            client_socket.send(("\exit").encode())
            client_socket.close()
            exit()
        elif (message.strip('\n') == "\send"):
            file_name = sys.stdin.readline()
            client_socket.send((f"SOF:{file_name}").encode())
            file = open(file_name.strip('\n'), 'rb')      You, 1 hour ago • added download
            while True:
                data = file.readline()
```

```
            while True:
                data = file.readline()
                if not data:
                    break
                client_socket.send(data)
            file.close()
            client_socket.send(("EOF").encode())
        else:
            to_send = name + ":" + message
            client_socket.send(to_send.encode())
            sys.stdout.write("\033[1A")
            sys.stdout.write(f"<You>: {message}")
            sys.stdout.flush()
```

Code for server.py

```
from _thread import start_new_thread as thread
from socket import socket, AF_INET, SOCK_STREAM, SOL_SOCKET, SO_REUSEADDR
import sys
from utils import *
import ssl
import warnings; warnings.filterwarnings("ignore")

server_socket = socket(AF_INET, SOCK_STREAM)
server_socket.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)

server_socket = ssl.wrap_socket(
    server_socket, server_side=True, keyfile="host.key", certfile="host.cert"
)

if len(sys.argv) != 3:
    print ("Usage: python3 server.py <hostname> <port>")
    exit()

ip = str(sys.argv[1])
port = int(sys.argv[2])

server_socket.bind((ip, port))
server_socket.listen(100)

print("Server started on port", port)

while 1:
    conn, addr = server_socket.accept()
    list_of_clients.append(conn)
    print('<'+addr[0] + ':' + str(addr[1]) + '> joined the chat')
    thread(client_thread,(conn,addr))
```

Code for utils.py

```
list_of_clients = []

def broadcast(message, connection):
    for clients in list_of_clients:
        if clients == connection:
            continue
        try:
            clients.send(message)
        except:
            clients.close()
            remove(clients)

def remove(connection):
    if connection in list_of_clients: list_of_clients.remove(connection)
```

```
def client_thread(conn, addr):
    announcement = '<' + addr[0] + ':' + str(addr[1]) + '>' + ' has joined the chat\n'
    broadcast(announcement.encode(), conn)
    while 1:
            try:
                message = conn.recv(2048).decode()
                if message:
                    if message == "\exit":
                        announcement = '<' + addr[0] + ':' + str(addr[1]) + '>' + ' has left the chat\n'
                        print(announcement)
                        broadcast(announcement.encode(), conn)
                        conn.close()
                        remove(conn)
                        break
                    if (message.split(':')[0] == 'SOF'):
                        broadcast(message.encode(), conn)
                        while (True):
                            message = conn.recv(2048).decode()
                            if (message == 'EOF'):
                                broadcast(message.encode(), conn)
                                break
                            broadcast(message.encode(), conn)      You, 1 hour ago • added download functionality
                    name = message.split(":")[0]
                    message_data = message.split(":")[1]
                    print ("<" + name + "> " + message_data, end='')
                    message_to_send = "<" + name + "> " + message_data
                    broadcast(message_to_send.encode(), conn)

                else:
                    remove(conn)
            except:
                continue
```