Computer Networks Labsheet 4

Socket Programming

Part1: Understanding the UDP socket programming

1. Understand the existing UDP client and server Python program used to send messages among each other. Change the port number if required.

   My Answer:

```
serverPort = 12000
```

   changes port to 12000

2. Execute the client and server code simultaneously. Have the screenshot in such a way that the command prompt captures your name and roll number.

   Sample screenshots:

   @client

```
D:\Academics\July to December 2024\S5 CSE Computer Networks\lab\ICN_Dhivvya_1468_socket>python ClientUDP.py
Input lowercase sentence: amrita vishwa vidyapeetham
Traceback (most recent call last):
  File "ClientUDP.py", line 7, in <module>
    modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
ConnectionResetError: [WinError 10054] An existing connection was forcibly closed by the remote host

D:\Academics\July to December 2024\S5 CSE Computer Networks\lab\ICN_Dhivvya_1468_socket>python ClientUDP.py
Input lowercase sentence: amrita vishwa vidyapeetham
AMRITA VISHWA VIDYAPEETHAM
received from  ('127.0.0.1', 12000)
```

   @server

```
D:\Academics\July to December 2024\S5 CSE Computer Networks\lab\ICN_Dhivvya_1468_socket>python ServerUDP.py
the server is ready to receive
AMRITA VISHWA VIDYAPEETHAM
above message received from  ('127.0.0.1', 60915)
```

   My Answer:

   client

```
PS C:\Users\exam\Downloads\ls4> python ClientUDP.py
Type lowercase sentence (or "quit" to exit): helloo
HELLOO
received from  ('127.0.0.1', 12000)
Type lowercase sentence (or "quit" to exit): my name is tanmayi
MY NAME IS TANMAYI
received from  ('127.0.0.1', 12000)
Type lowercase sentence (or "quit" to exit): how are u
HOW ARE U
received from  ('127.0.0.1', 12000)
Type lowercase sentence (or "quit" to exit):
```

server

```
PS C:\Users\exam\Downloads\ls4> & "C:/Program Files/Python310/python.exe" c:/Users/exam/Downloads/ls4/ServerUDP.py
the server is ready to receive
HELLOO
above message received from  ('127.0.0.1', 62996)
MY NAME IS TANMAYI
above message received from  ('127.0.0.1', 62996)
HOW ARE U
above message received from  ('127.0.0.1', 62996)
```

Continuous chatting between client and server

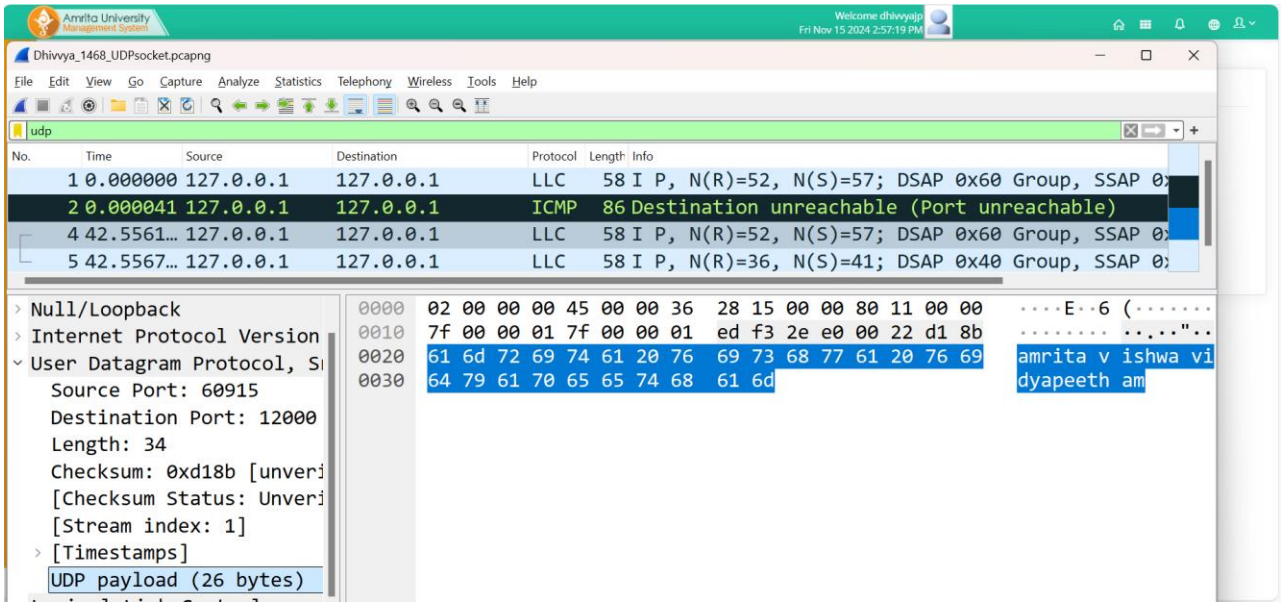
client

```
PS C:\Users\exam\Downloads\ls4> python ClientUDP.py
Type lowercase sentence (or "quit" to exit): hi
hello
received from  ('127.0.0.1', 12000)
Type lowercase sentence (or "quit" to exit): this is Tanmayi roll.31
this is server
received from  ('127.0.0.1', 12000)
Type lowercase sentence (or "quit" to exit):
```

Server

```
PS C:\Users\exam\Downloads\ls4> & "C:/Program Files/Python310/python.exe" c:/Users/exam/Downloads/ls4/ServerUDP.py
Server is ready to chat.
Client: hi
You (Server): hello
Client: this is Tanmayi roll.31
You (Server): this is server
```

3. Try to capture the wireshark frames filtering for udp captured in the loopback interface is shown below. Please make sure your identity is visible. You can use AUMS login. Always follow this protocol in all your lab submissions in this course.

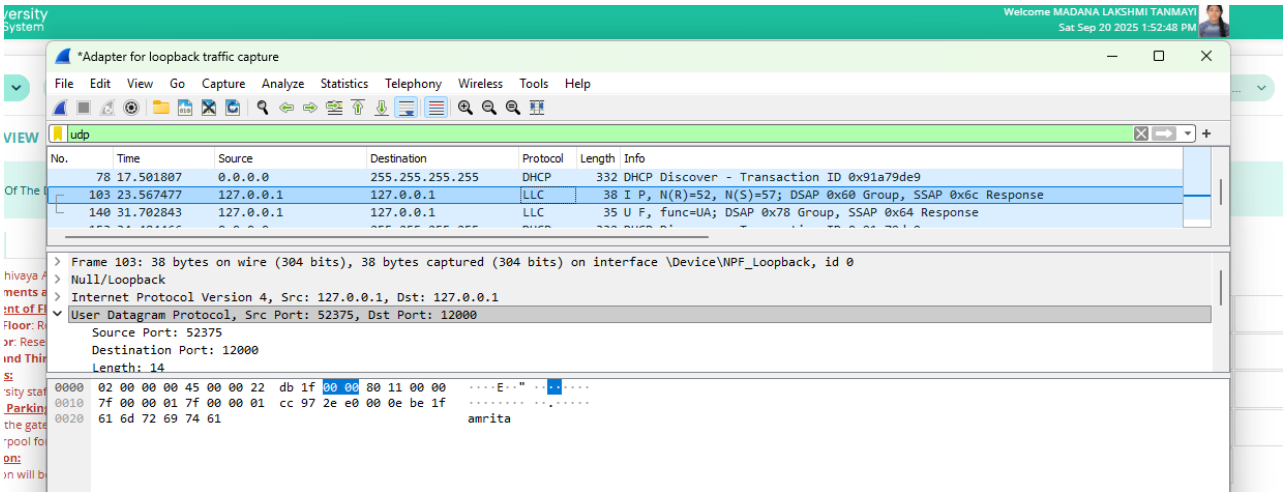


client

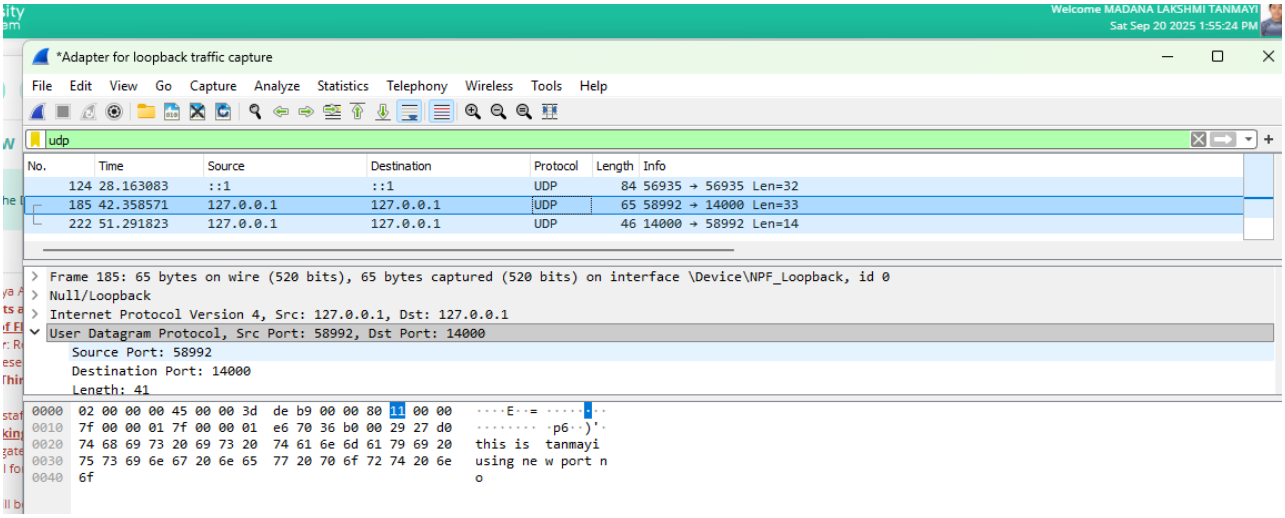
```
received from  ('127.0.0.1', 12000)
Type lowercase sentence (or "quit" to exit): amrita
yes
received from  ('127.0.0.1', 12000)
```

Server

```
Client: this is Tanmayi roll.31
You (Server): this is server
Client: amrita
You (Server): yes
```



4. Try to change the local host of client/server host to remote host and use right interface to capture the messages in wireshark.



Part2: Understanding the TCP socket programming

1. Understand the existing TCP client and server Python program used to send messages among each other. Change the port number if required.

   Port no : 13000

2. Execute the client and server code simultaneously in the same local host. Have the screenshot in such a way that the command prompt captures your name and roll number.

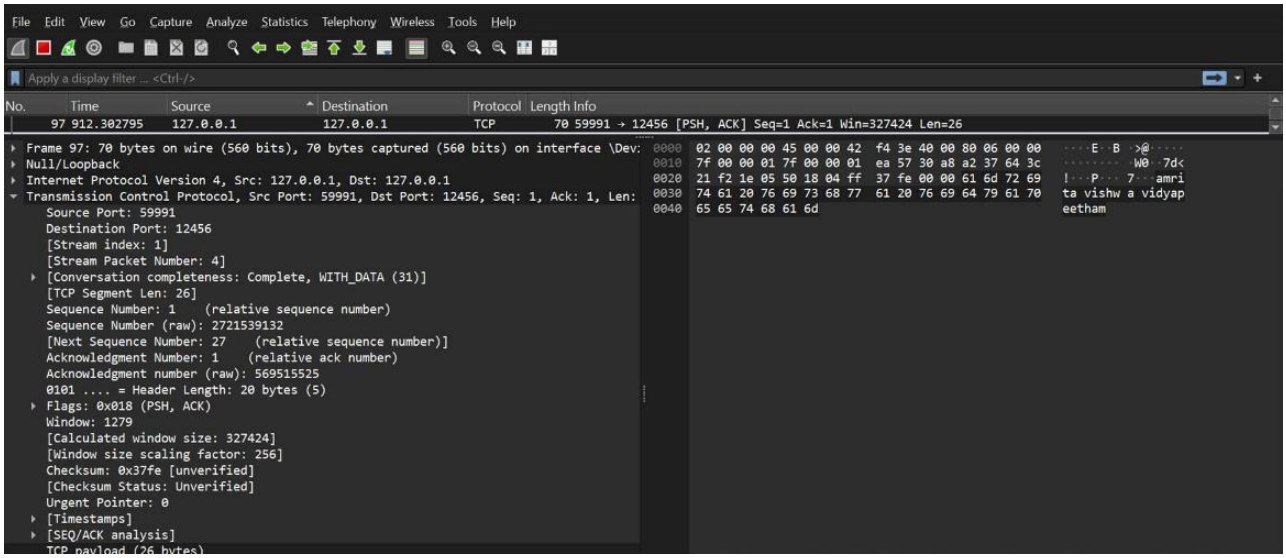
   server

```
PS C:\Users\exam\Downloads\ls4> & "C:/Program Files/Python310/python.exe" c:/Users/exam/Downloads/ls4/serverTCP.py
The server is ready to receive
sending  TANMAYI 31
to  ('127.0.0.1', 52581)
```

   client

```
PS C:\Users\exam\Downloads\ls4> python ClientTCP.py
Input lowercase sentence:tanmayi 31
From Server: TANMAYI 31
PS C:\Users\exam\Downloads\ls4>
```

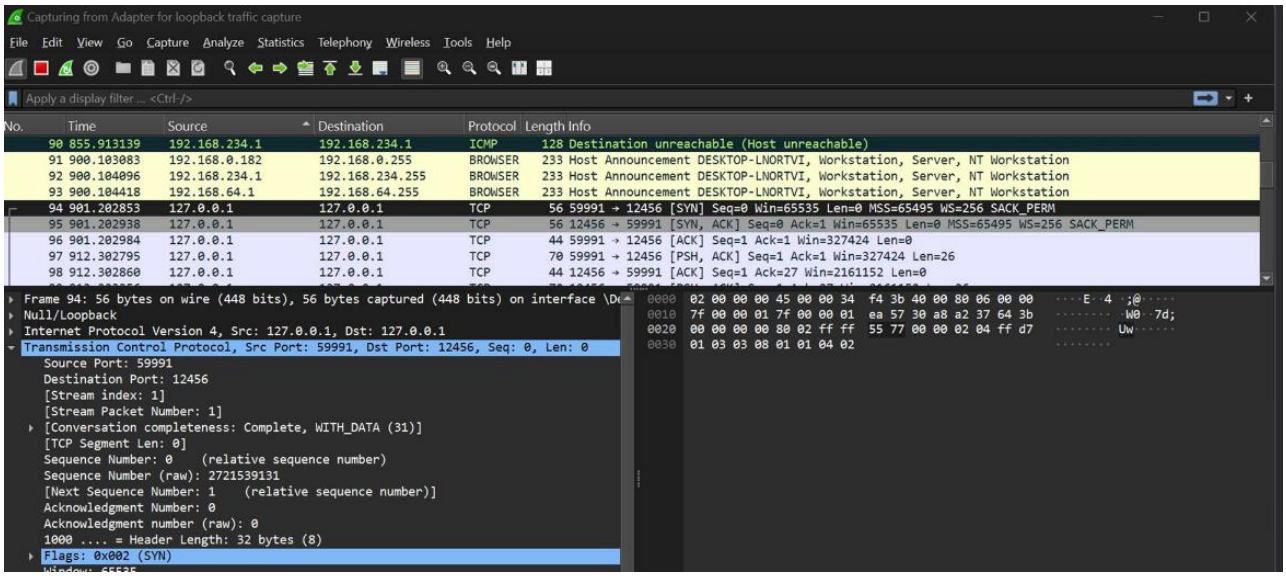
3. Try to capture the relevant wireshark frames and screenshot with proper identity rules specified earlier.



4. Execute at least one client and server process simultaneously in the different host. Have the screenshot in such a way that the command prompt captures your name and roll number.

5. Guess the frames used for TCP connection establishments before the data transfer. Provide the screenshots of 3 handshake frames(SYN, SYN ACK, ACK) in the list pane and identify the sender & receiver process for each of them.

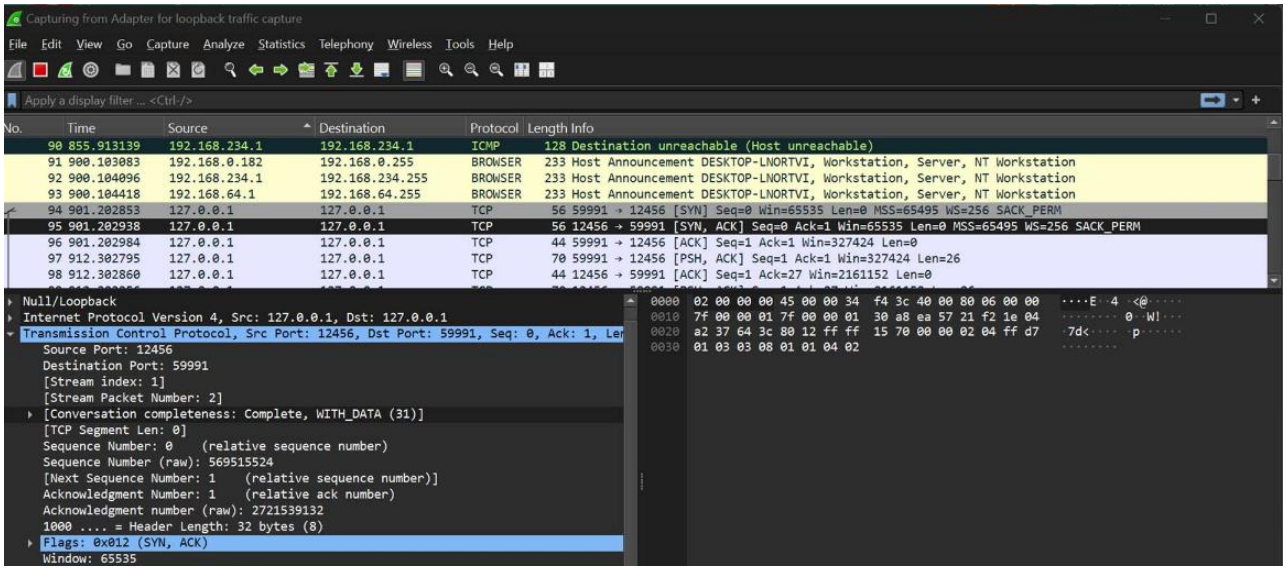
   Syn:



Sender: Client

Receiver: Server

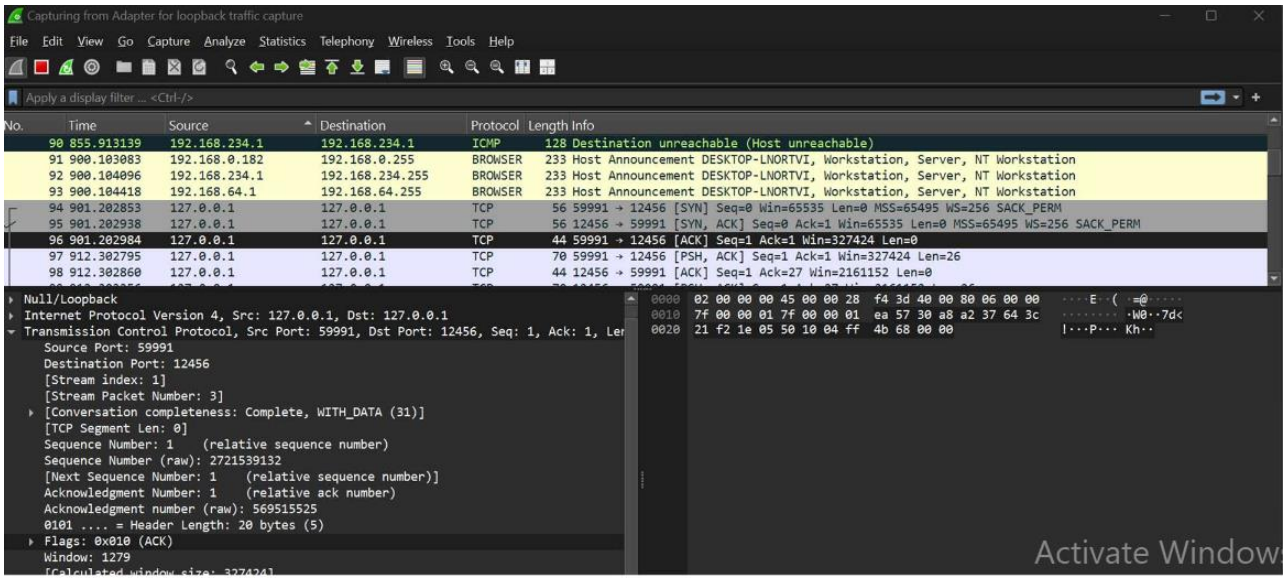
SYN-ACK:



Sender: Server

Receiver: Client

Ack:

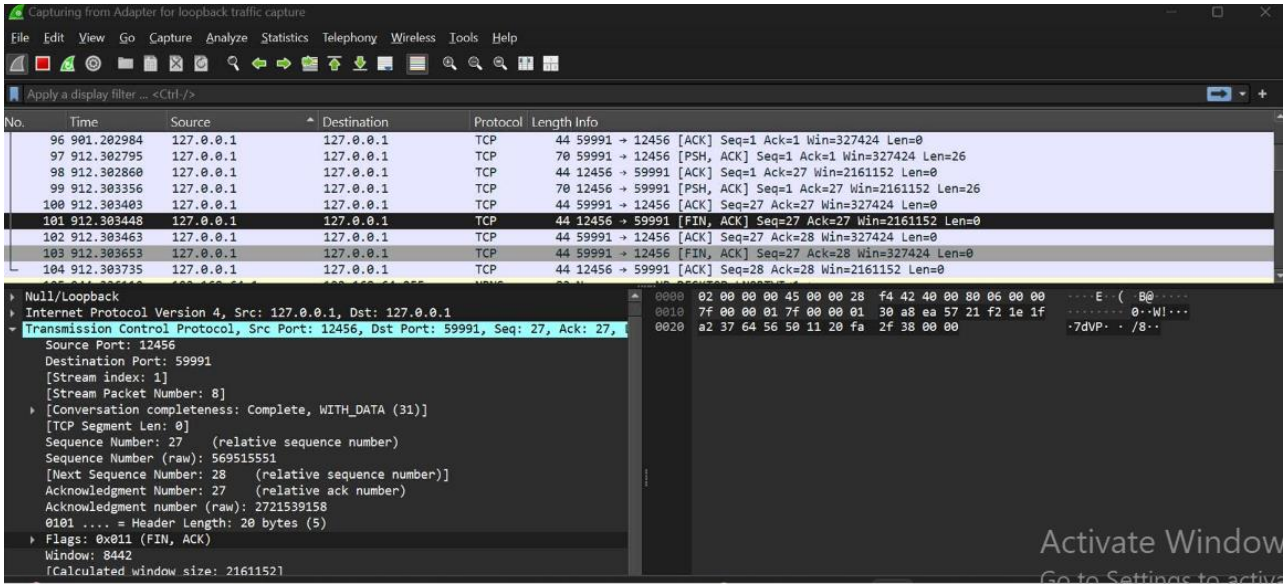


Sender: Client

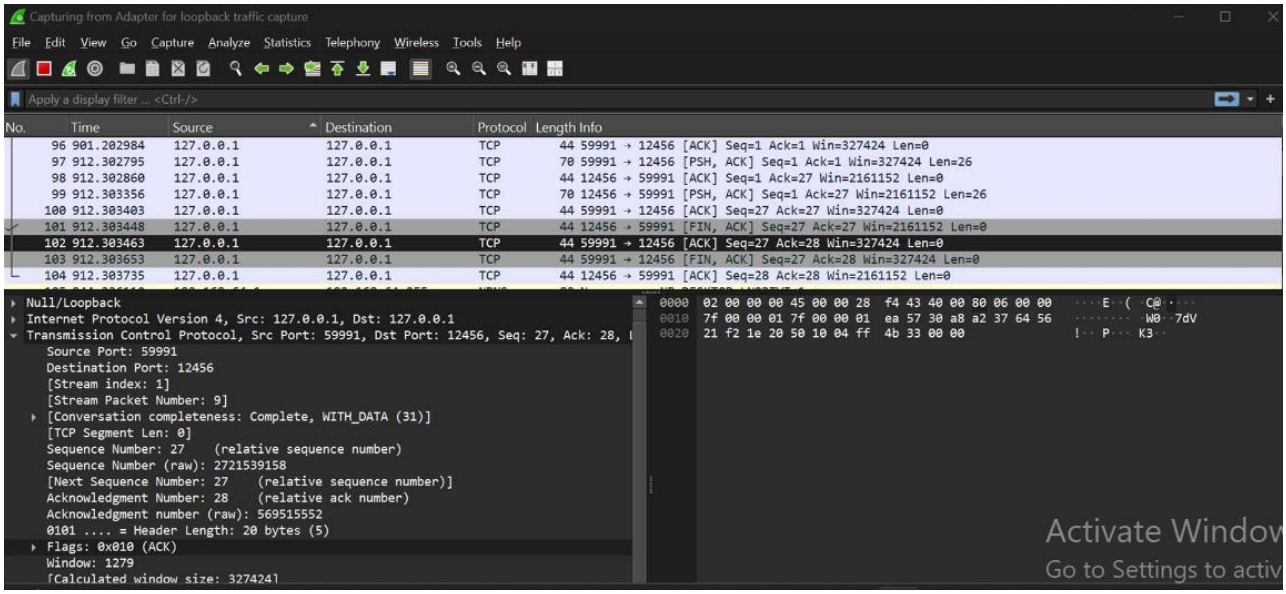
Receiver: Server

6. Guess the frames used for closing the TCP connection after the data transfer. Provide the screenshots of frames (FIN ACK, ACK, FIN ACK, ACK) which is involved in closing the connection.

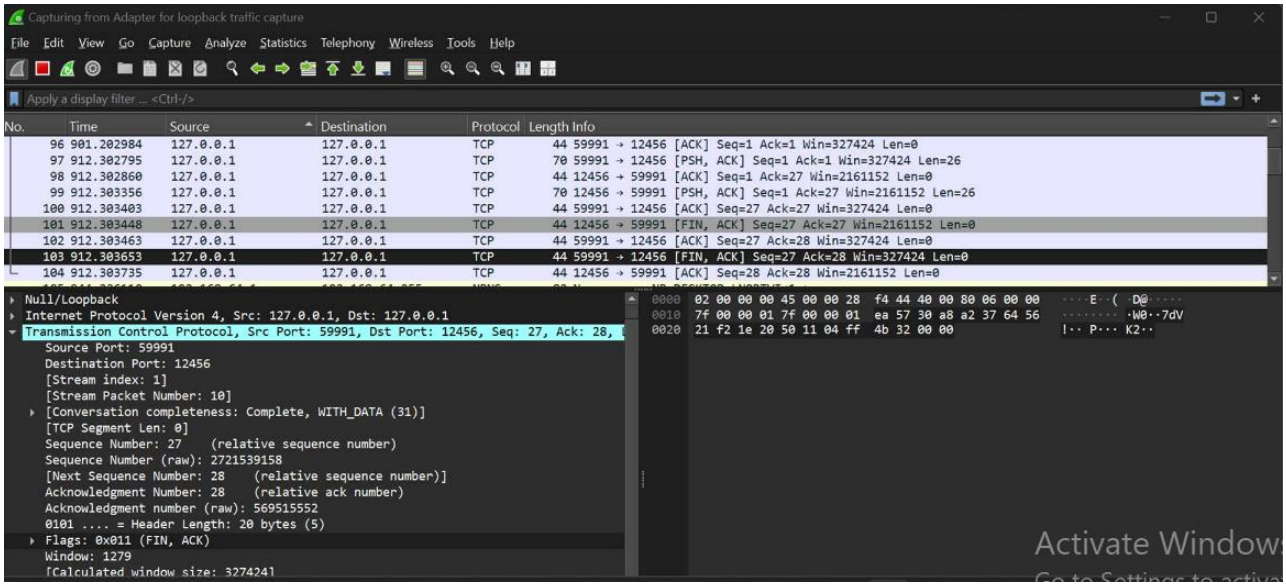
   Fin –ack:


Capturing from Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
96 901.202984 127.0.0.1 127.0.0.1 TCP 44 59991 → 12456 [ACK] Seq=1 Ack=1 Win=327424 Len=0
97 912.302795 127.0.0.1 127.0.0.1 TCP 70 59991 → 12456 [PSH, ACK] Seq=1 Ack=1 Win=327424 Len=26
98 912.302860 127.0.0.1 127.0.0.1 TCP 44 12456 → 59991 [ACK] Seq=1 Ack=27 Win=2161152 Len=0
99 912.303356 127.0.0.1 127.0.0.1 TCP 70 12456 → 59991 [PSH, ACK] Seq=1 Ack=27 Win=2161152 Len=26
100 912.303403 127.0.0.1 127.0.0.1 TCP 44 59991 → 12456 [ACK] Seq=27 Ack=27 Win=327424 Len=0
101 912.303448 127.0.0.1 127.0.0.1 TCP 44 12456 → 59991 [FIN, ACK] Seq=27 Ack=27 Win=2161152 Len=0
102 912.303463 127.0.0.1 127.0.0.1 TCP 44 59991 → 12456 [ACK] Seq=27 Ack=28 Win=327424 Len=0
103 912.303653 127.0.0.1 127.0.0.1 TCP 44 59991 → 12456 [FIN, ACK] Seq=27 Ack=28 Win=327424 Len=0
104 912.303735 127.0.0.1 127.0.0.1 TCP 44 12456 → 59991 [ACK] Seq=28 Ack=28 Win=2161152 Len=0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 12456, Dst Port: 59991, Seq: 27, Ack: 27,
Source Port: 12456
Destination Port: 59991
[Stream index: 1]
[Stream Packet Number: 8]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 27 (relative sequence number)
Sequence Number (raw): 569515551
[Next Sequence Number: 28 (relative sequence number)]
Acknowledgment Number: 27 (relative ack number)
Acknowledgment number (raw): 2721539158
0101 .... = Header Length: 20 bytes (5)
Flags: 0x011 (FIN, ACK)
Window: 8442
[Calculated window size: 2161152]
0000 02 00 00 00 45 00 00 28 f4 42 40 00 80 06 00 00 ····E·(B@·····
0010 7f 00 00 01 7f 00 00 01 30 a8 ea 57 21 f2 1e 1f ········ 0··W!···
0020 a2 37 64 56 50 11 20 fa 2f 38 00 00 ·7dVP· · /8··
Activate Window
Go to Settings to activ

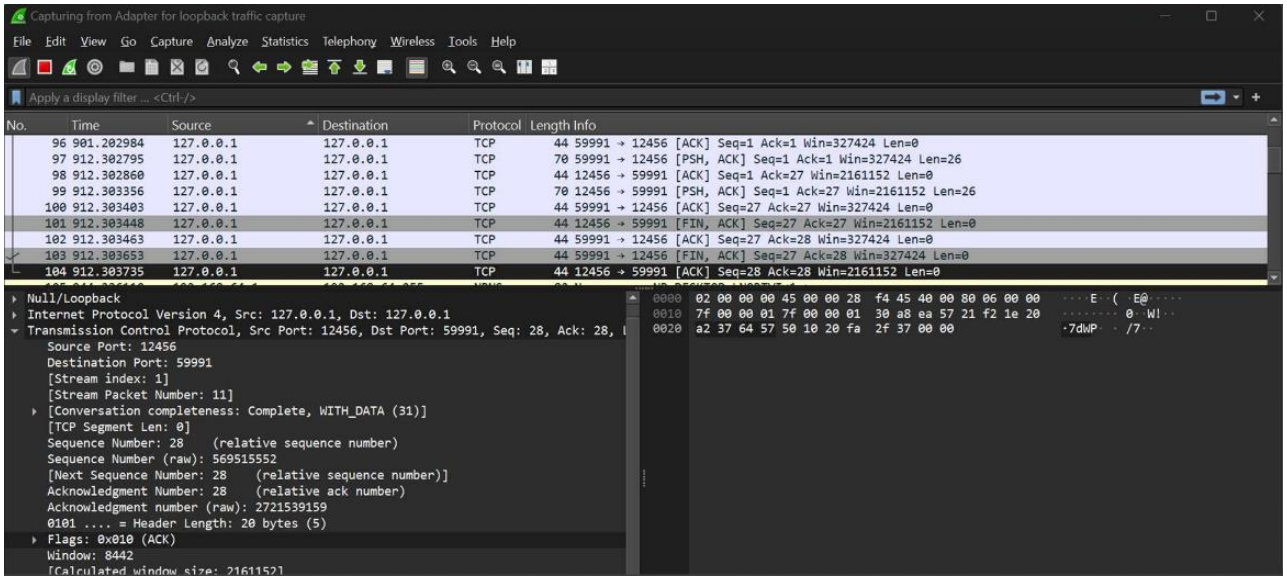
Ack:


Capturing from Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
96 901.202984 127.0.0.1 127.0.0.1 TCP 44 59991 → 12456 [ACK] Seq=1 Ack=1 Win=327424 Len=0
97 912.302795 127.0.0.1 127.0.0.1 TCP 70 59991 → 12456 [PSH, ACK] Seq=1 Ack=1 Win=327424 Len=26
98 912.302860 127.0.0.1 127.0.0.1 TCP 44 12456 → 59991 [ACK] Seq=1 Ack=27 Win=2161152 Len=0
99 912.303356 127.0.0.1 127.0.0.1 TCP 70 12456 → 59991 [PSH, ACK] Seq=1 Ack=27 Win=2161152 Len=26
100 912.303403 127.0.0.1 127.0.0.1 TCP 44 59991 → 12456 [ACK] Seq=27 Ack=27 Win=327424 Len=0
101 912.303448 127.0.0.1 127.0.0.1 TCP 44 12456 → 59991 [FIN, ACK] Seq=27 Ack=27 Win=2161152 Len=0
102 912.303463 127.0.0.1 127.0.0.1 TCP 44 59991 → 12456 [ACK] Seq=27 Ack=28 Win=327424 Len=0
103 912.303653 127.0.0.1 127.0.0.1 TCP 44 59991 → 12456 [FIN, ACK] Seq=27 Ack=28 Win=327424 Len=0
104 912.303735 127.0.0.1 127.0.0.1 TCP 44 12456 → 59991 [ACK] Seq=28 Ack=28 Win=2161152 Len=0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 59991, Dst Port: 12456, Seq: 27, Ack: 28,
Source Port: 59991
Destination Port: 12456
[Stream index: 1]
[Stream Packet Number: 9]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 27 (relative sequence number)
Sequence Number (raw): 2721539158
[Next Sequence Number: 27 (relative sequence number)]
Acknowledgment Number: 28 (relative ack number)
Acknowledgment number (raw): 569515552
0101 .... = Header Length: 20 bytes (5)
Flags: 0x010 (ACK)
Window: 1279
[Calculated window size: 327424]
0000 02 00 00 00 45 00 00 28 f4 43 40 00 80 06 00 00 ····E·(C@·····
0010 7f 00 00 01 7f 00 00 01 ea 57 30 a8 a2 37 64 56 ········ ·W0··7dV
0020 21 f2 1e 20 50 10 04 ff 4b 33 00 00 !·· P··· K3··
Activate Window
Go to Settings to activ

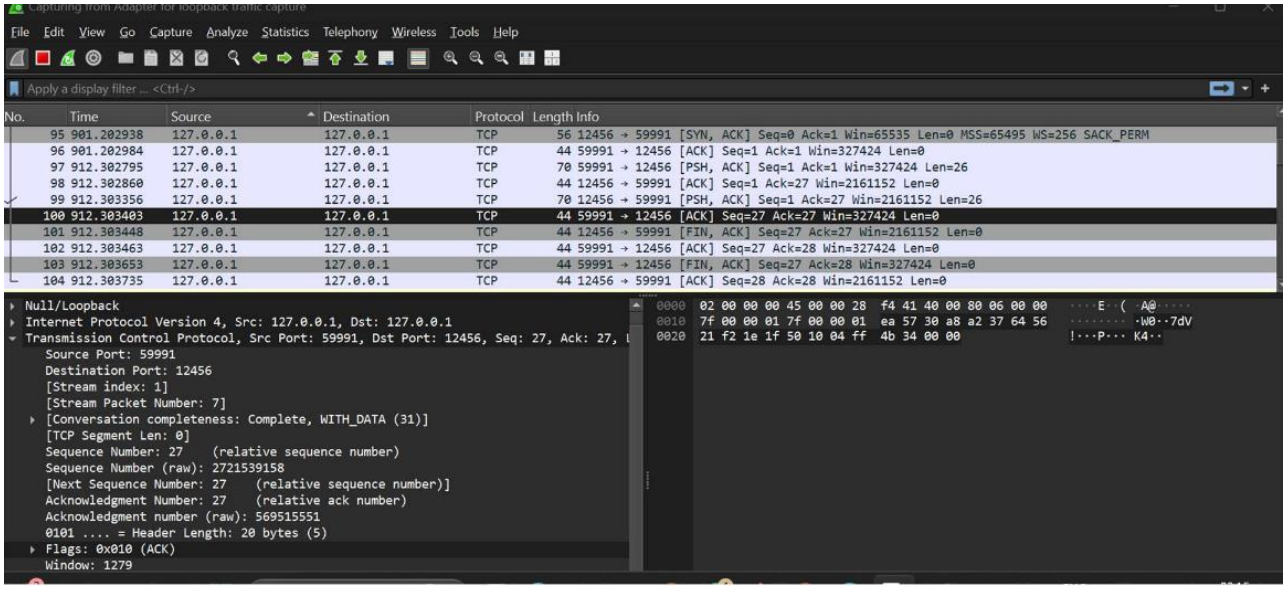
Fin – ack:



Ack:



7. Try to capture the relevant wireshark frames and screenshot with proper identity rules specified earlier.



8. Guess the frames used for TCP connection establishments before the data transfer. Provide the screenshots of 3 handshake frames(SYN, SYN ACK, ACK) in the list pane and identify the sender & receiver process for each of them.

9. Guess the frames used for closing the TCP connection after the data transfer. Provide the screenshots of frames (FIN ACK, ACK, FIN ACK, ACK) which is involved in closing the connection.

   8 and 9 done in 5 and 6 .

Part2: Webserver – application of TCP socket programming[Optional]

You will develop a web server that handles one HTTP request at a time. Your web server should accept and parse the HTTP request, get the requested file from the server's file system, create an HTTP response message consisting of the requested file preceded by header lines, and then send the response directly to the client. If the requested file is not present in the server, the server should send an HTTP "404 Not Found" message back to the client.

Code

Below you will find the skeleton code for the Web server. You are to complete the skeleton code. The places where you need to fill in code are marked with **#Fill in start** and **#Fill in end**. Each place may require one or more lines of code.

Running the Server

Put an HTML file (e.g., HelloWorld.html) in the same directory that the server is in. Run the server program. Determine the IP address of the host that is running the

server (e.g., 128.238.251.26). From another host, open a browser and provide the corresponding URL. For example:

http://128.238.251.26:6789/HelloWorld.html

'HelloWorld.html' is the name of the file you placed in the server directory. Note also the use of the port number after the colon. You need to replace this port number with whatever port you have used in the server code. In the above example, we have used the port number 6789. The browser should then display the contents of HelloWorld.html. If you omit ":6789", the browser will assume port 80 and you will get the web page from the server only if your server is listening at port 80.

Then try to get a file that is not present at the server. You should get a "404 Not Found" message.

What to Hand in

You will hand in the complete server code along with the screen shots of your client browser, verifying that you actually receive the contents of the HTML file from the server.

Skeleton Python Code for the Web Server

```
#import socket module
from socket import *
import sys # In order to terminate the program
serverSocket = #Fill in start                #Fill in end
#Prepare a sever socket
#Fill in start
#Fill in end
while True:
#Establish the connection print('Ready to serve...')
connectionSocket, addr =  #Fill in start  #Fill in end
try:
    message =    #Fill in start              #Fill in end
    filename = message.split()[1]
    f = open(filename[1:])
    outputdata = #Fill in start          #Fill in end
    # Send the HTTP response header line to the
connection socket
    #Fill in start            #Fill in end
    #Send the content of the requested file to the
client
    for i in range(0, len(outputdata)):
        connectionSocket.send(outputdata[i].encode())
     connectionSocket.send("\r\n".encode())
```

```
        connectionSocket.close()
        except IOError:
        #Send response message for file not found
        #Fill in start
        #Fill in end

        #Close client socket
        #Fill in start
        #Fill in end
serverSocket.close()
sys.exit()#Terminate the program after sending the
corresponding data
```

Optional Exercises

1. Currently, the web server handles only one HTTP request at a time. Implement a multithreaded server that is capable of serving multiple requests simultaneously. Using threading, first create a main thread in which your modified server listens for clients at a fixed port. When it receives a TCP connection request from a client, it will set up the TCP connection through another port and services the client request in a separate thread. There will be a separate TCP connection in a separate thread for each request/response pair.
2. Instead of using a browser, write your own HTTP client to test your server. Your client will connect to the server using a TCP connection, send an HTTP request to the server, and display the server response as an output. You can assume that the HTTP request sent is a GET method.

The client should take command line arguments specifying the server IP address or host name, the port at which the server is listening, and the path at which the

requested object is stored at the server. The following is an input command format to run the client.

client.py server_host server_port filename