Computer Networks Lab
Week 5

Vishal R
PES1UG19CS571
Section I

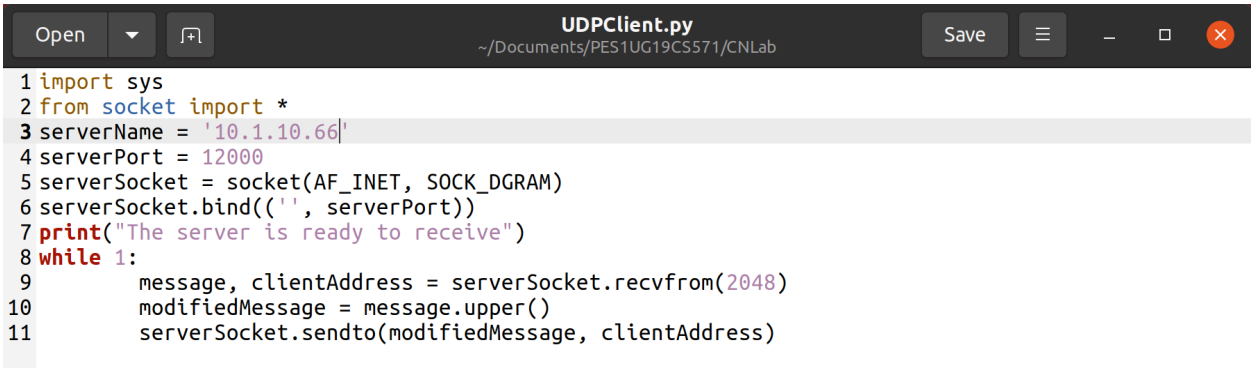
1. Socket Programming

1. Create an application that will convert lowercase letters to uppercase.
2. Create Socket API for both client and server.
3. Server IP address and port must be taken from the command line.

1.1 UDP connection

- A UDP commotion can be can be made between two machines with the help of socket interface. The `socket` library in Python can be used to achieve this.
- To create a UDP socket interface, the type of socket needs to be set as `SOCK_DGRAM.`
- The type of address needs to be set as `AF_INET` which corresponds to IPv4.
- Once the server socket application is created, it needs to be hosted. Hence we need to bind it to a host IP address and port number using the bind() function.
- Similarly, the client socket application needs to be connected to a host using the IP address and port number.
- Now, the socket can listen for incoming connections as well as send messages to connected host machines.

1.1.1 UDP Server Code




```
1 import sys
2 from socket import *
3 serverName = '10.1.10.66'
4 serverPort = 12000
5 serverSocket = socket(AF_INET, SOCK_DGRAM)
6 serverSocket.bind(('', serverPort))
7 print("The server is ready to receive")
8 while 1:
9         message, clientAddress = serverSocket.recvfrom(2048)
10        modifiedMessage = message.upper()
11        serverSocket.sendto(modifiedMessage, clientAddress)
```

1.1.2 UDP Client Code

```
import sys
from socket import *

serverName = sys.argv[1].encode()
serverPort = int(sys.argv[2].encode())
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = input('Input lowercase sentence:').encode()
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode())
clientSocket.close()
```

1.1.3 Starting the UDP server on the server machine

On the server machine, we will execute the `UDPServer.py` Code using the following command `$ python3 UDPServer.py` . This will start our UDP server on the server machine.

```
student@CSELAB:~/Documents/PES1UG19CS571/CN/Week5$ gedit UDPServer.py
^C
student@CSELAB:~/Documents/PES1UG19CS571/CN/Week5$ python3 UDPServer.py
The server is ready to receive
```

1.1.4 Connecting to the server machine from the client machine

On the client machine, we will execute the `UDPClient.py` code to make a connection to the server which is running on the other machine. We will execute the code by using the command `$ python3 UDPClient.py 10.1.10.66 12000`

```
student@CSELAB:~/Desktop/cs589/cn/week 5$ python3 UDPClient.py 10.1.10.66 12000
Input lowercase sentence:vishal r cs571
VISHAL R CS571
```

In the above figure, you can see that the server has responded back to the client with the input sentence converted to uppercase.

1.1.5 Wireshark packet capture

Wireshark was running in the background monitoring ‘any' interface to capture packets received and sent.

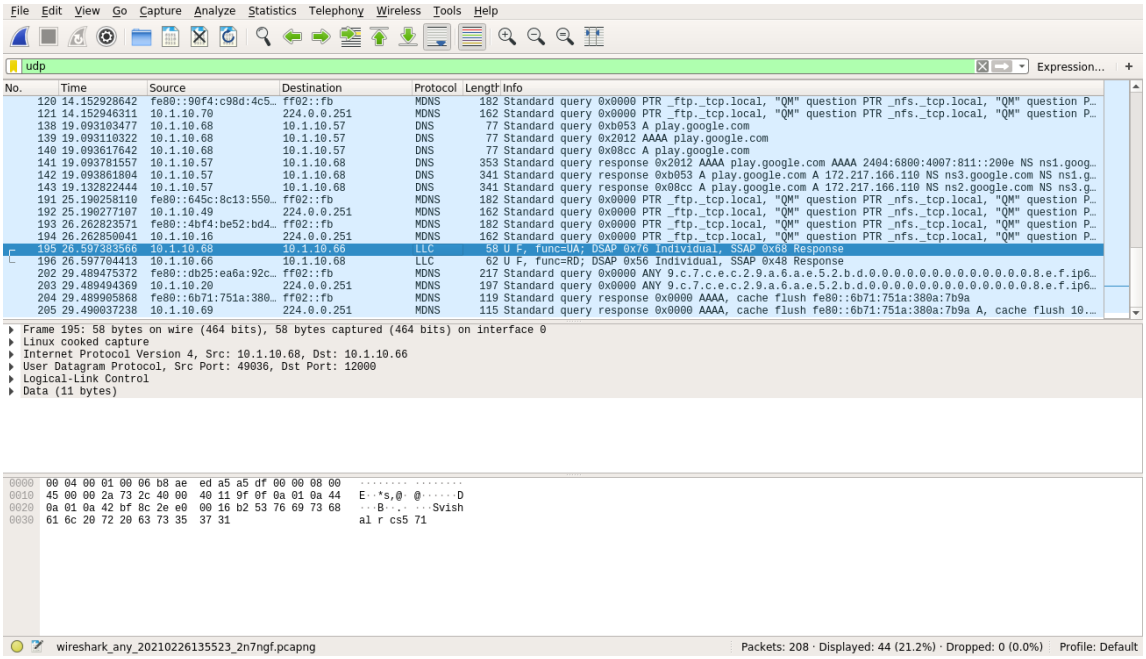


Figure : UDP packet sent from client to server machine

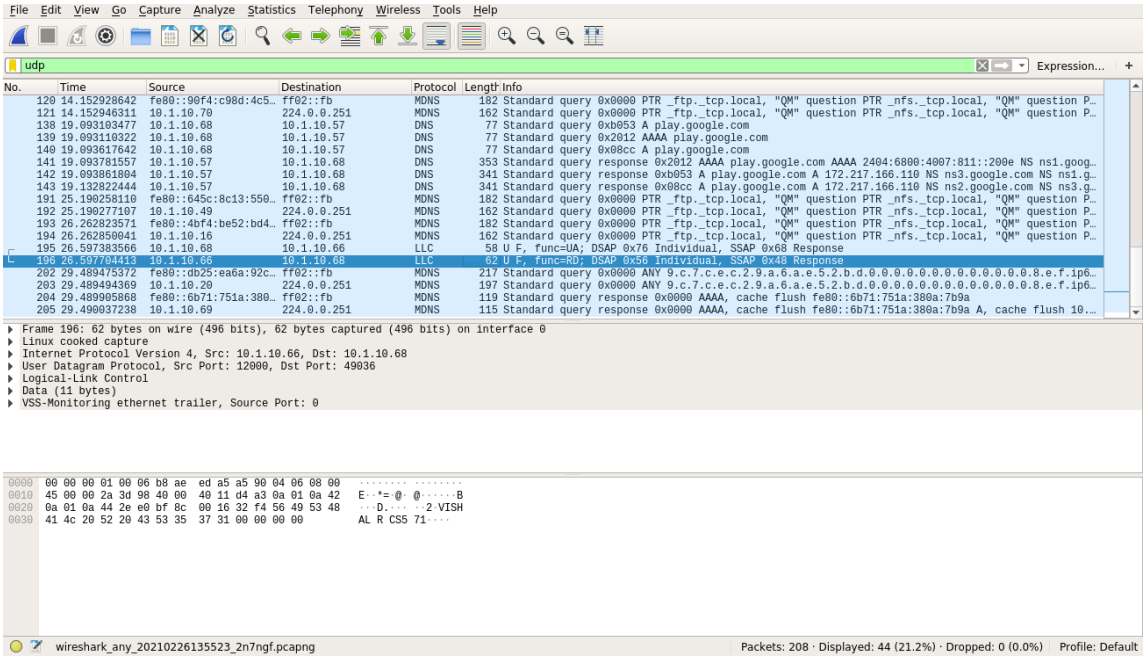


Figure : UDP packet received by the client from the server

1.2 TCP Connection

- A TCP connection can be made between the two machines with the help of a socket interface using the socket library.
- To create a TCP socket interface, the type of socket needs to be set as `SOCK_STREAM`.
- The type of addresses needs to be set as `AF_INET` which indicates that it is IPv4.
- Once the server socket application is created, it needs to be hosted on the server machine so that we can make a connection to the server from the client. To do this, we need to bind it to a host IP and a port number using the bind() function of socket library.
- The client socket application needs to connect to a host using the IP address and port number.
- Now, the socket can listen for incoming connections as well as send messages to connected host machines.

1.2.1 TCP Server Code

```
Open
from socket import *
serverName = '10.1.10.66'
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind((serverName,serverPort))
serverSocket.listen(1)
print("The server is ready to receive")
while 1:
        connectionSocket, addr = serverSocket.accept()
        sentence = connectionSocket.recv(1024)
        capitalizedSentence = sentence.upper()
        connectionSocket.send(capitalizedSentence)
        connectionSocket.close()
```

1.2.2 TCP Client Code

```
from socket import *
serverName = "10.1.10.66"
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = input('Input lowercase sentence:').encode()
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

1.2.3 Starting the TCP server on the server machine

The TCP server can be started on the server machine by executing the `TCPServer.py` code by using `$ python3 TCPServer.py` command

```
student@CSELAB:~/Documents/PES1UG19CS571/CN/Week5$ python3 TCPServer.py
The server is ready to receive
```

1.2.4 Establishing a connection from the client to the server

To establish a connection to the server, we need to execute the `TCPClient.py` code by executing `$ python3 TCPClient.py` command

```
student@CSELAB:~/Desktop/cs589/cn/week 5$ python3 TCPClient.py
Input lowercase sentence:vishal r cs 571
From Server: VISHAL R CS 571
```

1.2.4 Wireshark Packet Capture for TCP Connection

Wireshark was kept open, monitoring ‘any' interface for capturing packets send during the above experiment.

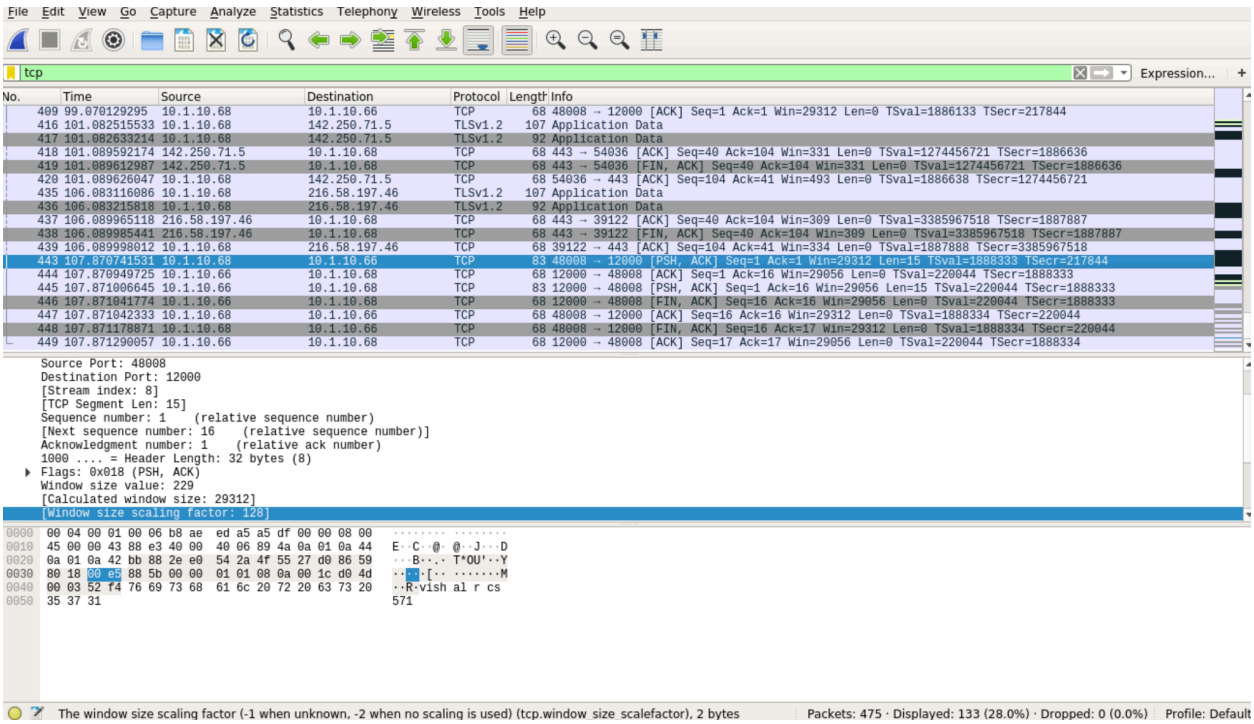


Figure : TCP packet sent by client to the server.

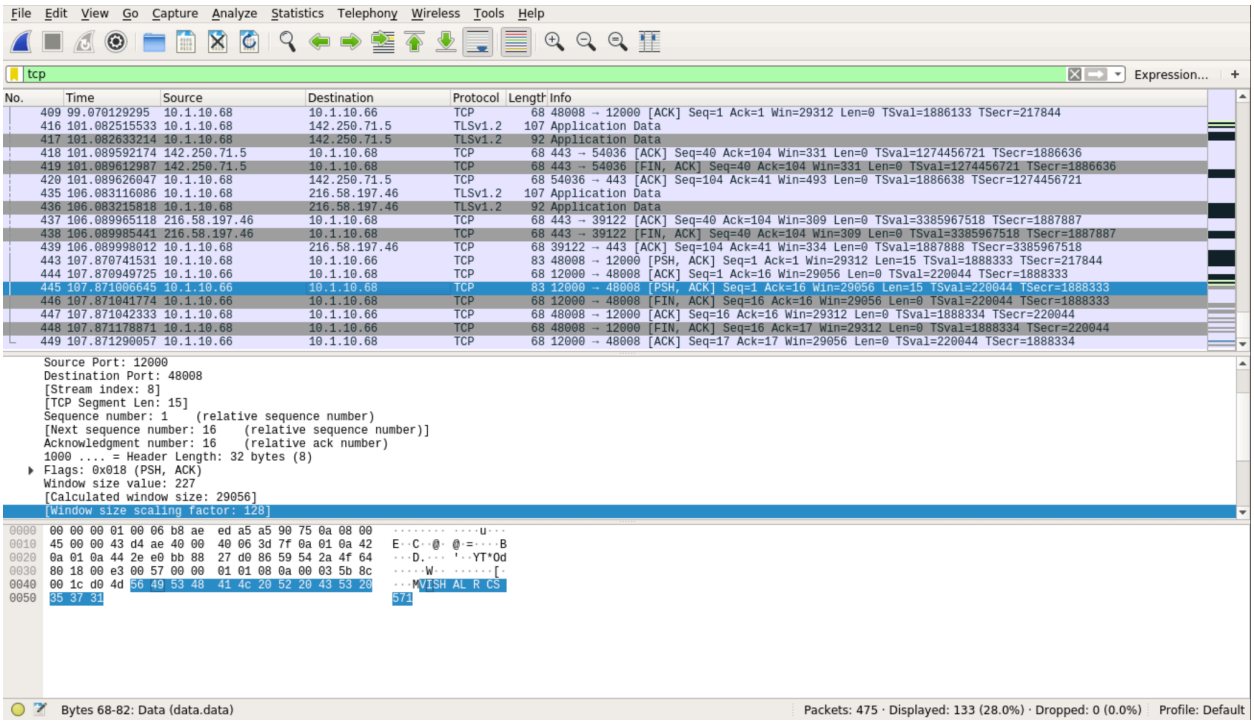


Figure : TCP packet sent by server to the client machine.

1.3 Questions

1. Suppose you run TCPClient before you run TCPServer, what happens? Why?

When the TCP Client is executed before the TCPServer, we will get a ConnectionRefusedError. This is because the server socket application we are trying to connect has not been initiated and hence the server is not listening for any connections on the given port number. A TCP connection can be established between two socket interfaces only when a host machine listens to request on a given IP and port number and accepts connections made by another machine at the same address and port.

```
student@CSELAB:~/Desktop/cs589/cn/week 5$ python3 TCPClient.py
Traceback (most recent call last):
  File "TCPClient.py", line 5, in <module>
    clientSocket.connect((serverName,serverPort))
ConnectionRefusedError: [Errno 111] Connection refused
student@CSELAB:~/Desktop/cs589/cn/week 5$
```

2. Suppose you run UDPClient before you run UDPServer, what happens? Why?

When we run the client without starting the UDP server, we do not get an error. This is because, UDP does not require a prior connection to be set up between the host machines for transfer of data to begin.

```
student@CSELAB:~/Desktop/cs589/cn/week 5$ python3 UDPClient.py 10.1.10.66 12000
Input lowercase sentence:server not running
```

3. What happens if your use different port numbers for the client and the server sides?

This will lead to a ConnectionRefusedError for a TCP Connection. Whereas for UDP connection, since no prior connection is required to be established between host and server for data transfer, we do not get any errors. Messages sent by the client will be lost and the server will not receive the messages sent by client.

```
student@CSELAB:~/Desktop/cs589/cn/week 5$ python3 UDPClient.py 10.1.10.66 12002
Input lowercase sentence:port changed
^CTraceback (most recent call last):
  File "UDPClient.py", line 9, in <module>
    modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
KeyboardInterrupt
```

```
student@CSELAB:~/Desktop/cs589/cn/week 5$ python3 TCPClient.py
Traceback (most recent call last):
  File "TCPClient.py", line 5, in <module>
    clientSocket.connect((serverName,serverPort))
ConnectionRefusedError: [Errno 111] Connection refused
student@CSELAB:~/Desktop/cs589/cn/week 5$
```

2. Task 2 - Web Server

In this task, we will create a simple web server that can handle only one request at a time.

```
# Import socket module
from socket import *

# Create a TCP server socket
#(AF_INET is used for IPv4 protocols)
#(SOCK_STREAM is used for TCP)

serverSocket = socket(AF_INET, SOCK_STREAM)

# Assign a port number
serverPort = 8080

# Bind the socket to server address and server port
serverSocket.bind(("", serverPort))

# Listen to at most 1 connection at a time
serverSocket.listen(1)

# Server should be up and running and listening to the incoming connections
while True:
        print 'Ready to serve...'

        # Set up a new connection from the client
        connectionSocket, addr = serverSocket.accept()

        # If an exception occurs during the execution of try clause
        # the rest of the clause is skipped
        # If the exception type matches the word after except
        # the except clause is executed
        try:
                # Receives the request message from the client
                message =  connectionSocket.recv(1024)
                # Extract the path of the requested object from the message
                # The path is the second part of HTTP header, identified by [1]
                filename = message.split()[1]
                # Because the extracted path of the HTTP request includes
                # a character '\', we read the path from the second character
                f = open(filename[1:])
                # Store the entire contenet of the requested file in a temporary buffer
                outputdata = f.read()
                # Send the HTTP response header line to the connection socket
                connectionSocket.send("HTTP/1.1 200 OK\r\n\r\n")

                # Send the content of the requested file to the connection socket
                for i in range(0, len(outputdata)):
                        connectionSocket.send(outputdata[i])
                connectionSocket.send("\r\n")

                # Close the client connection socket
                connectionSocket.close()
```

Figure : Creating a Web Server

To start this server, we need to execute `$ python2 WebServer.py` command on server machine.

```
student@CSELAB:~/Documents/PES1UG19CS571/CN/Week5$ python2 WebServer.py
Ready to serve...
```

We will create a file called as file.txt in the server's folder. We will add few sentences into this file and save the file and restart the server.

On the client, we will go to the browser and type `http://10.1.10.66:8080/file.txt` in the search bar.

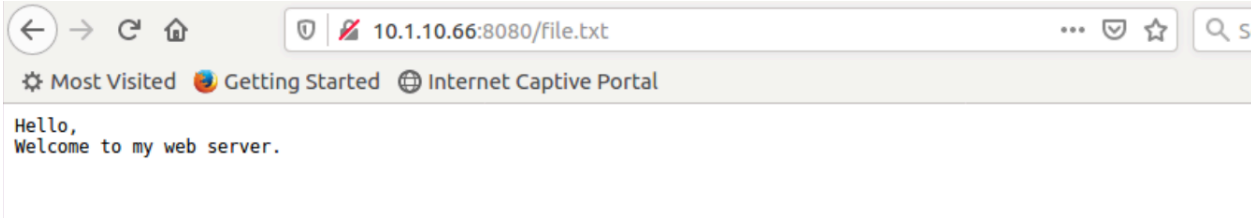


Figure : Opening file.txt from client machine

Wireshark Packet Capture

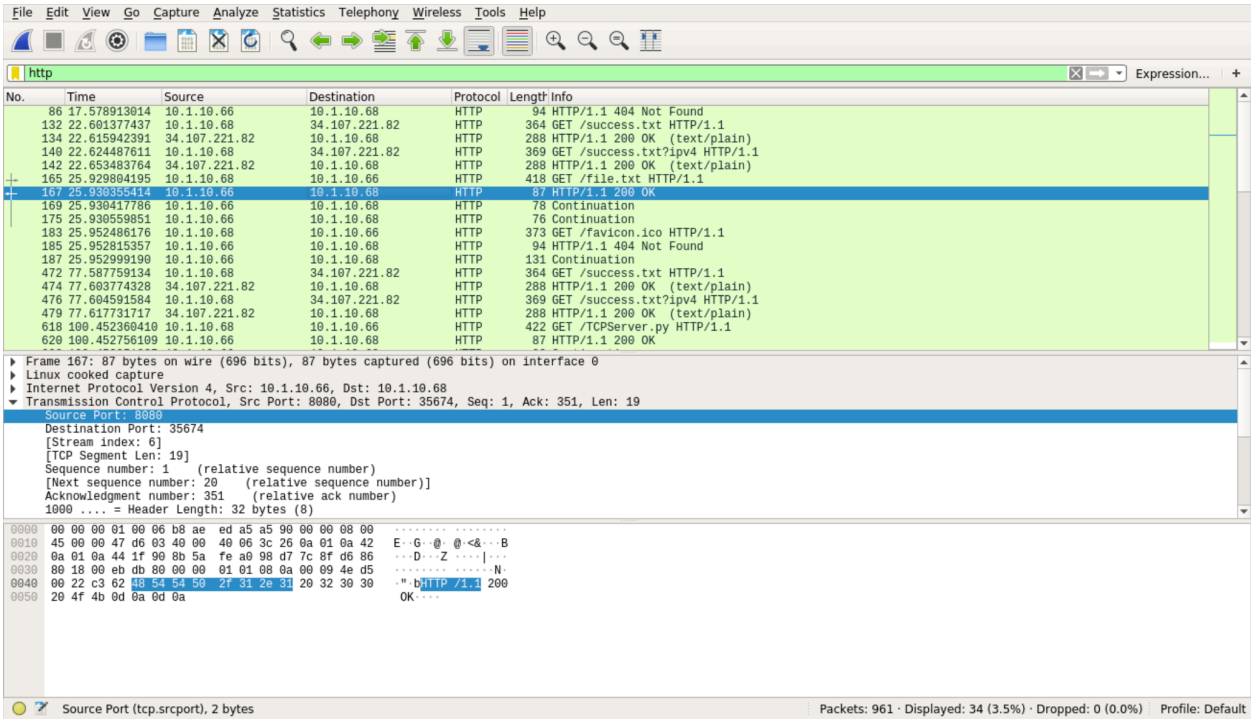
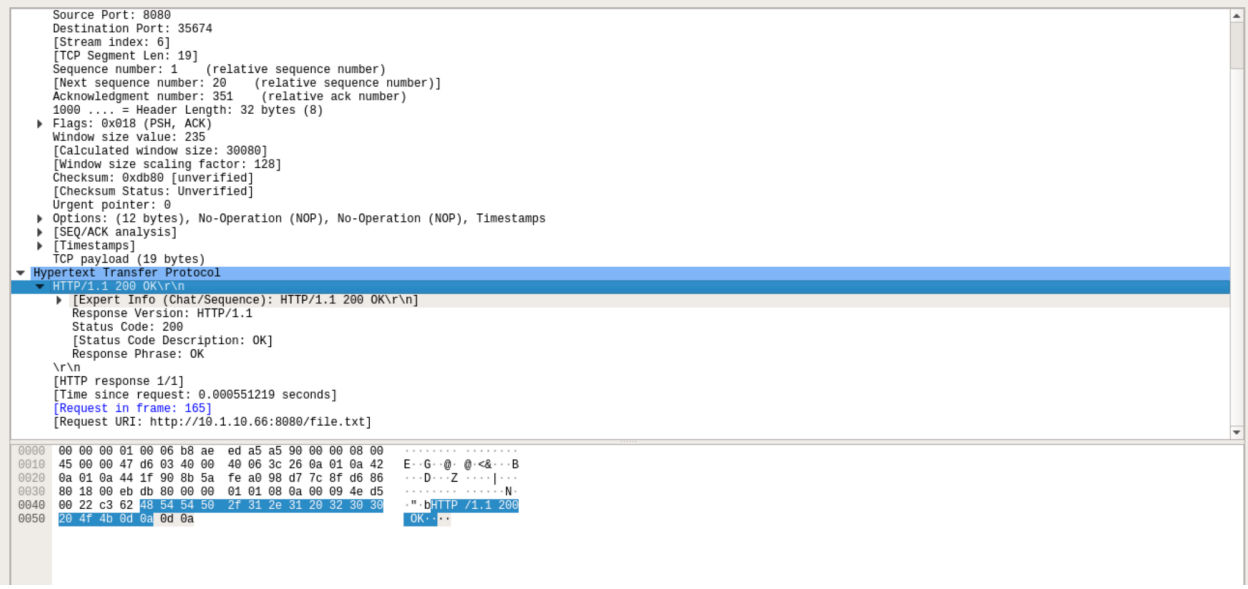


Figure : Capturing packets using Wireshark



Requesting for a file thats not there in the server, the server will respond back with a 404 Not Found.