CN Lab Report Week 5

PES2UG19CS412
Suhan B Revankar

1. Socket Programming

1. Create an application that will
    a. Convert lowercase letters to uppercase
        - e.g. [a…z] to [A…Z]
        - code will not change any special characters, e.g. &*!
    b. If the character is in uppercase, the program must not alter
2. Create Socket API both for client and server.
3. Must take the server address and port from the Command Line Interface (CLI).

1.1 TCP Connection

- A TCP connection can be made between two machines with the help of a socket interface using the socketlibrary on Python3.
- 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_INETwhich corresponds to IPv4.
- Once the server socket application is created, it needs to be hosted and hence needs to bind to a host IP and port number using thebind()function.
- Similarly, the client socket application needs to connect to a host using the IP address and port number.
- The socket can now listen for incoming connections as well as send messages to connected host machines.

1.1.1 TCP Server

tcp_server.py
~/Desktop/week 5

```python
from socket import *
serverPort = 12001
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('10.0.2.15',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()
```

Python Tab Width: 8 Ln 13, Col 9 INS

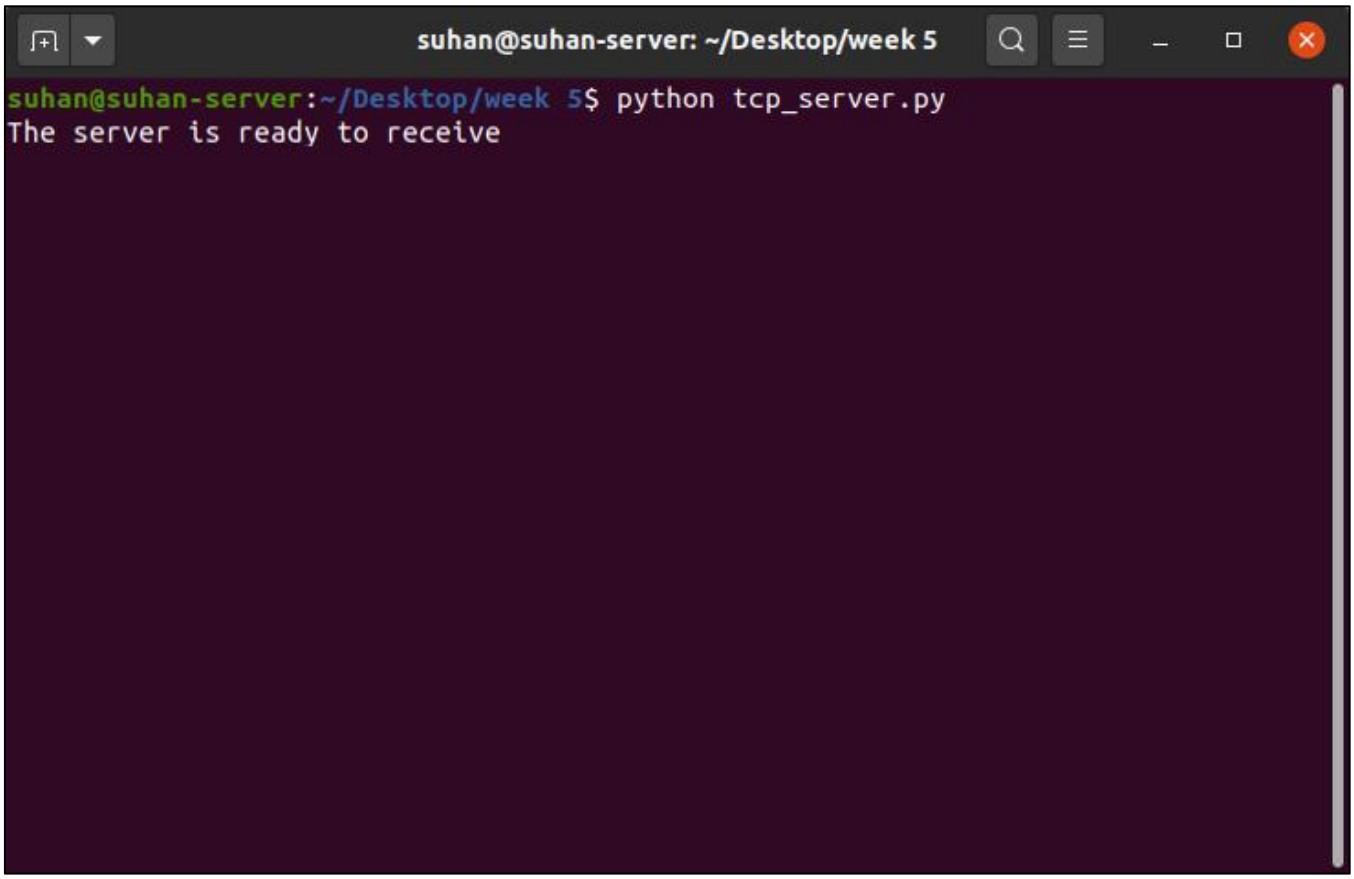
1.1.2 TCP Client

Open | tcp_client.py ~/Desktop/week 5 | Save

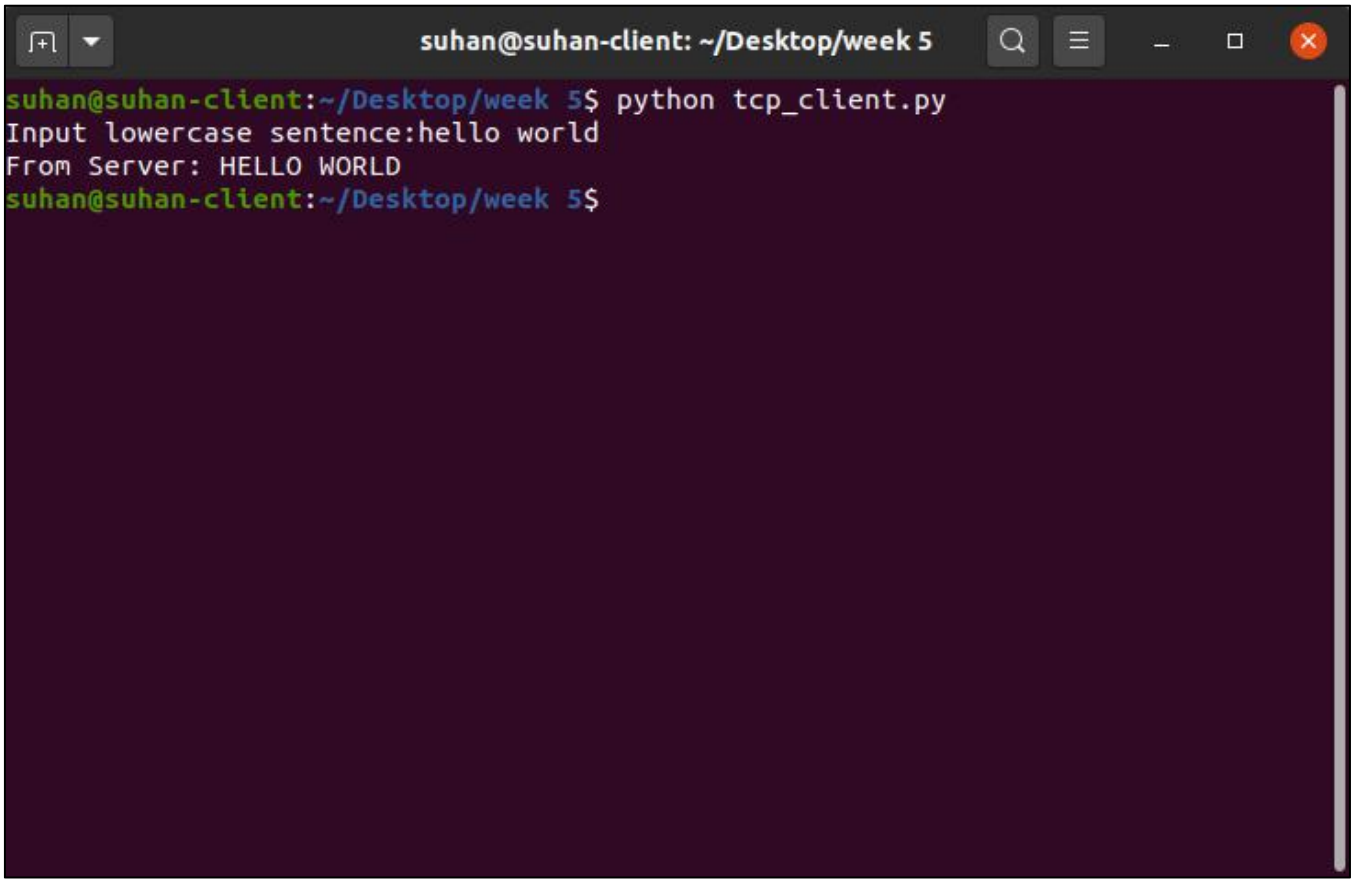
```python
from socket import *
serverName = '10.0.2.15'
serverPort = 12001
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

Python | Tab Width: 8 | Ln 3, Col 13 | INS

1.1.3 TCP Connection between Server and Client

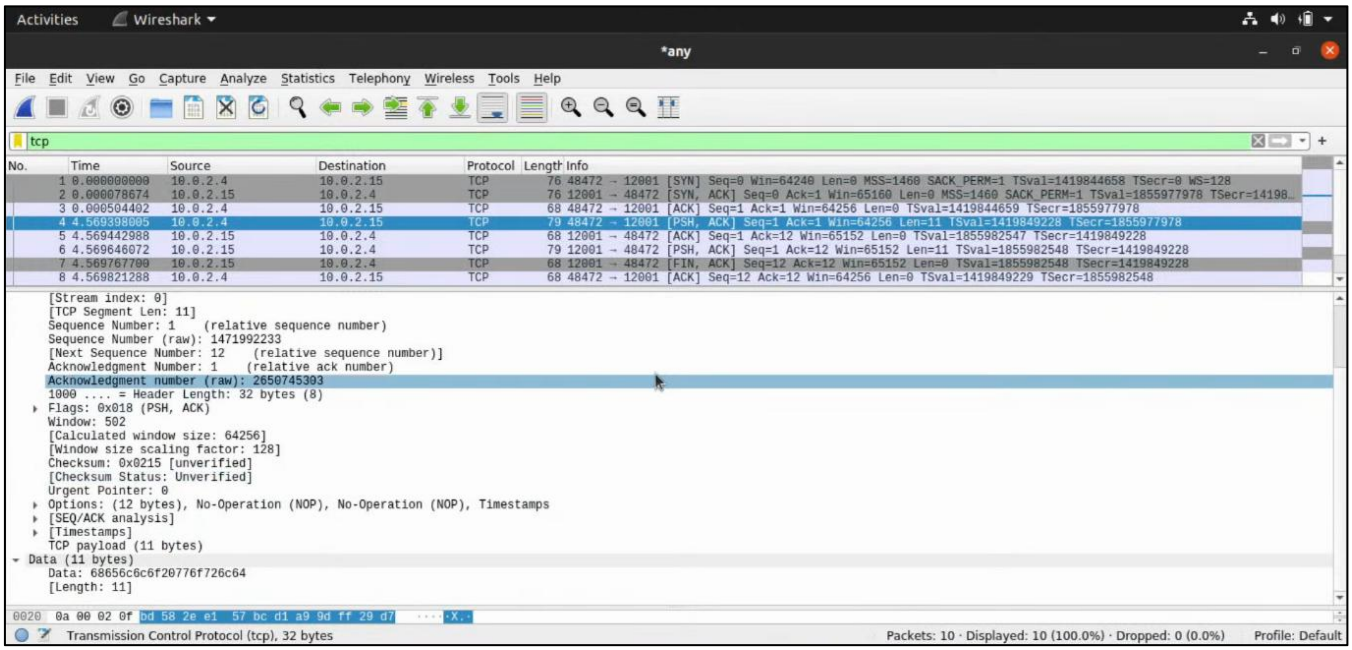


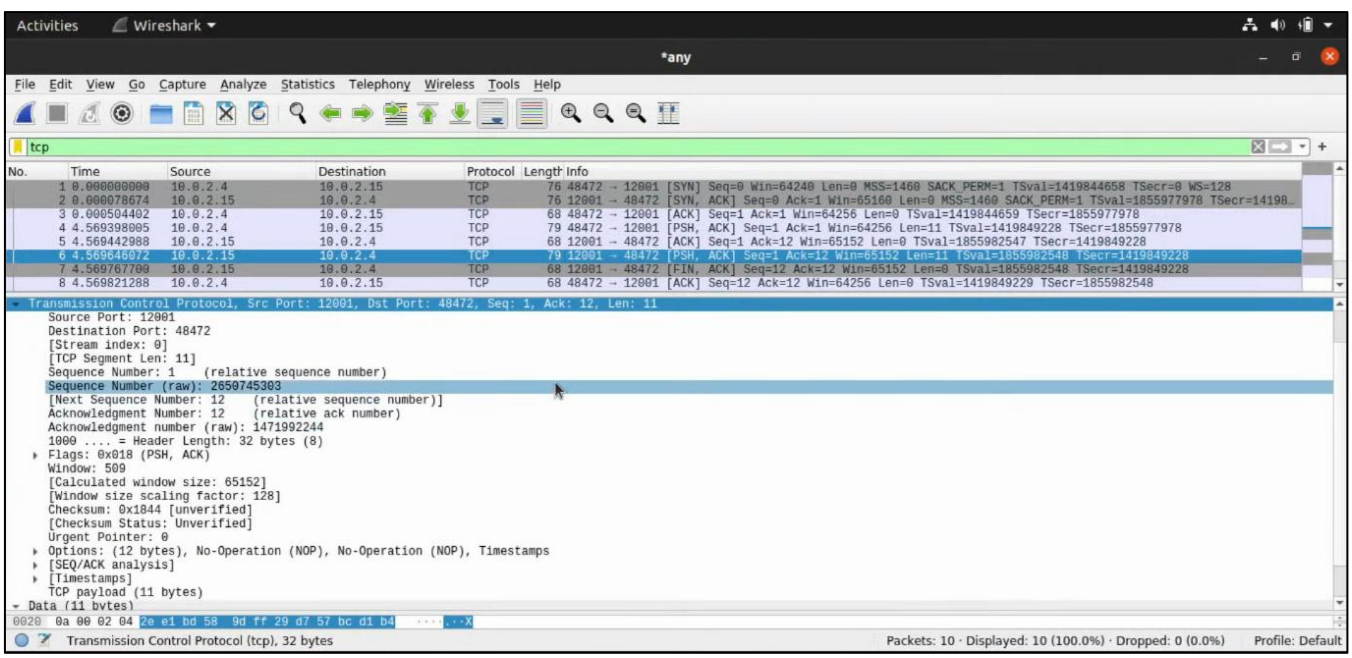
TCP Server


suhan@suhan-client: ~/Desktop/week 5
suhan@suhan-client:~/Desktop/week 5$ python tcp_client.py
Input lowercase sentence:hello world
From Server: HELLO WORLD
suhan@suhan-client:~/Desktop/week 5$

TCP Client

1.1.4 Wireshark Capture for TCP Connection





1.2 UDP Connection

- A UDP connection can be made between two machines with the help of a socket interface using the socketlibrary on Python3.
- To create a UDP socket interface, the type of socket needs to be set as SOCK_DGRAM.
- The type of addresses needs to be set as AF_INETwhich corresponds to IPv4.

- Once the server socket application is created, it needs to be hosted and hence needs to bind to a host IP and port number using thebind()function.
- Similarly, the client socket application needs to connect to a host using the IP address and port number.
- The socket can now listen for incoming connections as well as send messages to connected host machines.

1.2.1 UDP Server

udp_server.py
~/Desktop/week 5

```python
from socket import *
serverPort = 11000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('10.0.2.15', serverPort))
print "The server is ready to receive"
while 1:
        message, clientAddress = serverSocket.recvfrom(2048)
        modifiedMessage = message.upper()
        serverSocket.sendto(modifiedMessage, clientAddress)
```

Python Tab Width: 8 Ln 10, Col 9 INS

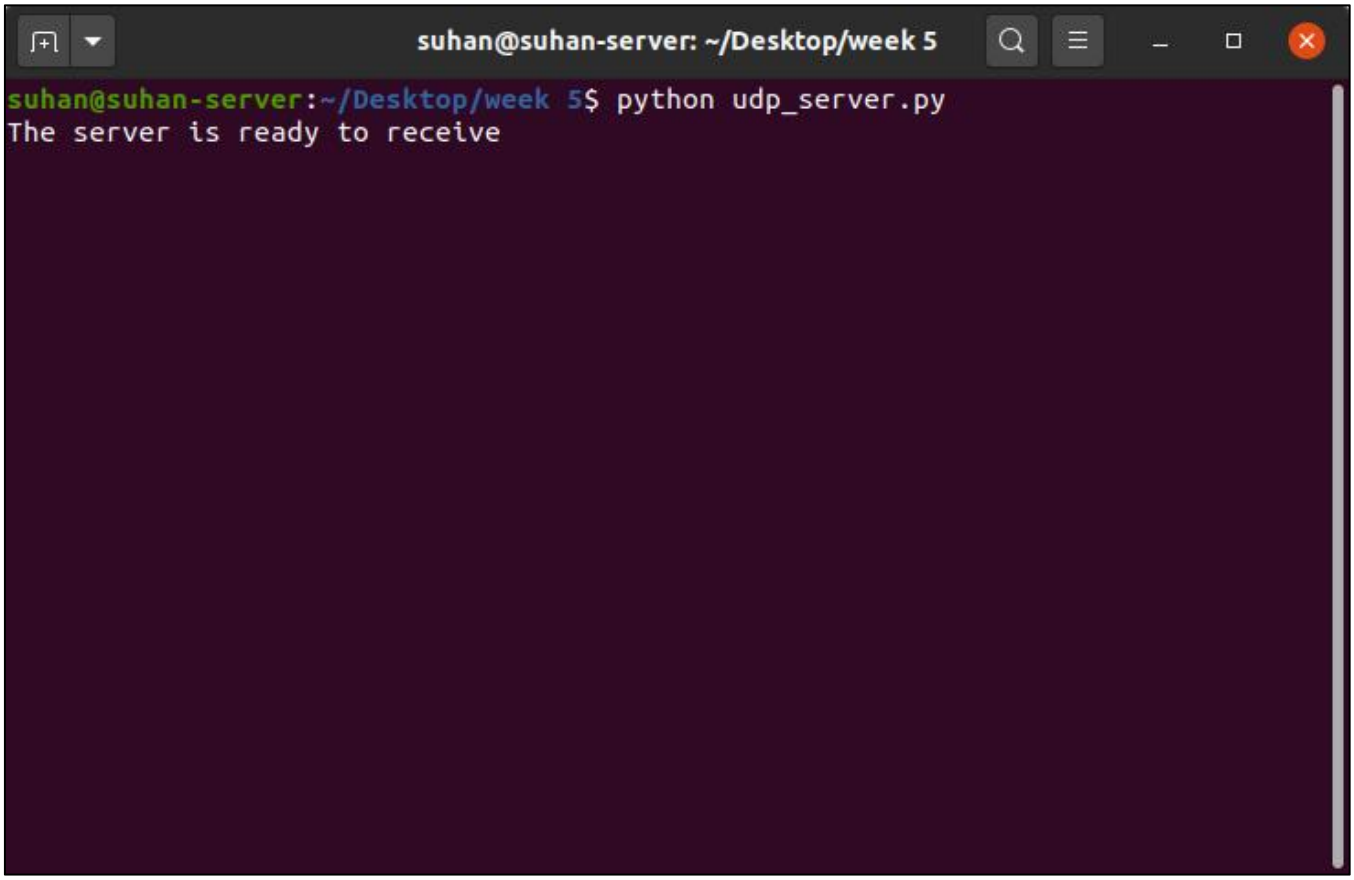
1.2.2 UDP Client

*udp_client.py
~/Desktop/week 5

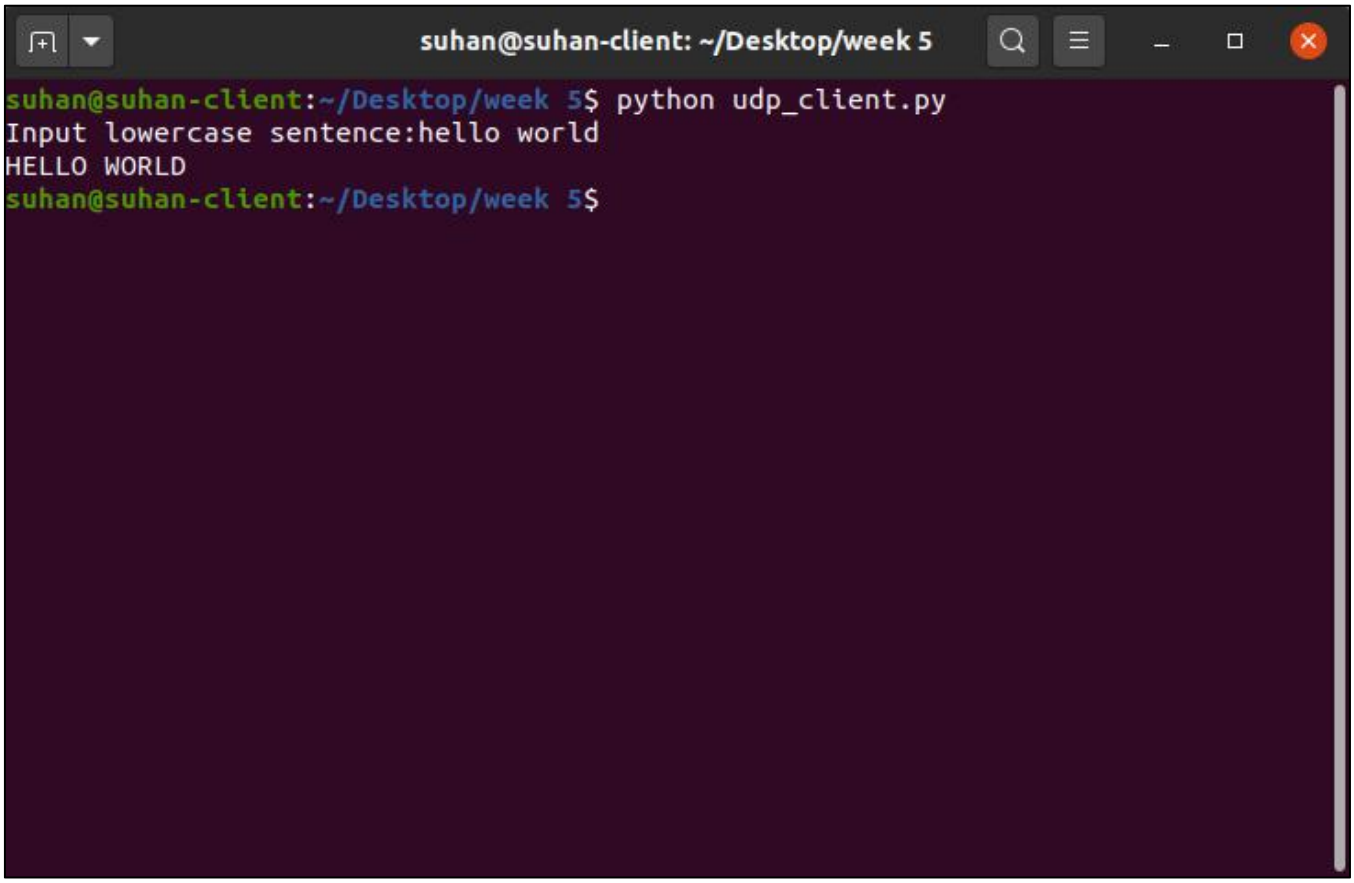
```python
from socket import *
serverName = '10.0.2.15'
serverPort = 11000
clientSocket = socket(AF_INET,SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.close()
```

Python Tab Width: 8 Ln 10, Col 1 INS

1.2.3 UDP Connection between Server and Client

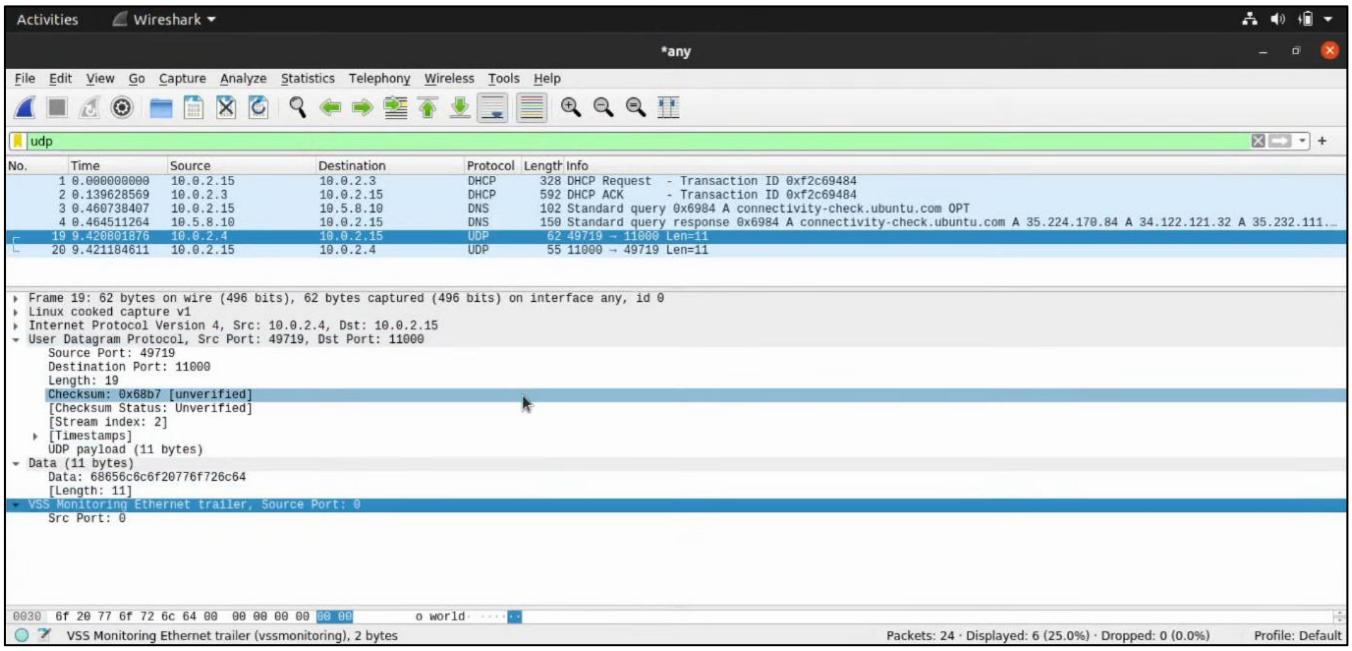


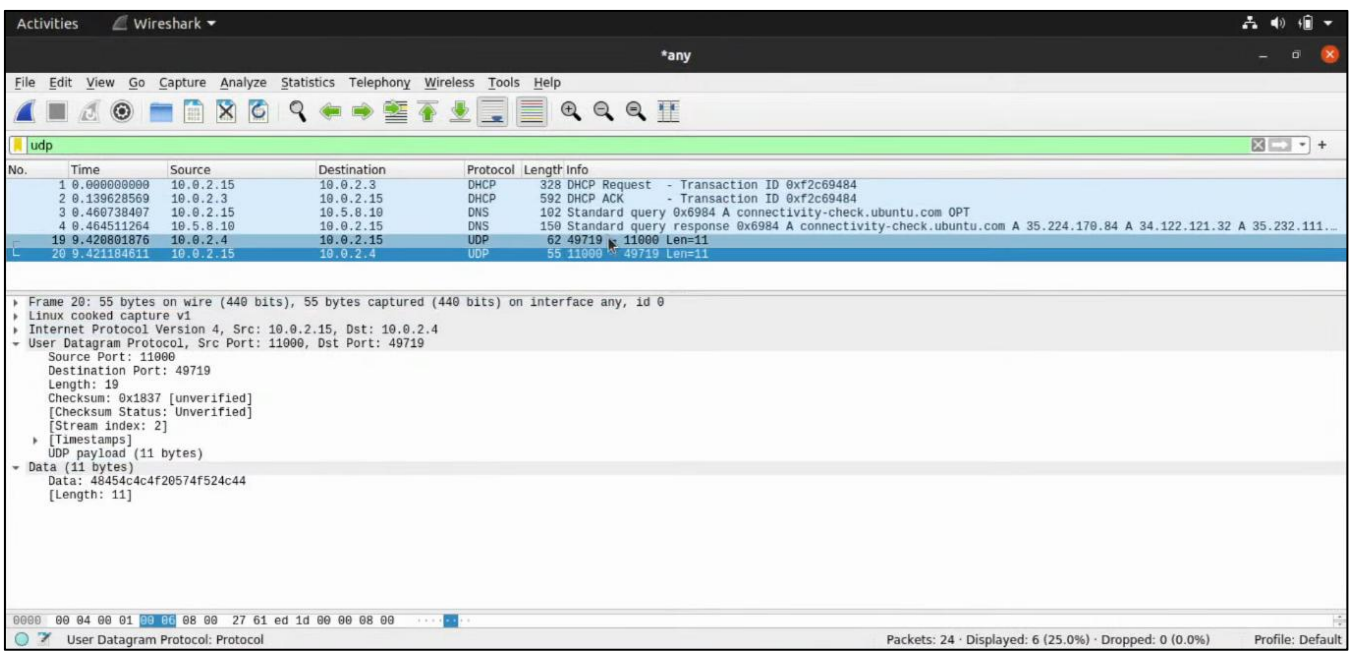
UDP Server


suhan@suhan-client: ~/Desktop/week 5
suhan@suhan-client:~/Desktop/week 5$ python udp_client.py
Input lowercase sentence:hello world
HELLO WORLD
suhan@suhan-client:~/Desktop/week 5$

UDP Client

1.2.4 Wireshark Capture for UDP Connection



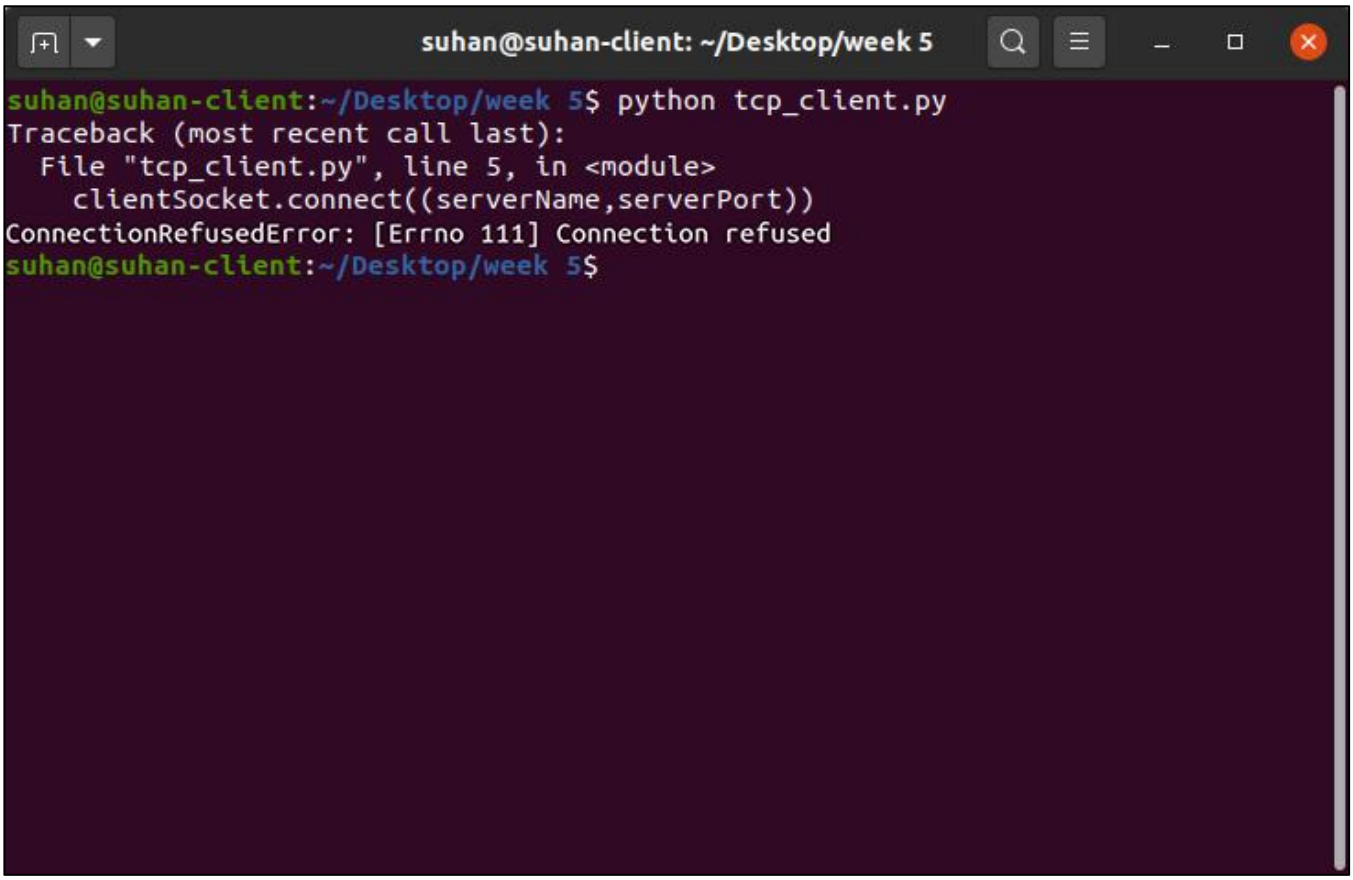


1.3 Problems

Q1. *Suppose you run TCPClient before you run TCPServer. What happens? Why?*

Answer – This will lead to a **ConnectionRefusedError**, since the *server socket application we are trying to connect to has not been initiated and is not listening for connections on the given port number*. Hence, any connection requests sent by a client machine at that IP and port number immediately fail since the connection gets refused.

A TCP connection can be established between two socket interfaces only when a host machine listens to requests on a given IP address and port number and accepts connections made by another machine at the same address and port.



Q2. *Suppose you run UDPClient before you run UDPServer. What happens? Why?*

Answer – No error will be obtained since *UDP does not require a prior connection to be set up between the host machines for data transfer to begin*. It is a connectionless protocol which transfers packets of data to a destination IP and port number without verifying the existence of the connection. Hence, it is prone to data integrity issues such as loss of packets.

If any packets of data are sent before the server is executed, the packets are lost forever and will not reach the server socket application. However, if any packets of data are sent after the server is executed, the client will be able to send packets to a destination server and also receive response packets in return.

Q3. *What happens if you use different port numbers for the client and server sides?*

Answer – This will lead to a **ConnectionRefusedError** for a TCP connection, since the *server socket application we are trying to connect to is not listening for requests at the same port number as the one the client socket application is trying to connect* with. Hence, the connection between the two socket interfaces is never setup and the connection is downright refused.

However, on a UDP connection, *since no prior connection is required to be established between the host machines for data transfer to take place, no error as such is obtained*. Any messages sent by the client are lost since the destination server does not exist.

2. Task 2: Web Server

In this assignment, you will develop a simple Web server in Python that is capable of processing only one request. Specifically, your Web server will

For this assignment, the companion Web site provides the skeleton code for your server.

Your job is to complete the code, run your server, and then test your server by sending requests from browsers running on different hosts. If you run your server on a host that already has a web server .

Web server running on it, then you should use a different port than port 80 for your Web server.

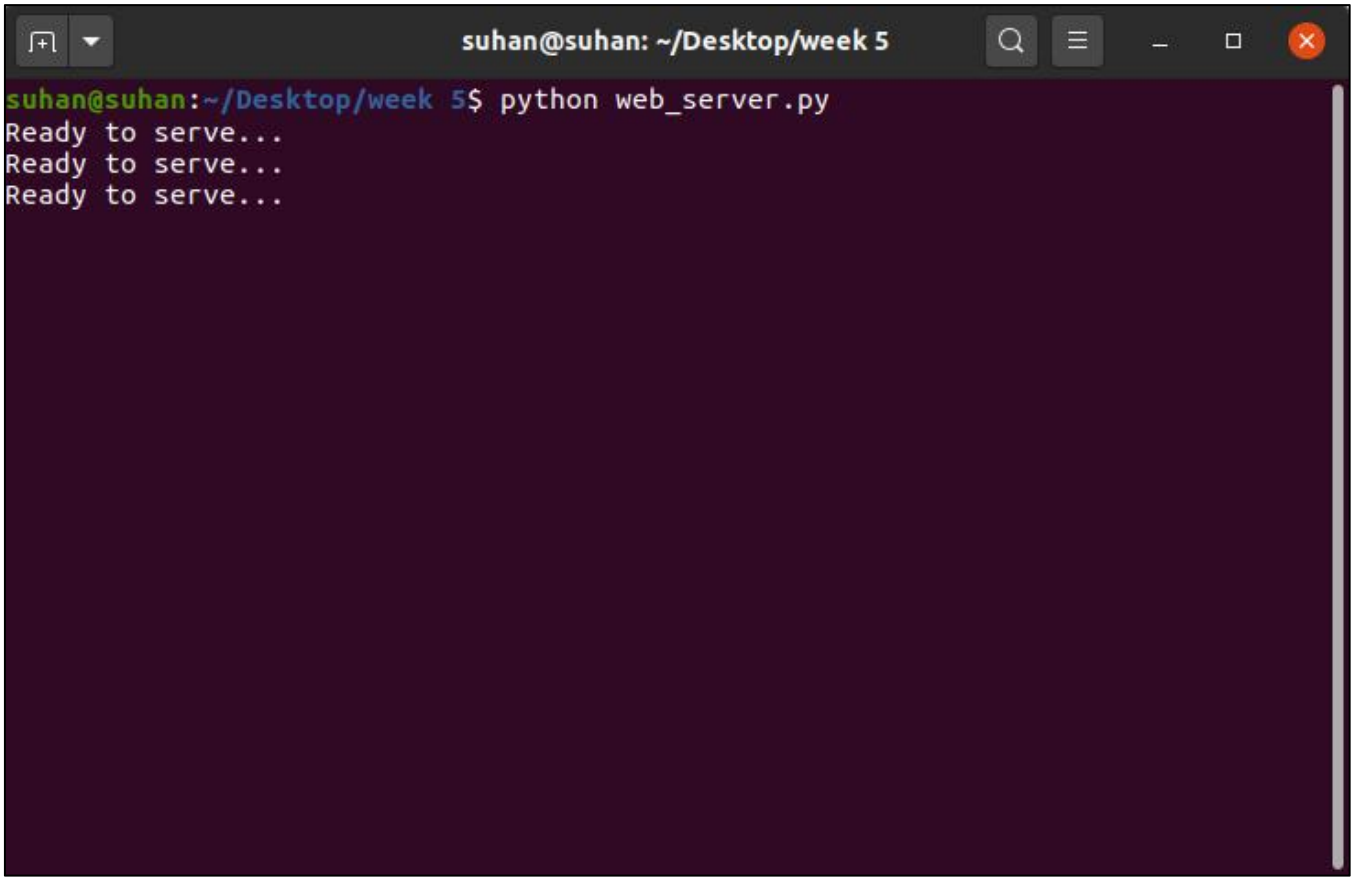
a) <u>create a connection socket when contacted by a client (browser);</u>

web_server.py
~/Desktop/week 5

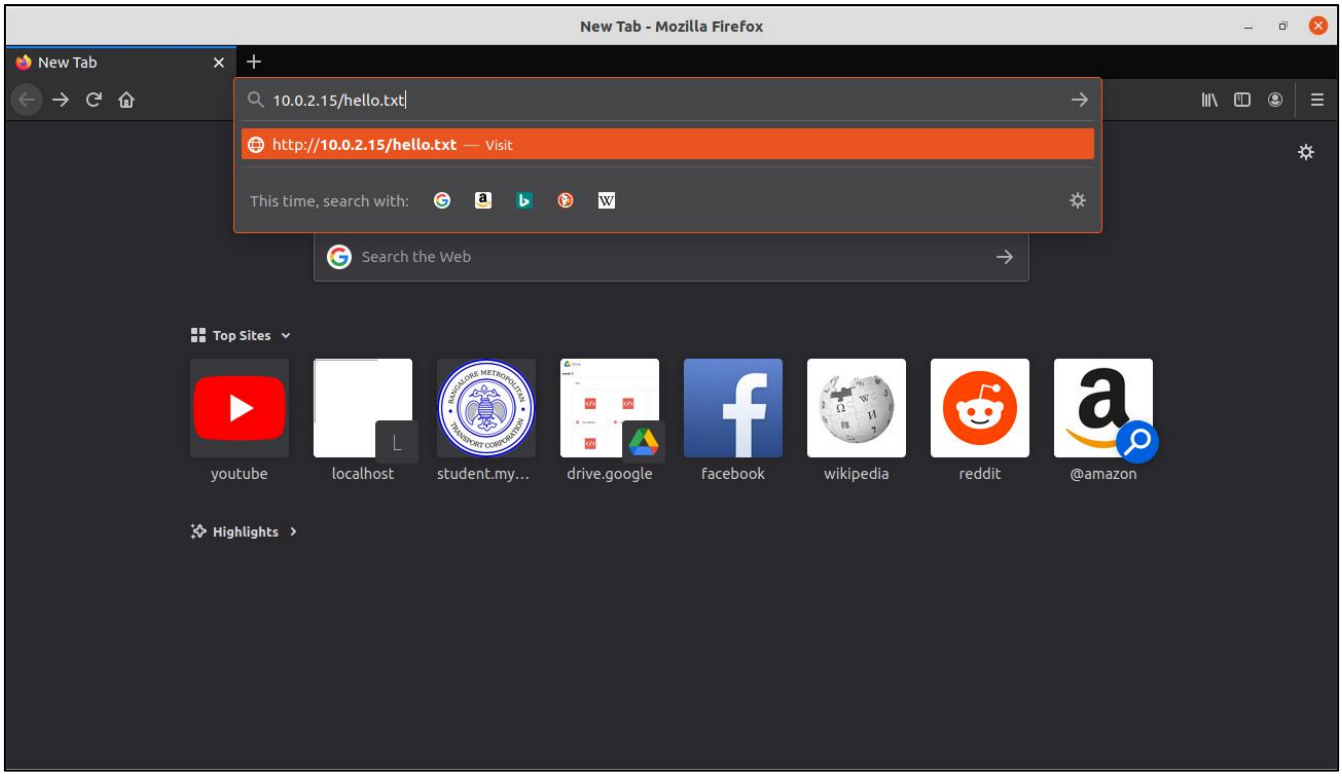
```python
# Import socket module
from socket import *

# 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 = 3000

# Bind the socket to server address and server port
serverSocket.bind(("10.0.2.15", 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()

        except IOError:
                # Send HTTP response message for file not found
                connectionSocket.send("HTTP/1.1 404 Not Found\r\n\r\n")
                connectionSocket.send("<html><head></head><body><h1>404 Not Found</h1></body></html>\r\n")
                # Close the client connection socket
                connectionSocket.close()

serverSocket.close()
```

Python Tab Width: 8 Ln 1, Col 1 INS

b) receive the HTTP request from this connection;

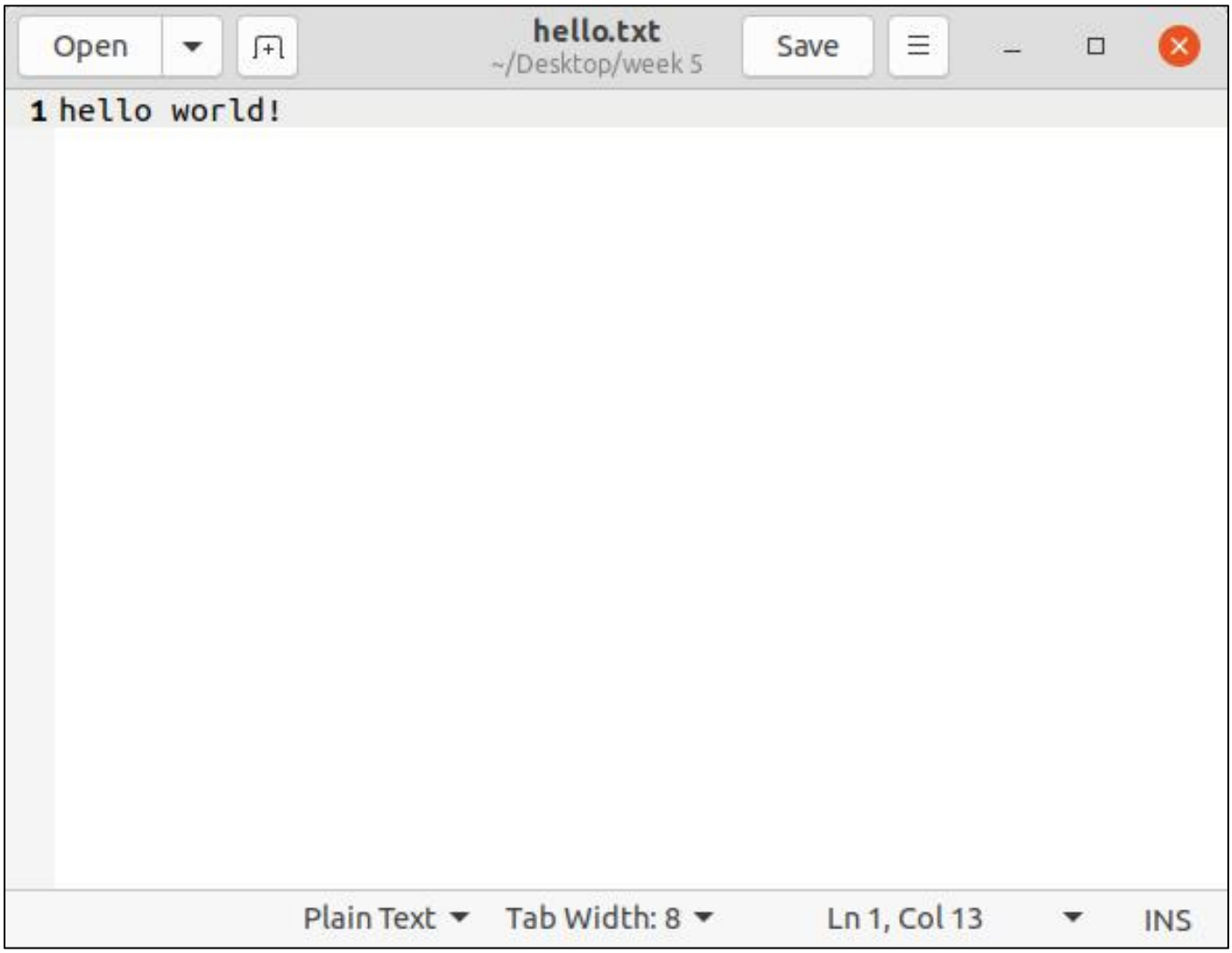


C) parse the request to determine the specific file being requested;



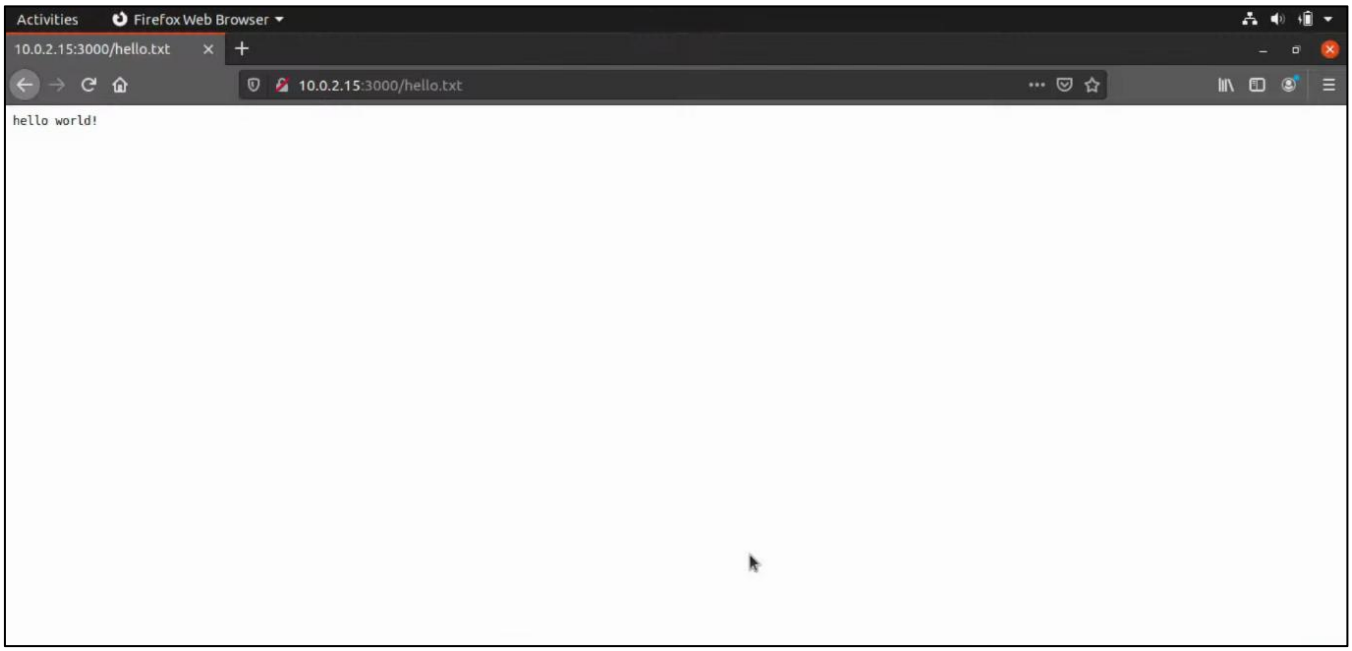
Web Browser requesting the file in server

D) <u>get the requested file from the server's file system;</u>



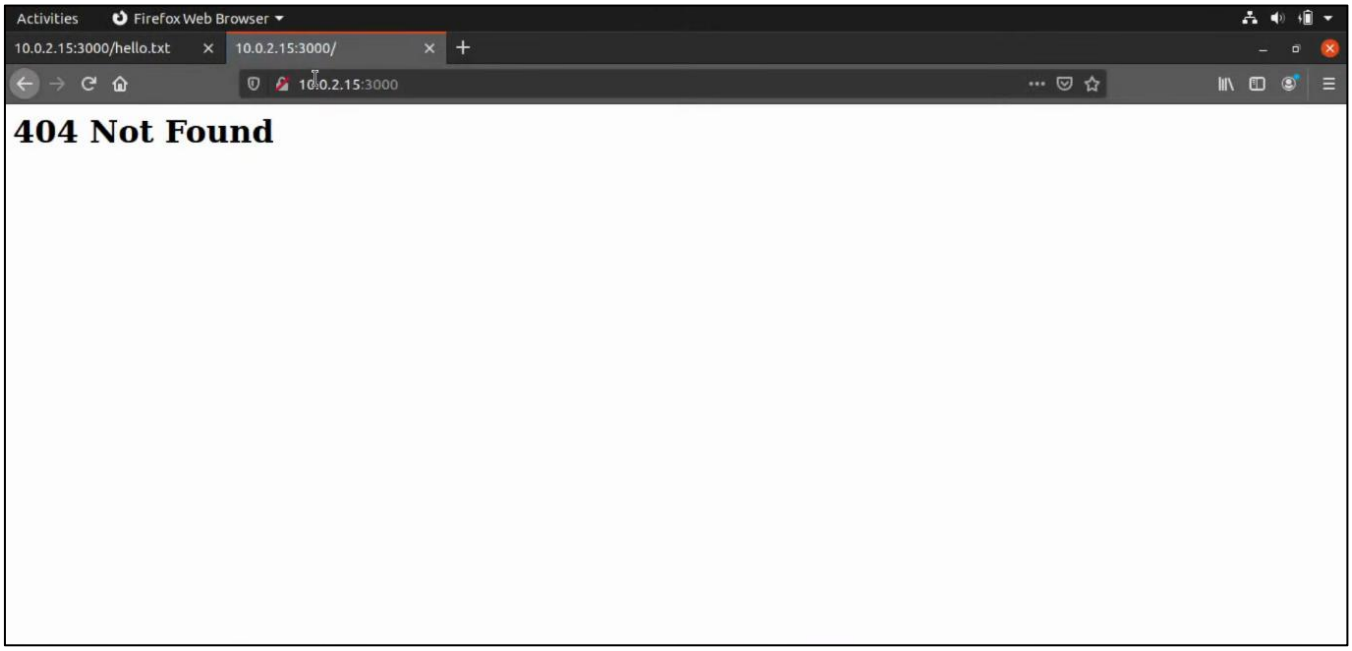
Data File consisting message present in same folder

e) create an HTTP response message consisting of the requested file preceded by header lines;



Message in file being displayed in web browser

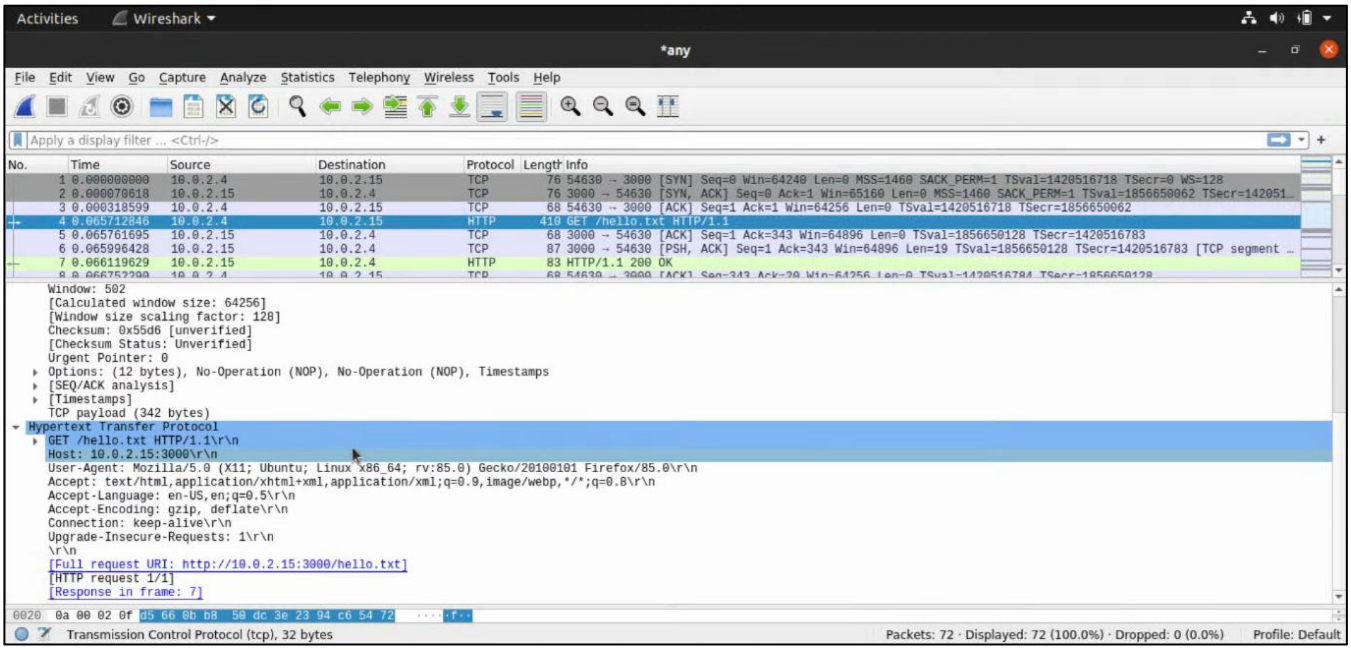
f) send the response over the TCP connection to the requesting browser.
If a browser requests a file that is not present in your server, your server should return a **“404 Not Found” error message.**

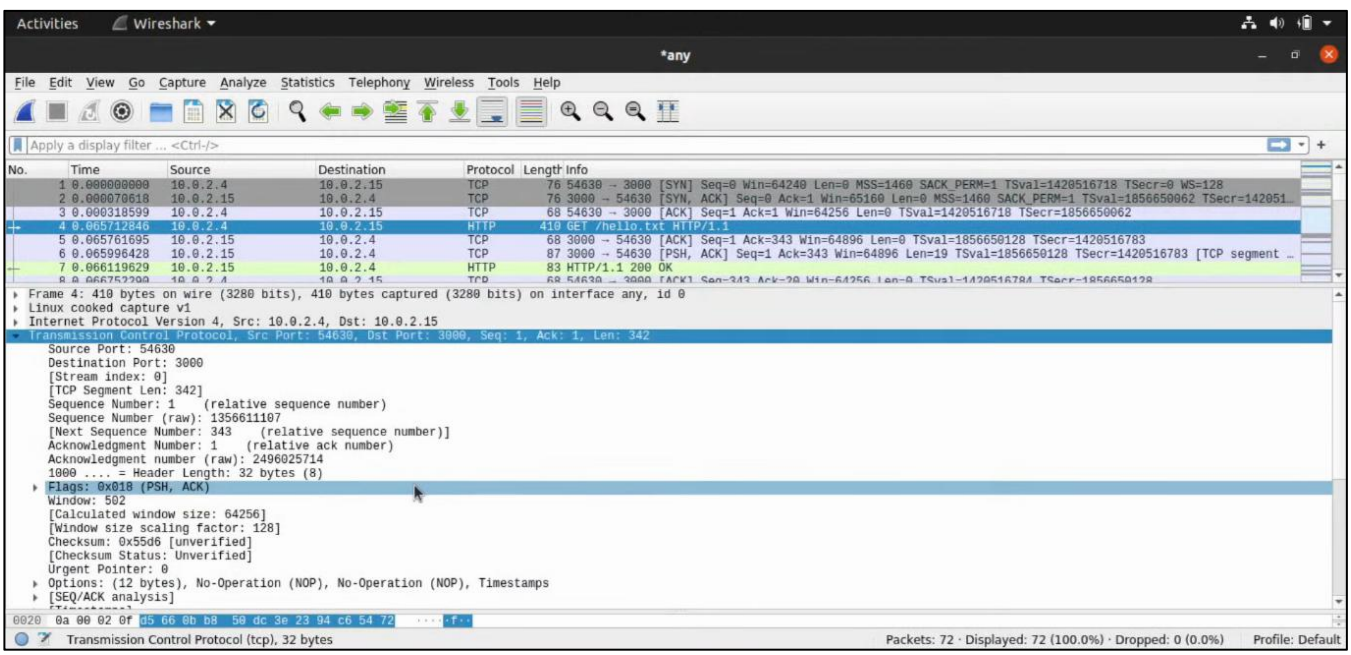


Message displayed for invalid

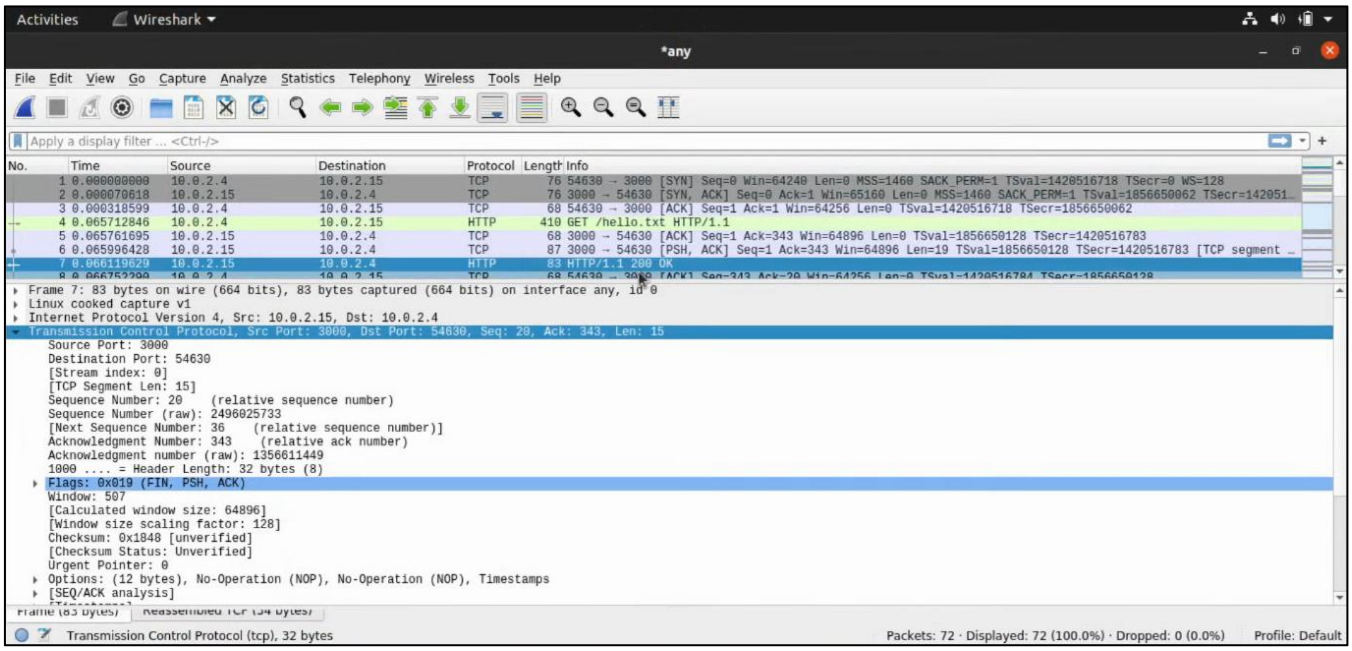
2.3 Wireshark Capture for Web Server

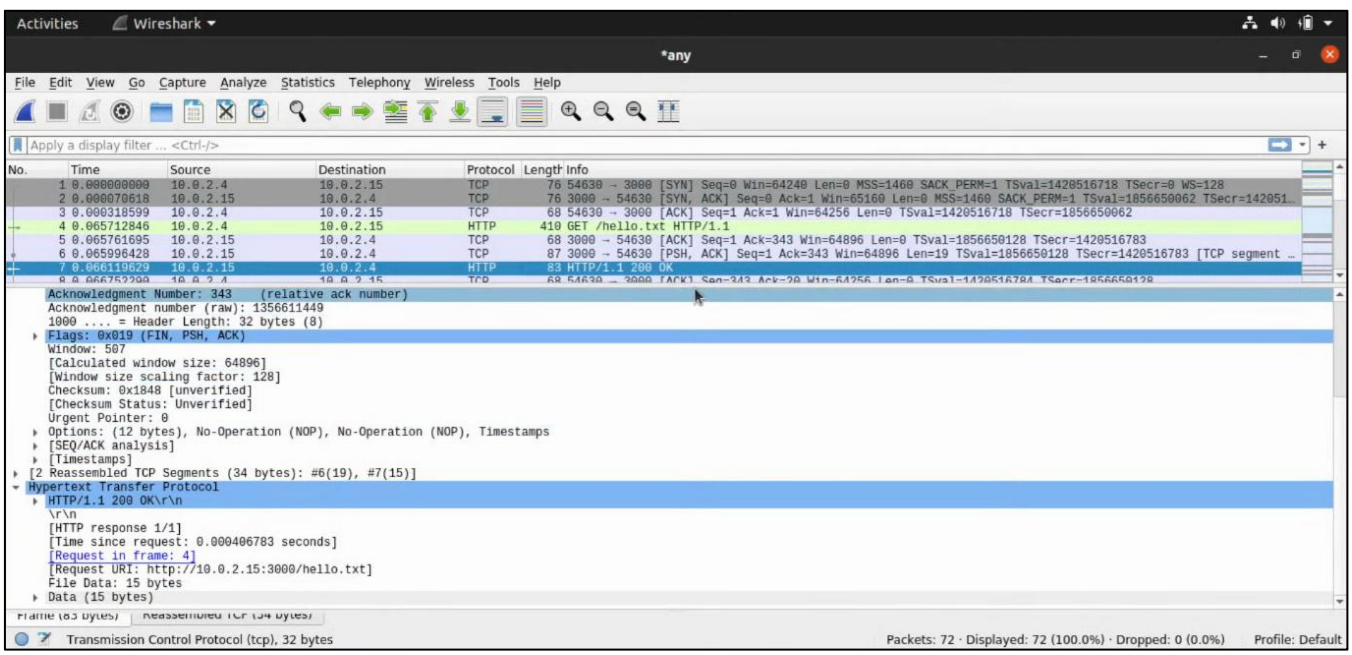
The Wireshark Packet Captures similarly shows the request and response packets received by the web server from the server and client, respectively..





Wireshark Capture for Server Machine





Wireshark Capture for Client Machine