

# Computer Networks Lab

UE19CS256

Week 5

Name: Sreenath Saikumar

Semester: 4    Section: G

**SRN:** PES2UG19CS406

Date: 27/02/21

## Objective:

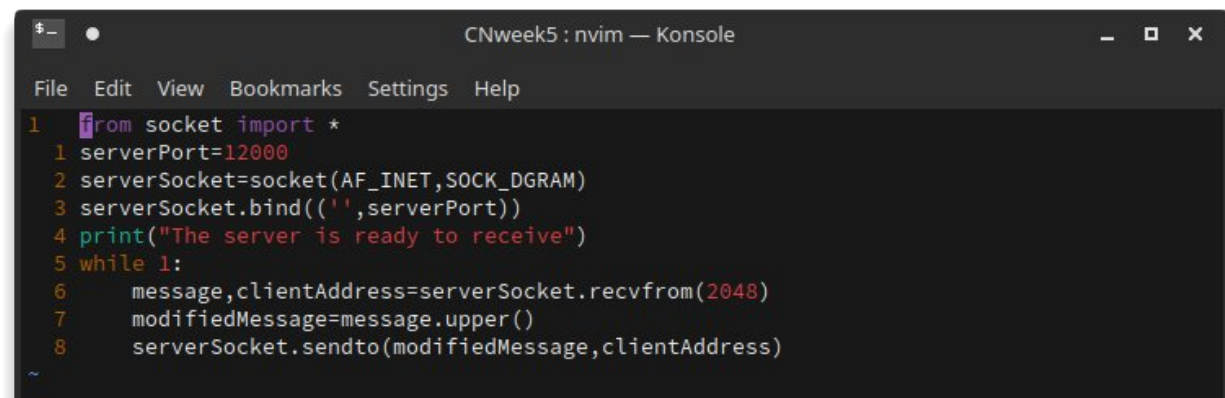
To develop a simple Client-Server application using TCP and UDP.

### Task 1:

- 1) Create an application that will:
  - a) Convert lowercase letters to uppercase
    - i. e.g. [a..z] to [A..Z]
    - ii. 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 for both client and server.
- 3) Must take server address and port from CLI.

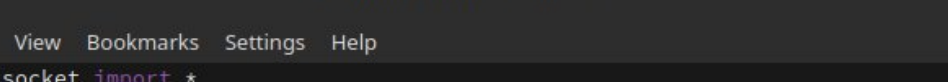
### Socket Programming with UDP:

#### UDPServer.py



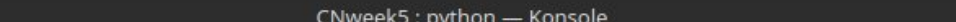
```
$ - CNweek5 : nvim — Konsole
File Edit View Bookmarks Settings Help
1 from socket import *
2 serverPort=12000
3 serverSocket=socket(AF_INET,SOCK_DGRAM)
4 serverSocket.bind('',serverPort)
5 print("The server is ready to receive")
6 while 1:
7     message,clientAddress=serverSocket.recvfrom(2048)
8     modifiedMessage=message.upper()
9     serverSocket.sendto(modifiedMessage,clientAddress)
10
```

## UDPClient.py



```
$ _ CNweek5 : nvim — Konsole
File Edit View Bookmarks Settings Help
1 from socket import *
2 serverName='192.168.0.105'
3 serverPort=12000
4 clientSocket=socket(AF_INET,SOCK_DGRAM)
5 message=raw_input("Enter lowercase sentence:")
6 clientSocket.sendto(message,(serverName,serverPort))
7 modifiedMessage,serverAddress=clientSocket.recvfrom(2048)
8 print(modifiedMessage)
9 clientSocket.close()
~
~
```

### UDP Connection between Server and Client:



```
$ _ CNweek5 : python — Konsole
File Edit View Bookmarks Settings Help
screenath in ~/Downloads/CNweek5 λ python UDPServer.py
The server is ready to receive
```

## Running UDPServer.py in a terminal instance

The screenshot shows a terminal window titled "CNweek5 : zsh — Konsole". The menu bar includes File, Edit, View, Bookmarks, Settings, and Help. Three instances of running the script are shown:

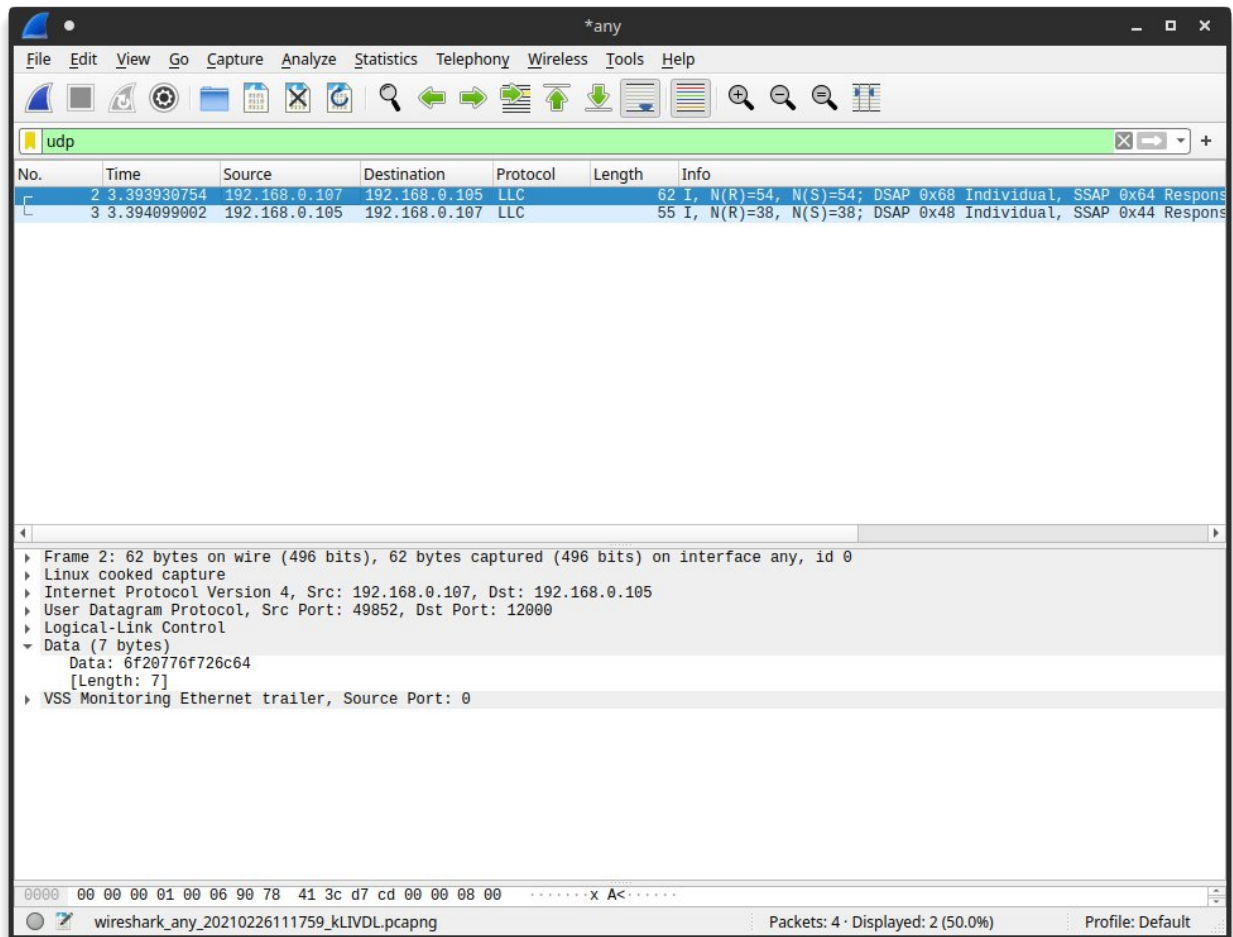
```
sreenath in ~/Downloads/CNweek5 λ python UDPClient.py
Enter lowercase sentence:hello world
HELLO WORLD
```

```
sreenath in ~/Downloads/CNweek5 λ python UDPClient.py
Enter lowercase sentence:testing udp
TESTING UDP
```

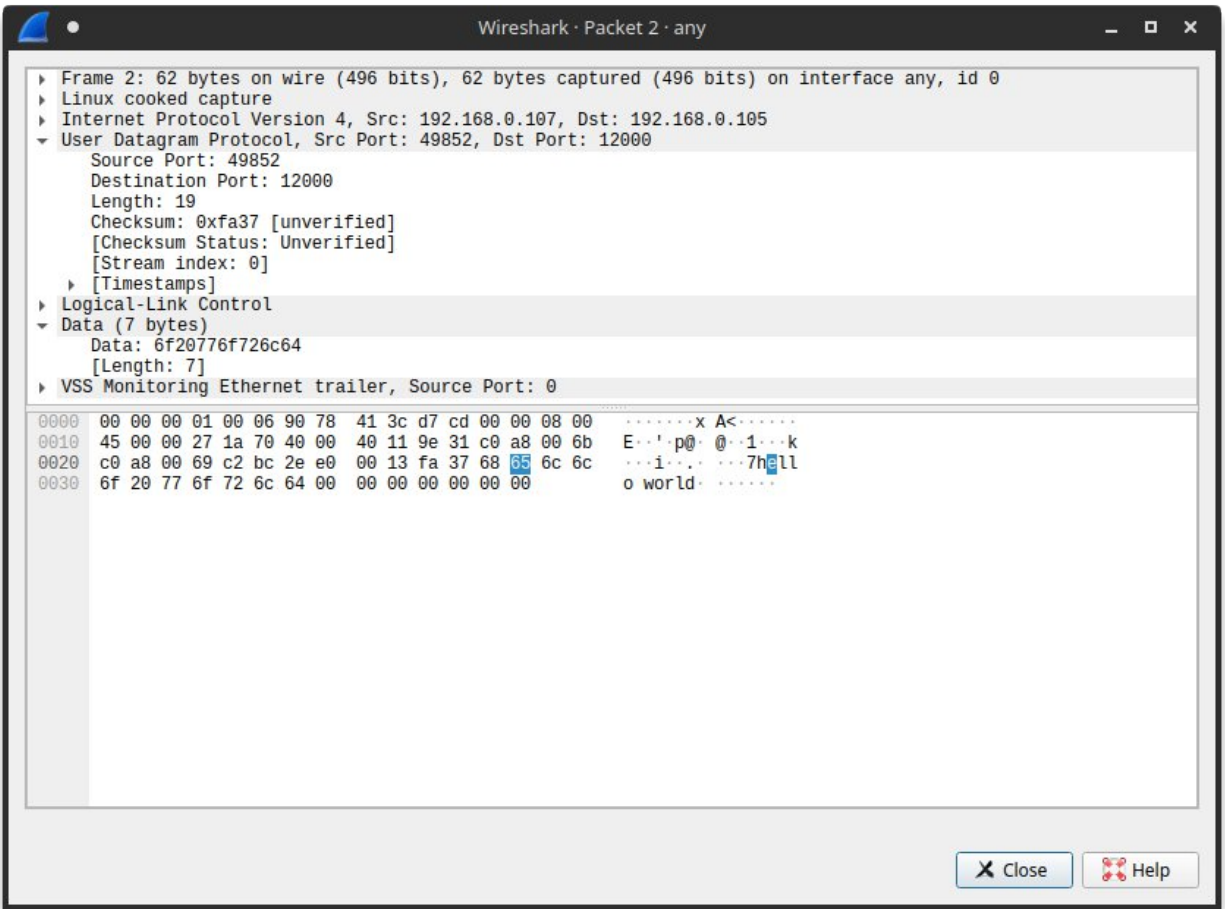
```
sreenath in ~/Downloads/CNweek5 λ python UDPClient.py
Enter lowercase sentence:abababababababababa
ABABABABABABABABABA
```

In each case, the input is correctly capitalized or converted to uppercase as expected by the program's logic.

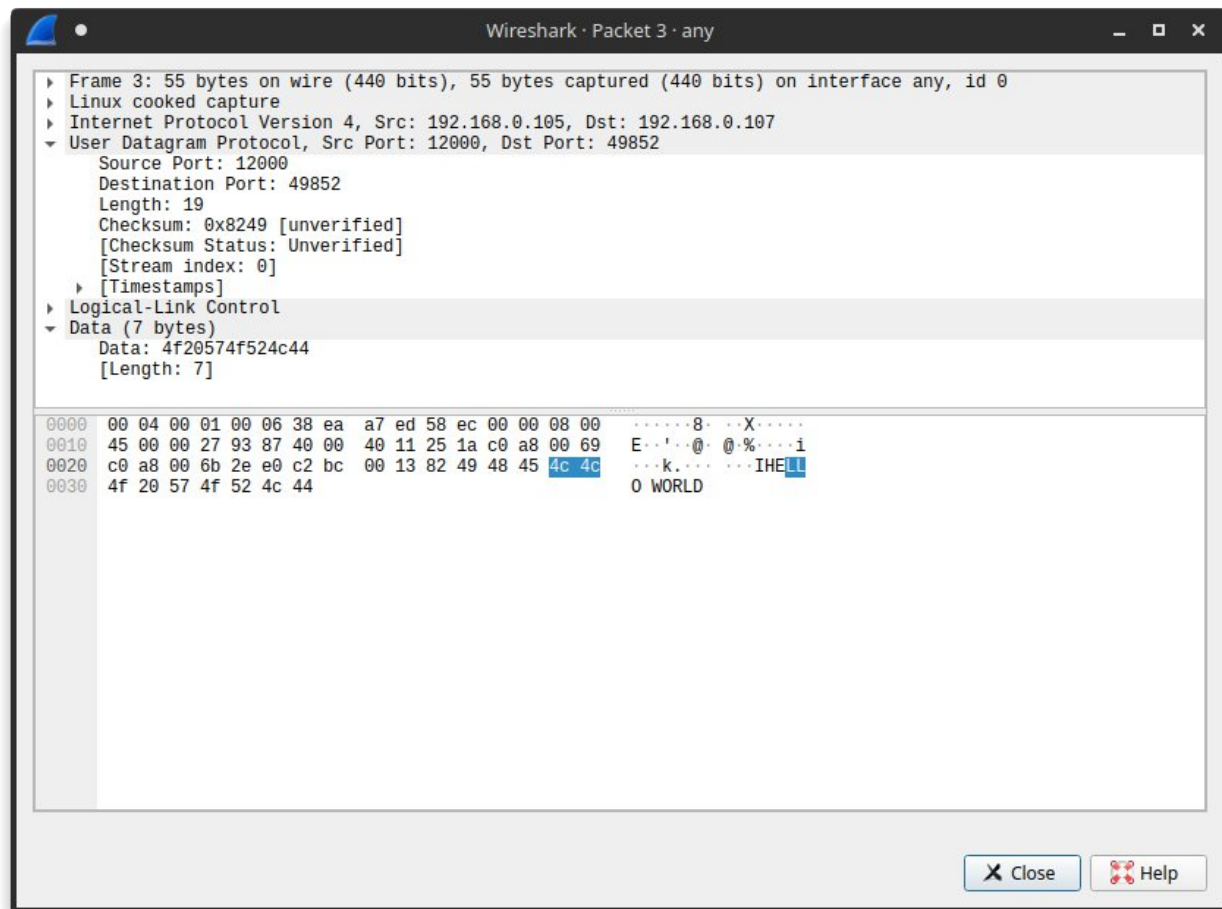
### Running UDPClient.py in another terminal instance



Wireshark capture



Client to Server



Server to Client

## Socket Programming with TCP

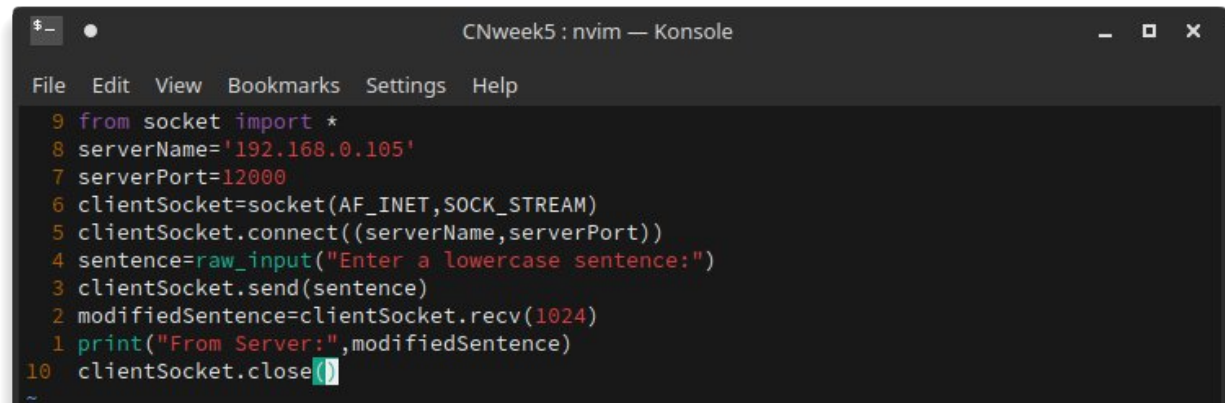
### TCPServer.py

```

CNweek5 : nvim — Konsole <2>
File Edit View Bookmarks Settings Help
11 from socket import *
10 serverPort=12000
9 serverSocket=socket(AF_INET,SOCK_STREAM)
8 serverSocket.bind(('',serverPort))
7 serverSocket.listen(1)
6 print("The server is ready to receive")
5 while 1:
4     connectionSocket,addr=serverSocket.accept()
3     sentence=connectionSocket.recv(1024)
2     capitalizedSentence=sentence.upper()
1     connectionSocket.send(capitalizedSentence)
12 connectionSocket.close()

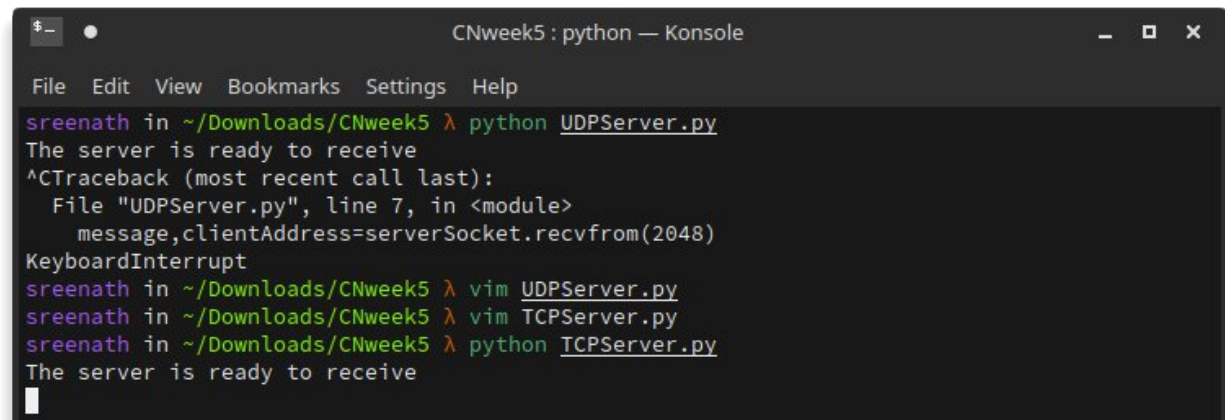
```

## TCPClient.py



```
$ - CNweek5 : nvim — Konsole
File Edit View Bookmarks Settings Help
9 from socket import *
8 serverName='192.168.0.105'
7 serverPort=12000
6 clientSocket=socket(AF_INET,SOCK_STREAM)
5 clientSocket.connect((serverName,serverPort))
4 sentence=raw_input("Enter a lowercase sentence:")
3 clientSocket.send(sentence)
2 modifiedSentence=clientSocket.recv(1024)
1 print("From Server:",modifiedSentence)
10 clientSocket.close()
```

TCP Connection between Server and Client:



```
$ - CNweek5 : python — Konsole
File Edit View Bookmarks Settings Help
sreenath in ~/Downloads/CNweek5 λ python UDPServer.py
The server is ready to receive
^CTraceback (most recent call last):
  File "UDPServer.py", line 7, in <module>
    message,clientAddress=serverSocket.recvfrom(2048)
KeyboardInterrupt
sreenath in ~/Downloads/CNweek5 λ vim UDPServer.py
sreenath in ~/Downloads/CNweek5 λ vim TCPServer.py
sreenath in ~/Downloads/CNweek5 λ python TCPServer.py
The server is ready to receive
```

TCP Server running in one terminal instance



```
CNweek5: zsh — Konsole
File Edit View Bookmarks Settings Help
sreenath in ~/Downloads/CNweek5 λ vim UDPClient.py
sreenath in ~/Downloads/CNweek5 λ vim TCPClient.py
sreenath in ~/Downloads/CNweek5 λ python TCPClient.py
Enter a lowercase sentence:hello
('From Server:', 'HELLO')
sreenath in ~/Downloads/CNweek5 λ python TCPClient.py
Enter a lowercase sentence:test
('From Server:', 'TEST')
sreenath in ~/Downloads/CNweek5 λ python TCPClient.py
Enter a lowercase sentence:hello world
('From Server:', 'HELLO WORLD')
sreenath in ~/Downloads/CNweek5 λ
```

TCP Client in another terminal instance

The Wireshark capture shows a TCP connection established between 192.168.0.107 and 192.168.0.105. The connection is initiated by a SYN packet from 192.168.0.107 to 192.168.0.105. The server responds with a SYN, ACK packet. The client then sends a PSH, ACK packet. The server responds with an ACK packet. The client then sends a FIN packet, and the server responds with a FIN, ACK packet. The connection is then closed.

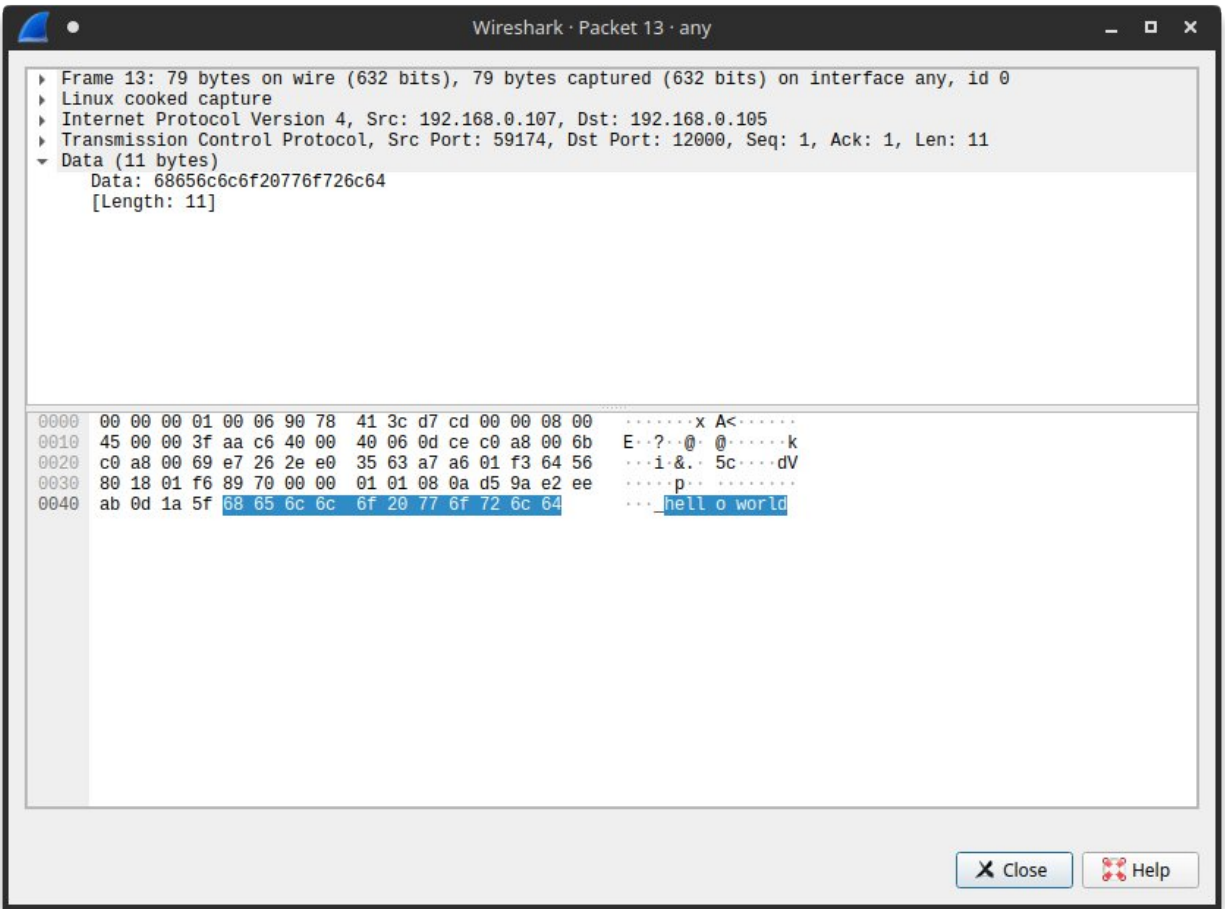
No.	Time	Source	Destination	Protocol	Length	Info
4	9.459423129	192.168.0.107	192.168.0.105	TCP	76	59174 → 12000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM
5	9.459478222	192.168.0.105	192.168.0.107	TCP	76	12000 → 59174 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460
6	9.463210673	192.168.0.107	192.168.0.105	TCP	68	59174 → 12000 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=358366
13	17.142700274	192.168.0.107	192.168.0.105	TCP	79	59174 → 12000 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=11 TSval=
14	17.142845627	192.168.0.105	192.168.0.107	TCP	68	12000 → 59174 [ACK] Seq=1 Ack=12 Win=65152 Len=0 TSval=28697
15	17.142952536	192.168.0.105	192.168.0.107	TCP	79	12000 → 59174 [PSH, ACK] Seq=1 Ack=12 Win=65152 Len=11 TSval=
16	17.146121061	192.168.0.107	192.168.0.105	TCP	68	59174 → 12000 [ACK] Seq=12 Ack=12 Win=64256 Len=0 TSval=3583
17	17.146271980	192.168.0.107	192.168.0.105	TCP	68	59174 → 12000 [FIN, ACK] Seq=12 Ack=12 Win=64256 Len=0 TSval=
18	17.187448160	192.168.0.105	192.168.0.107	TCP	68	12000 → 59174 [ACK] Seq=12 Ack=13 Win=65152 Len=0 TSval=2869
20	20.616148026	192.168.0.105	192.168.0.107	TCP	68	12000 → 59174 [FIN, ACK] Seq=12 Ack=13 Win=65152 Len=0 TSval=
21	20.701623490	192.168.0.107	192.168.0.105	TCP	68	59174 → 12000 [ACK] Seq=13 Ack=13 Win=64256 Len=0 TSval=3583

Frame 13: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface any, id 0  
Linux cooked capture  
Internet Protocol Version 4, Src: 192.168.0.107, Dst: 192.168.0.105  
Transmission Control Protocol, Src Port: 59174, Dst Port: 12000, Seq: 1, Ack: 1, Len: 11  
Data (11 bytes)  
Data: 68656c6c6f20776f726c64  
[Length: 11]

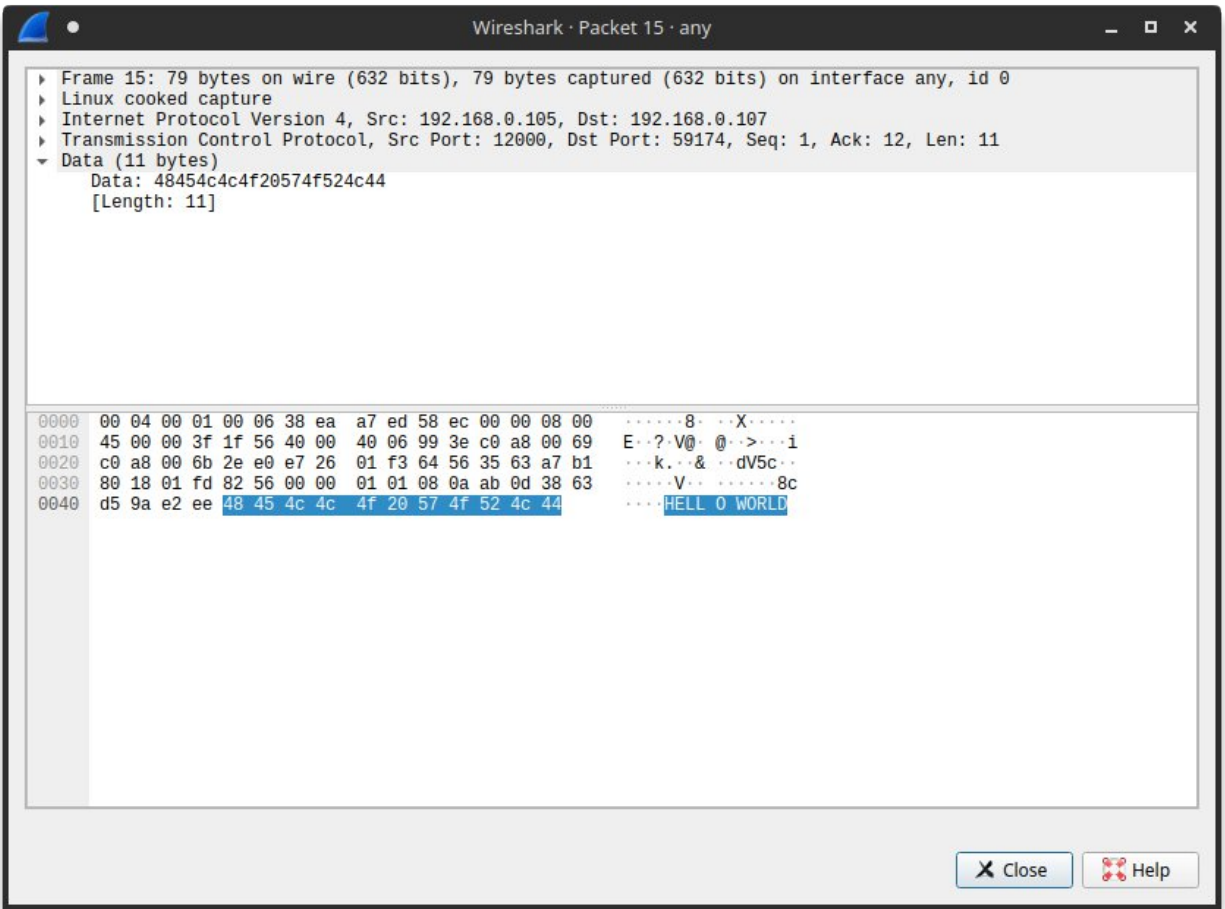
wireshark\_any\_20210226111546\_1jg2Nu.pcapng Packets: 21 · Displayed: 11 (52.4%) · Dropped: 0 (0.0%) Profile: Default

Wireshark Capture





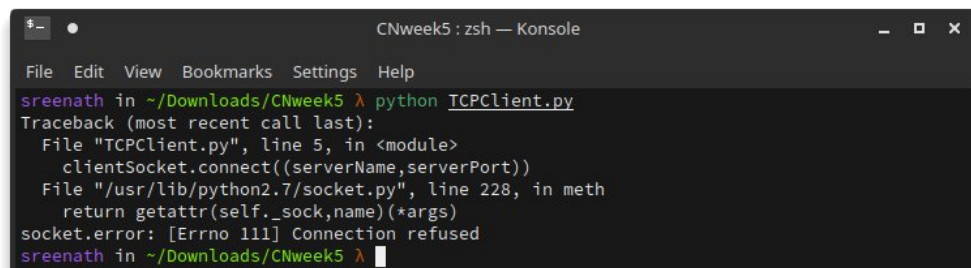
Client to Server



Server to Client

## Problems:

1. Suppose you run TCPClient before you run TCPServer. What happens?  
Why?
  - a. We get a `ConnectionRefusedError` since the server socket application hasn't been started and isn't listening on the given port number. Hence, any connection request sent from a client machine to the specified IP and port is immediately refused.

A screenshot of a terminal window titled "CNweek5 : zsh — Konsole". The terminal shows a Python script execution that results in a connection refusal error. The prompt is "sreenath in ~/Downloads/CNweek5". The command executed is "python TCPClient.py". The output is a traceback showing the error occurred in "TCPClient.py" at line 5, in the module, specifically in the "clientSocket.connect((serverName,serverPort))" call. It further shows the error in "socket.py" at line 228, in the "meth" function, where it returns "getattr(self.\_sock,name)(\*args)". The final error message is "socket.error: [Errno 111] Connection refused". The prompt returns to "sreenath in ~/Downloads/CNweek5".

```
File "TCPClient.py", line 5, in <module>
  clientSocket.connect((serverName,serverPort))
File "/usr/lib/python2.7/socket.py", line 228, in meth
  return getattr(self._sock,name)(*args)
socket.error: [Errno 111] Connection refused
sreenath in ~/Downloads/CNweek5
```

2. Suppose you run UDPClient before you run UDPServer. What happens?  
Why?
  - a. Since UDP doesn't require any prior connection to be established, we will not get any errors as UDP is connectionless and transfers data to a destination IP and port without verifying whether the connection exists. If any packets of data are sent while the server isn't running, the packets are simply lost.
3. What happens if you use different port numbers for the client and server sides?
  - a. We obtain a `ConnectionRefusedError` for a TCP connection since the server socket application isn't listening for requests on the same port number that the client is sending data to.

However, on a UDP Connection, by virtue of it's connectionless behaviour, all data sent by the client is just lost since the destination doesn't exist.

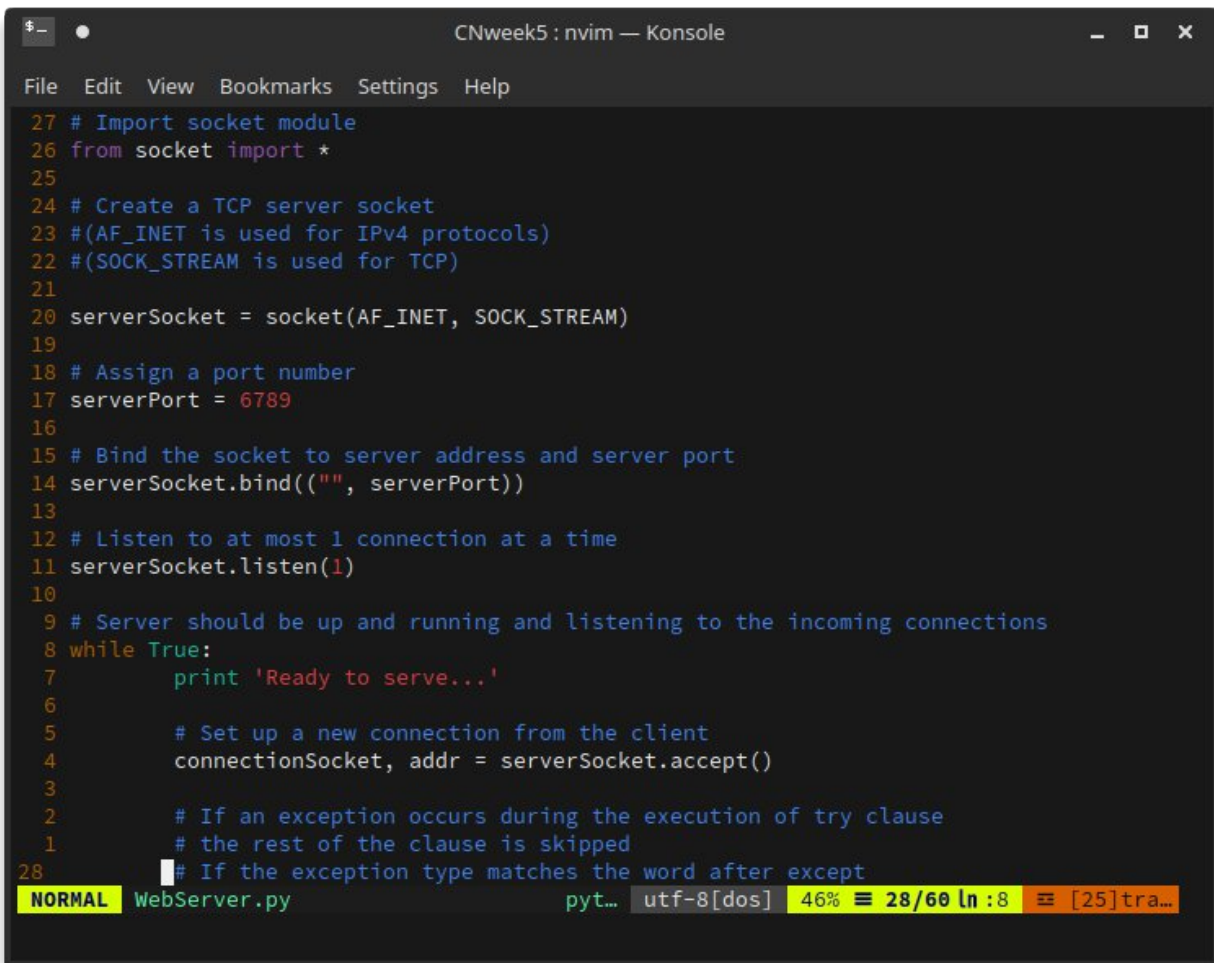
## Task 2: Web Server

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

- a) create a connection socket when contacted by a client (browser);
- b) receive the HTTP request from this connection;
- c) parse the request to determine the specific file being requested;
- d) get the requested file from the server's file system;
- e) create an HTTP response message consisting of the requested file preceded by header lines; and
- 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.

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

# WebServer.py



```
$ _ CNweek5 : nvim — Konsole
File Edit View Bookmarks Settings Help
27 # Import socket module
26 from socket import *
25
24 # Create a TCP server socket
23 #(AF_INET is used for IPv4 protocols)
22 #(SOCK_STREAM is used for TCP)
21
20 serverSocket = socket(AF_INET, SOCK_STREAM)
19
18 # Assign a port number
17 serverPort = 6789
16
15 # Bind the socket to server address and server port
14 serverSocket.bind(('', serverPort))
13
12 # Listen to at most 1 connection at a time
11 serverSocket.listen(1)
10
9 # Server should be up and running and listening to the incoming connections
8 while True:
7     print 'Ready to serve...'
6
5     # Set up a new connection from the client
4     connectionSocket, addr = serverSocket.accept()
3
2     # If an exception occurs during the execution of try clause
1     # the rest of the clause is skipped
28     # If the exception type matches the word after except
NORMAL WebServer.py          pyt... utf-8[dos] 46% 28/60 ln :8 [25]tra...
```

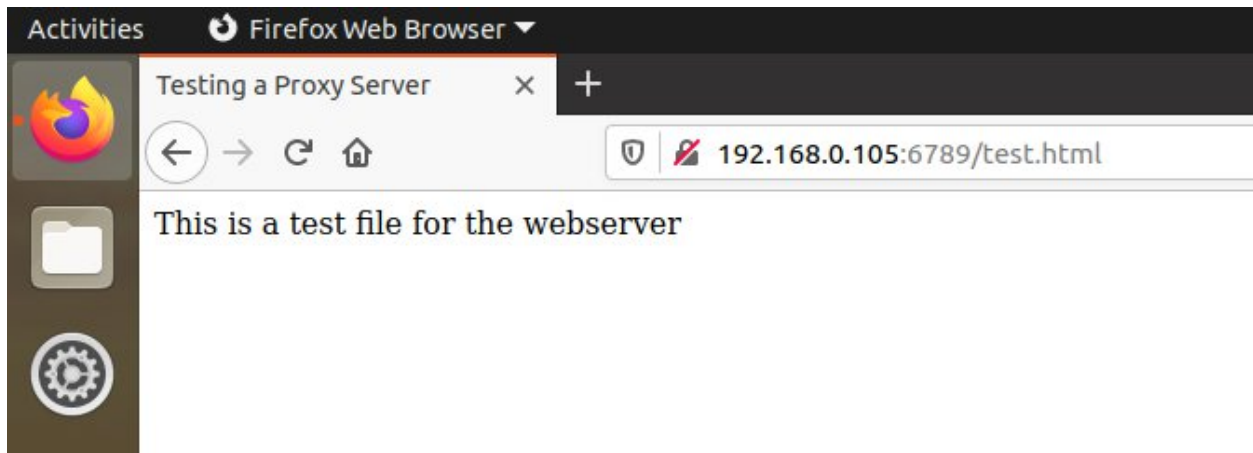
```
$ - CNweek5 : nvim — Konsole
File Edit View Bookmarks Settings Help
25     try:
24         # Receives the request message from the client
23         message = connectionSocket.recv(1024)
22         # Extract the path of the requested object from the message
21         # The path is the second part of HTTP header, identified by [1]
20         filename = message.split()[1]
19         # Because the extracted path of the HTTP request includes
18         # a character '\', we read the path from the second character
17         f = open(filename[1:])
16         # Store the entire content of the requested file in a temporary buff
er
15         outputdata = f.read()
14         # Send the HTTP response header line to the connection socket
13         connectionSocket.send("HTTP/1.1 200 OK\r\n\r\n")
12
11         # Send the content of the requested file to the connection socket
10         for i in range(0, len(outputdata)):
9             connectionSocket.send(outputdata[i])
8         connectionSocket.send("\r\n")
7
6         # Close the client connection socket
5         connectionSocket.close()
4
3     except IOError:
2         # Send HTTP response message for file not found
1         connectionSocket.send("HTTP/1.1 404 Not Found\r\n\r\n")
55         connectionSocket.send("<html><head></head><body><h1>404 Not Found</h1>
></body></html>\r\n")
NORMAL WebServer.py          pyt... utf-8[dos] 91% == 55/60 ln :8 == [25]tra...
```



```
CNweek5 : nvim — Konsole
File Edit View Bookmarks Settings Help
25     filename = message.split()[1]
24     # Because the extracted path of the HTTP request includes
23     # a character '\', we read the path from the second character
22     f = open(filename[1:])
21     # Store the entire content of the requested file in a temporary buff
er
20     outputdata = f.read()
19     # Send the HTTP response header line to the connection socket
18     connectionSocket.send("HTTP/1.1 200 OK\r\n\r\n")
17
16     # Send the content of the requested file to the connection socket
15     for i in range(0, len(outputdata)):
14         connectionSocket.send(outputdata[i])
13     connectionSocket.send("\r\n")
12
11     # Close the client connection socket
10     connectionSocket.close()
9
8     except IOError:
7         # Send HTTP response message for file not found
6         connectionSocket.send("HTTP/1.1 404 Not Found\r\n\r\n")
5         connectionSocket.send("<html><head></head><body><h1>404 Not Found</h1
></body></html>\r\n")
4         # Close the client connection socket
3         connectionSocket.close()
2
1 serverSocket.close()
60
NORMAL WebServer.py      pyt... utf-8[dos] 100% 60/60 ln :1 [25]tra...
```

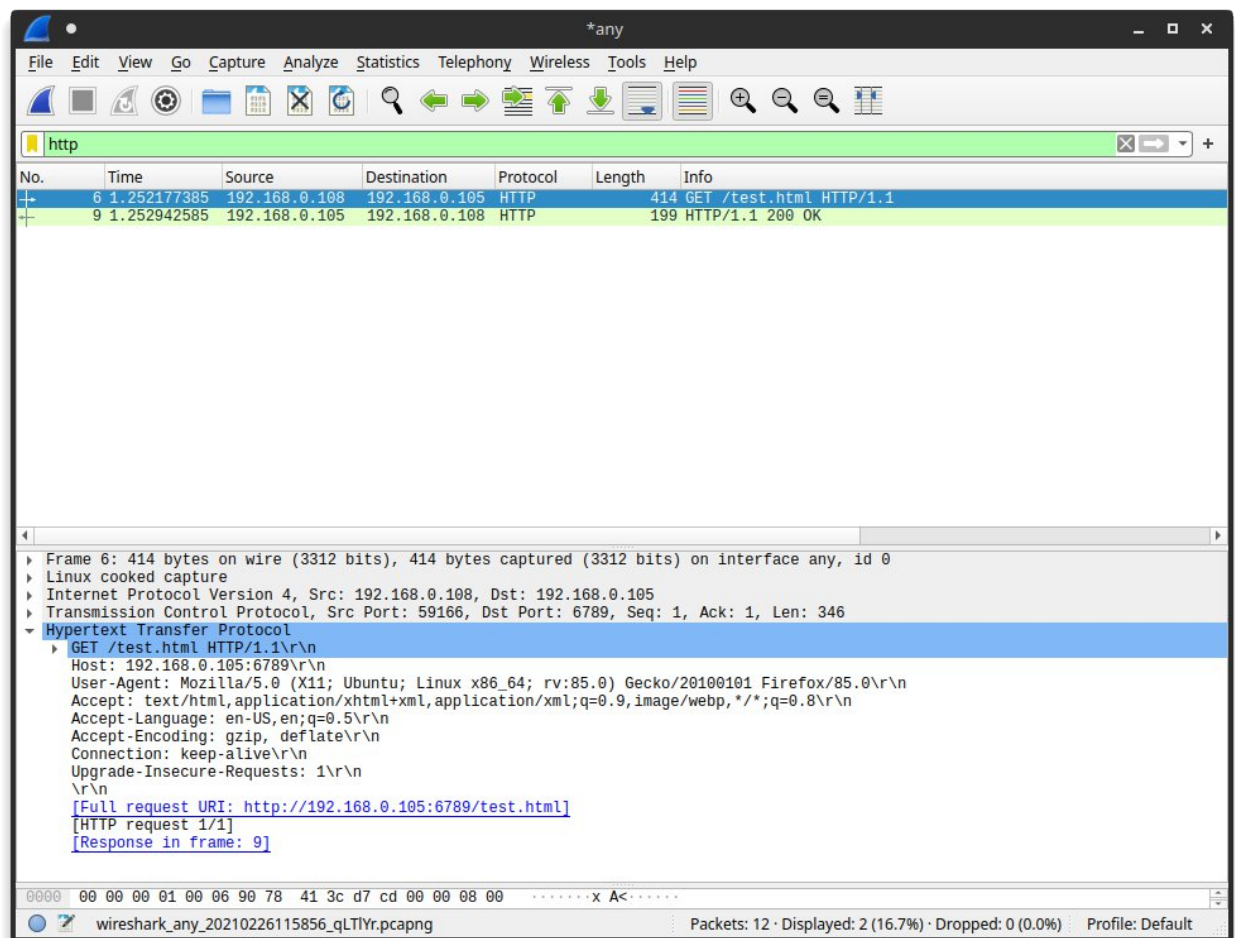
Source Code for Web Server.



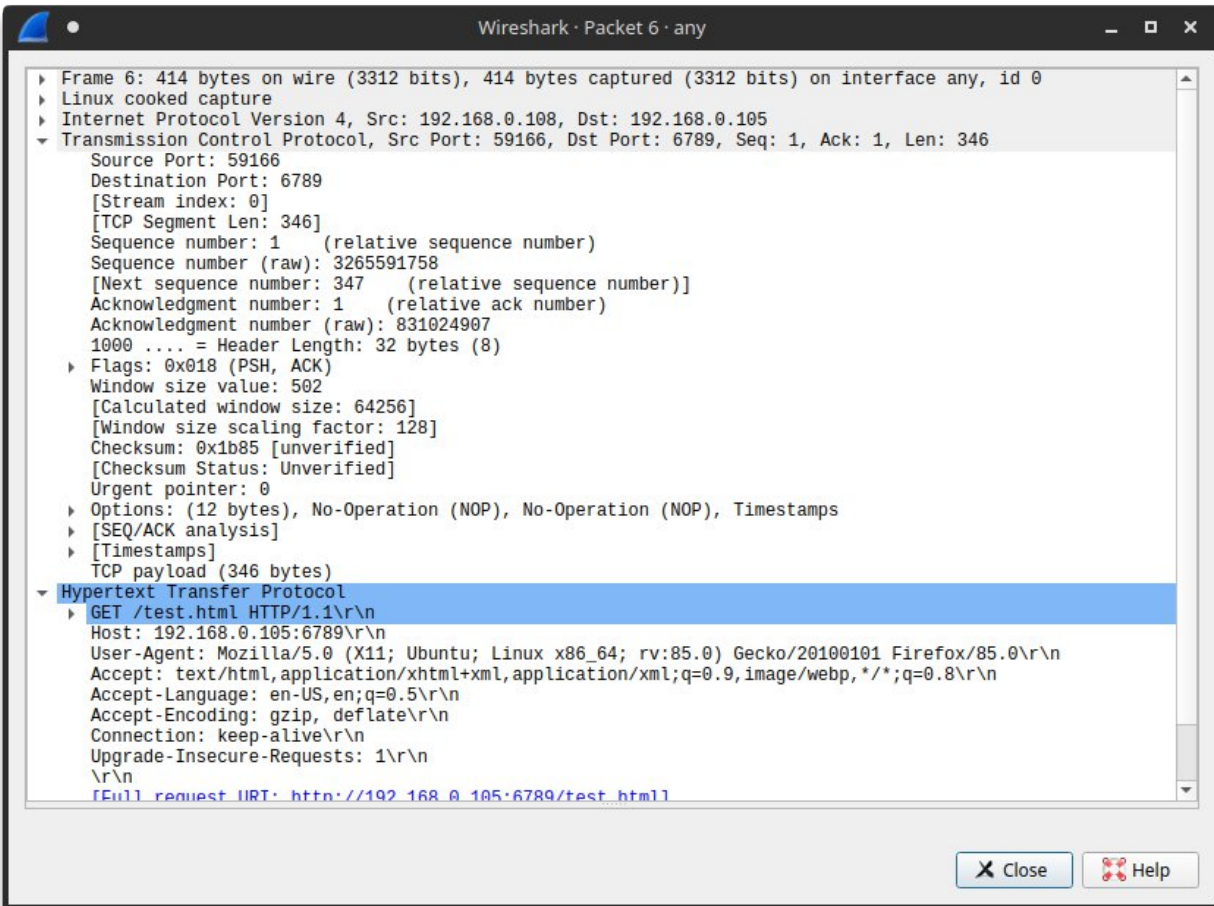


Accessing test.html from the client side

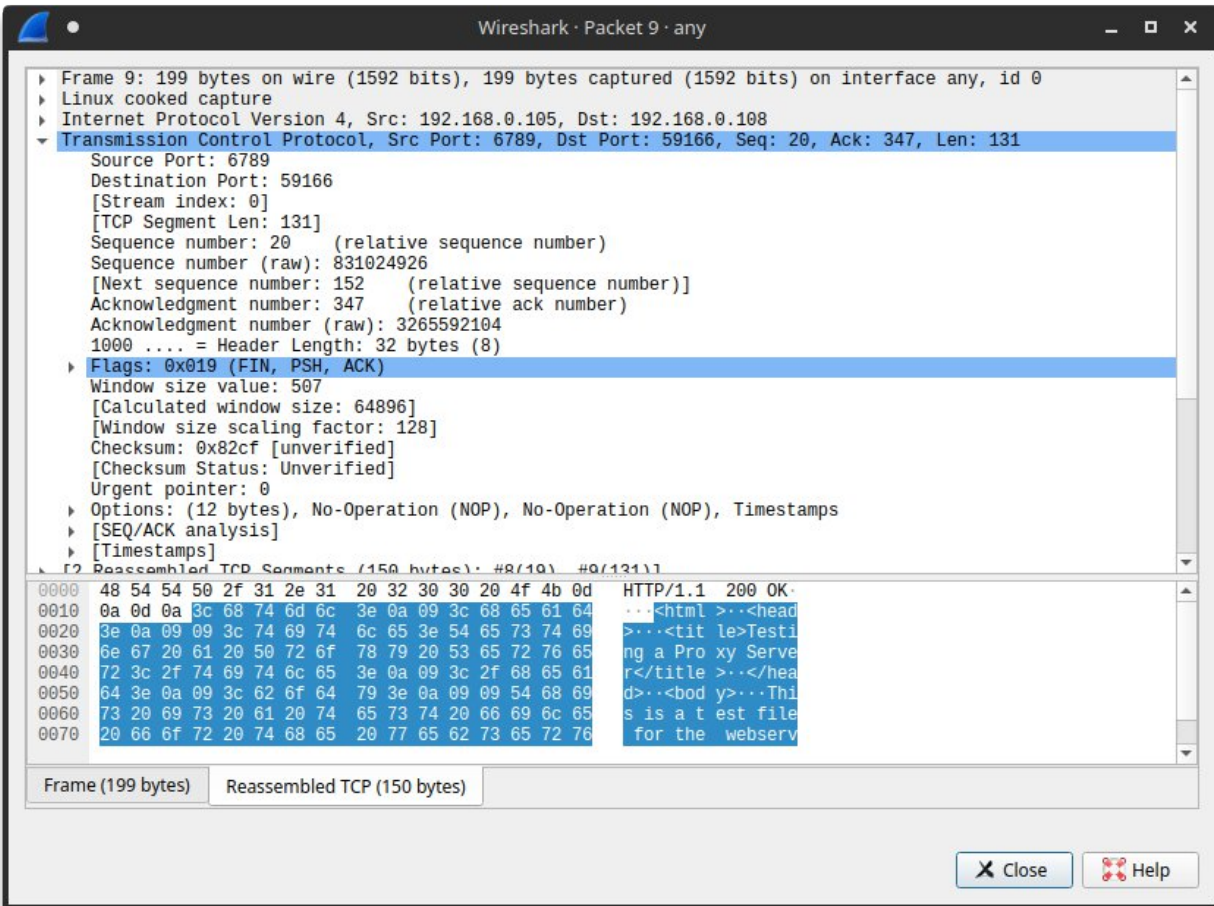
If the file doesn't exist on the server, we get a 404 Not found error.



Wireshark capture



HTTP request packet



## HTTP Response Packet