

COMPUTER NETWORKS LABORATORY – UE19CS255

WEEK5 – Simple Client-Server Application using Network Socket Programming

Date : 27/2/2021

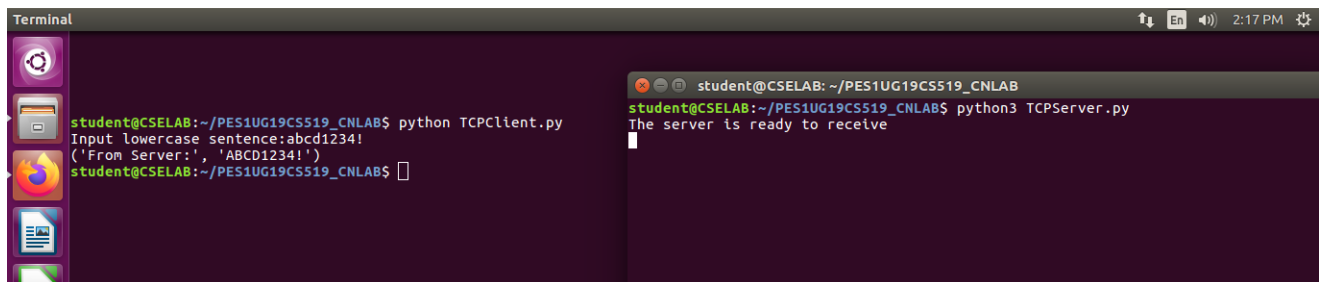
Name	SRN	Section
Sumukh Raju Bhat	PES1UG19CS519	H

TASK 1:

We set the servername/hostname as **localhost** and server script was run on one terminal and client on another.

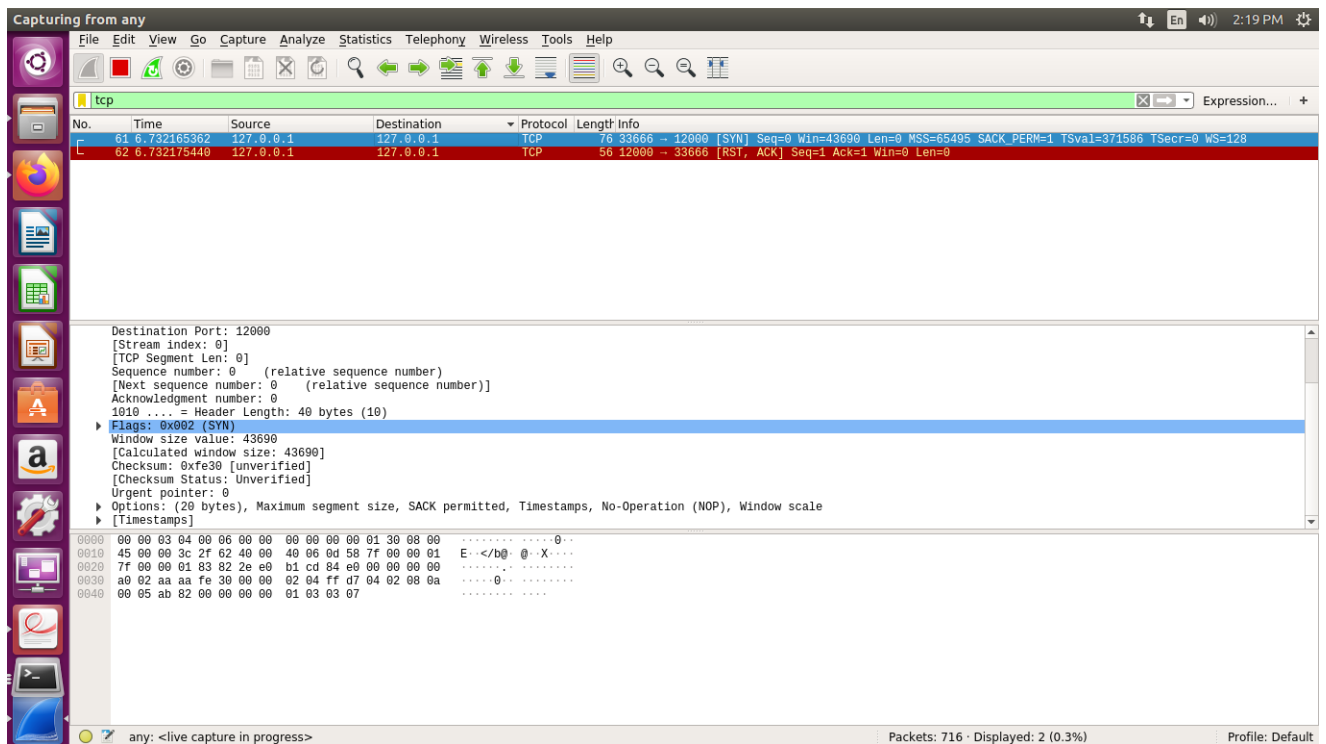
TCP:

To create a TCP socket interface, we need to use **SOCK_STREAM** as type of socket of python socket library. Type of address is set as **AF_INET which is IPv4**. We use **bind()** to bind a host IP and port number.

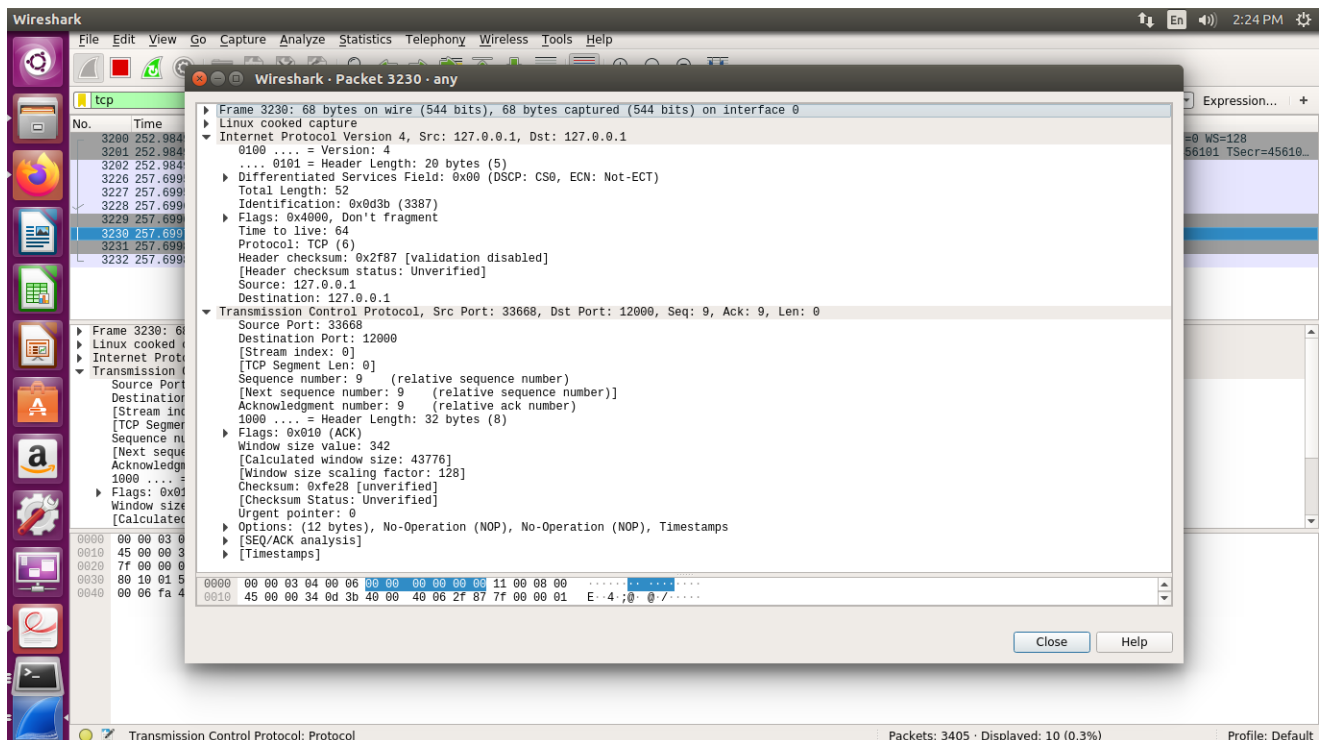


The image shows two terminal windows side-by-side. The left window is titled 'Terminal' and shows a user running 'python TCPClient.py'. The user inputs 'lowercase sentence:abcd1234!'. The output shows 'From Server:', 'ABCD1234!'. The right window is titled 'student@CSELAB: ~/PES1UG19CS519_CNLAB' and shows a user running 'python3 TCPServer.py'. The output shows 'The server is ready to receive'.

When server script is run on one terminal and client script on another, the characters written on client side gets converted to uppercase in the server side and gets printed on client. Both in my case via **localhost(127.0.0.1)** and **port 12000**.

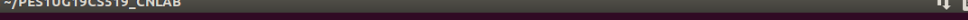


Request packet:



We can see both source and destination being 127.0.0.1(localhost), and serverport/dst port being 12000. TCP is being used. Similarly the response packet, but src port will be 12000 [As set in our python code as serverPort]

To create a **UDP** socket interface, we need to use **SOCK_DGRAM** as type of socket of python socket library. Type of address is set as **AF_INET** which is IPv4. We use **bind()** to bind a host IP and port number.



The screenshot shows a terminal window with a dark background. The prompt is `student@CSELAB: ~/PES1UG19CS519_CNLAB`. The user enters `gedit UDPClient.py` and `python UDPClient.py`. The program outputs `Input lowercase sentence:abcd1234!` and `ABCD1234!`. The terminal window has a sidebar on the left with icons for home, applications, and Firefox. The top bar shows system icons and the time 2:28 PM.

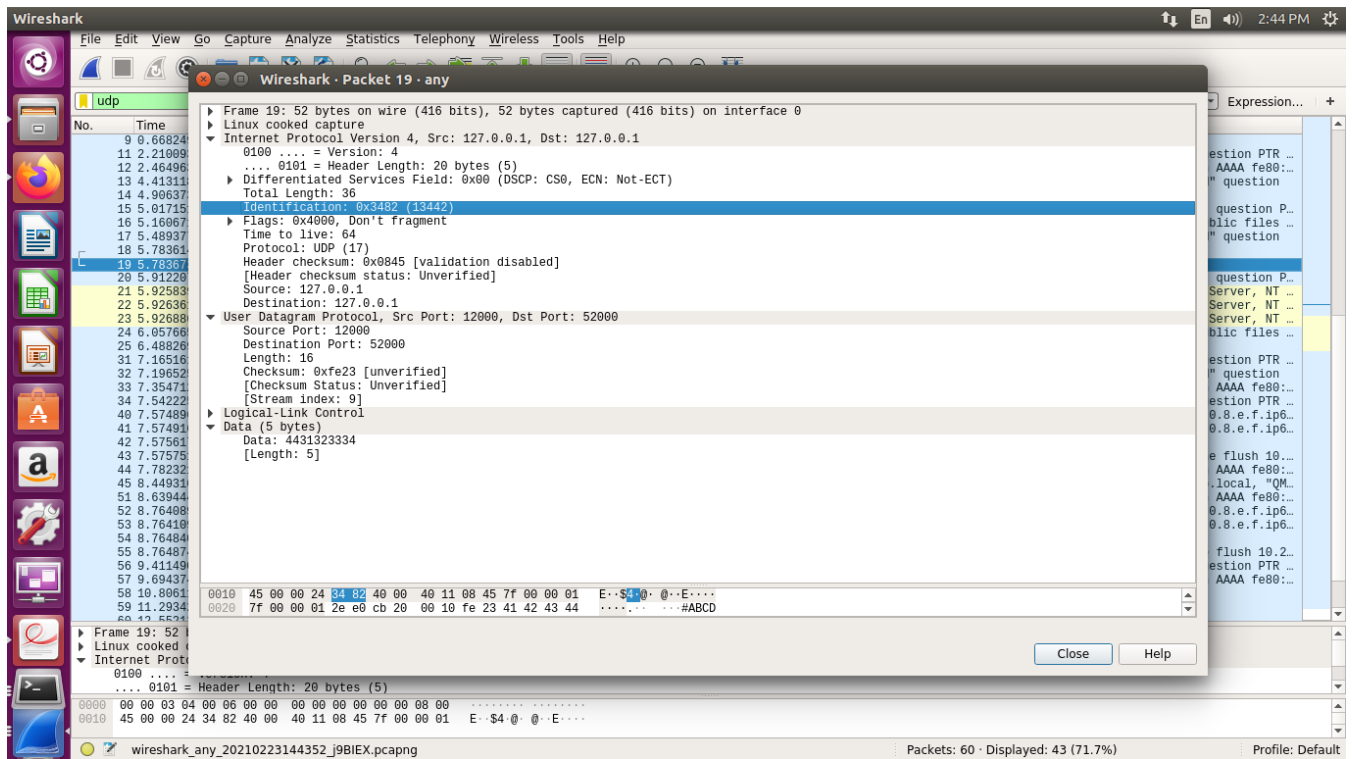
any

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

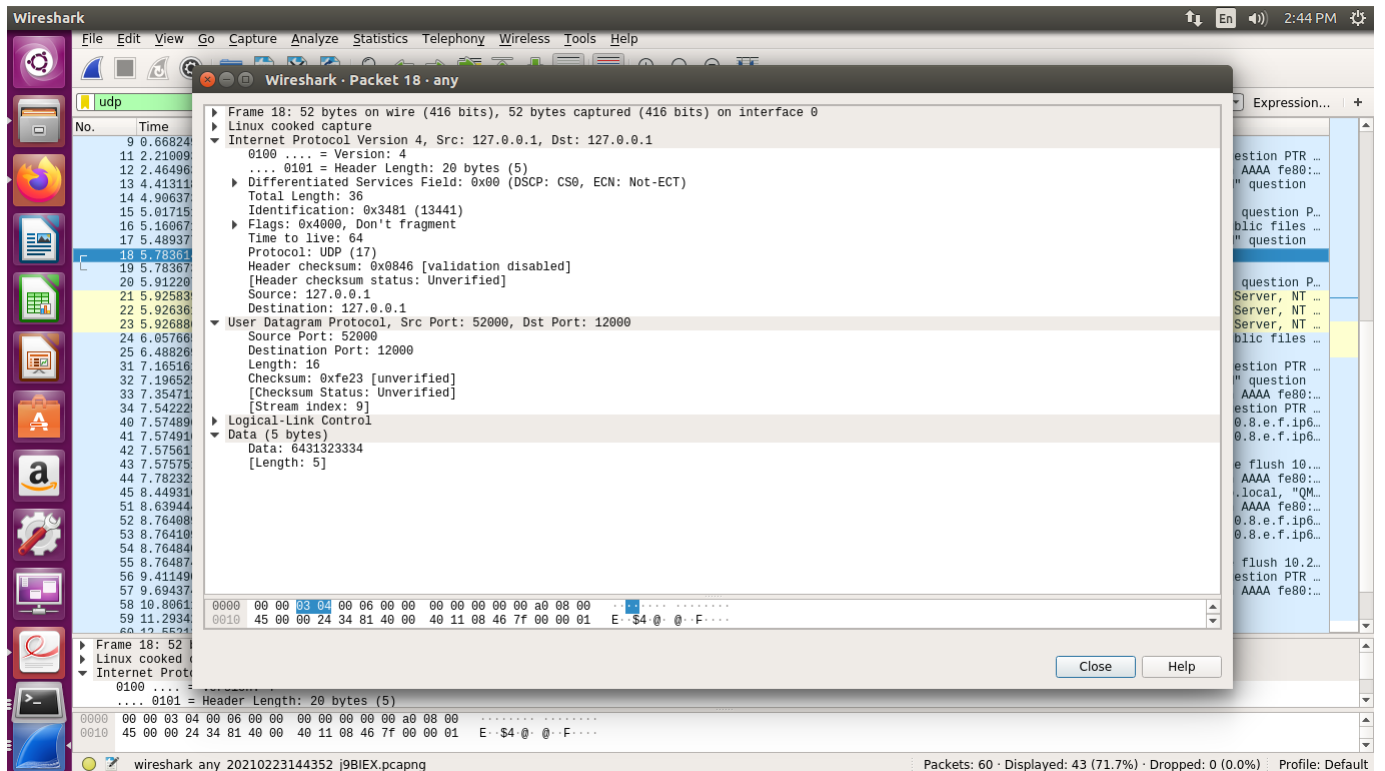
udp Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
61	7.834222057	fe80::f5fb:5a8f:1ae...	ff02::fb	MDNS	384	Standard query 0x0000 TXT UBUNTU18.04_smb_tcp.local, "QM" question PTR_ipp_tcp.local, "QM"
62	7.847283576	fe80::3a71:f863:c93...	ff02::fb	MDNS	127	Standard query 0x0000 A linux.local, "QM" question AAAA linux.local, "QM" question AAAA fe80...
63	7.931415511	10.2.20.131	10.2.20.255	BROWSER	245	Browser Election Request
64	7.931460841	10.2.20.144	10.2.20.255	NBNS	94	Name query NB WORKGROUP-id>
65	7.931658051	10.2.20.180	10.2.20.255	NBNS	94	Name query NB WORKGROUP-id>
66	7.931844838	10.2.20.247	10.2.20.255	NBNS	94	Name query NB WORKGROUP-id>
67	7.978642386	fe80::f5fb:5a8f:1ae...	ff02::fb	MDNS	127	Standard query 0x0000 A linux.local, "QM" question AAAA linux.local, "QM" question AAAA fe80...
68	9.234935864	10.2.20.148	10.2.20.0.251	MDNS	364	Standard query 0x0000 SRV pesu's public files on linux_webdav_tcp.local, "QM" question PTR ...
69	9.386866418	10.2.20.148	224.0.0.251	MDNS	123	Standard query 0x0000 A linux.local, "QM" question AAAA linux.local, "QM" question AAAA fe80...
70	9.933469379	10.2.20.144	10.2.20.255	BROWSER	245	Browser Election Request
71	9.934089845	10.2.20.68	10.2.20.255	BROWSER	238	Browser Election Request
72	9.93497652	10.2.20.247	10.2.20.255	BROWSER	245	Browser Election Request
73	9.934271723	10.2.20.161	10.2.20.255	BROWSER	245	Browser Election Request
74	9.934284868	10.2.20.217	10.2.20.255	BROWSER	245	Browser Election Request
75	9.934285782	10.2.20.18	10.2.20.255	BROWSER	245	Browser Election Request
76	9.934286598	10.2.20.131	10.2.20.255	BROWSER	245	Browser Election Request
77	9.934589142	10.2.20.173	10.2.20.255	BROWSER	245	Browser Election Request
78	9.934520627	10.2.20.194	10.2.20.255	BROWSER	245	Browser Election Request
79	9.974429933	10.2.20.213	224.0.0.251	MDNS	364	Standard query 0x0000 SRV pesu's public files on linux_webdav_tcp.local, "QM" question PTR ...
80	10.069391463	0.0.0.0	255.255.255.255	DHCP	344	DHCP Discover - Transaction ID 0xd3a4a0e
81	10.113714321	10.2.20.251	10.2.20.251	MDNS	123	Standard query 0x0000 A linux.local, "QM" question AAAA linux.local, "QM" question AAAA fe80...
82	11.679610332	127.0.0.1	127.0.0.1	LLC	53	U, func=UA; DSAP 0x00 Group, SSAP 0x62 Command
83	11.679610358	127.0.0.1	127.0.0.1	LLC	53	U, func=UA; DSAP 0x00 Group, SSAP 0x62 Command
84	12.937651834	10.2.20.131	10.2.20.255	BROWSER	245	Browser Election Request
85	14.311104522	fe80::20c6:2aa8:884...	ff02::fb	MDNS	226	Standard query 0x0000 ANY 9.0.6.3.3.4.8.8.8.a.a.2.6.c.0.2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.e.f.ip6...
86	14.311124150	10.2.20.102	224.0.0.251	MDNS	296	Standard query 0x0000 ANY 9.0.6.3.3.4.8.8.8.a.a.2.6.c.0.2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.e.f.ip6...
87	14.311820611	fe80::555:5420:3c95...	ff02::fb	MDNS	128	Standard query response 0x0000 AAAA, cache flush fe80::555:5420:3c95:22be A, cache flush 10.2...
88	14.311875692	10.2.20.183	224.0.0.251	MDNS	124	Standard query response 0x0000 AAAA, cache flush fe80::555:5420:3c95:22be A, cache flush 10.2...
89	14.558536675	fe80::6649:75aa:b37...	ff02::fb	MDNS	384	Standard query 0x0000 TXT pesu's public files on linux_webdav_tcp.local, "QM" question PTR ...
90	14.571222417	fe80::6649:75aa:b37...	ff02::fb	MDNS	127	Standard query 0x0000 A linux.local, "QM" question AAAA linux.local, "QM" question AAAA fe80...
91	14.939670518	10.2.20.131	10.2.20.255	BROWSER	245	Browser Election Request
92	15.074416996	fe80::5e39:8708:e78...	ff02::fb	MDNS	226	Standard query 0x0000 ANY e.4.6.5.6.6.0.5.8.0.7.8.9.3.e.5.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.e.f.ip6...
93	15.074509090	10.2.20.228	224.0.0.251	MDNS	296	Standard query 0x0000 ANY e.4.6.5.6.6.0.5.8.0.7.8.9.3.e.5.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.e.f.ip6...
94	15.074913473	fe80::6823:465a:656...	ff02::fb	MDNS	128	Standard query response 0x0000 AAAA, cache flush fe80::6823:465a:6561:bc6f A, cache flush 10...
95	15.075062913	10.2.20.244	224.0.0.251	MDNS	124	Standard query response 0x0000 AAAA, cache flush fe80::6823:465a:6561:bc6f A, cache flush 10...

Response packet:



Request packet:

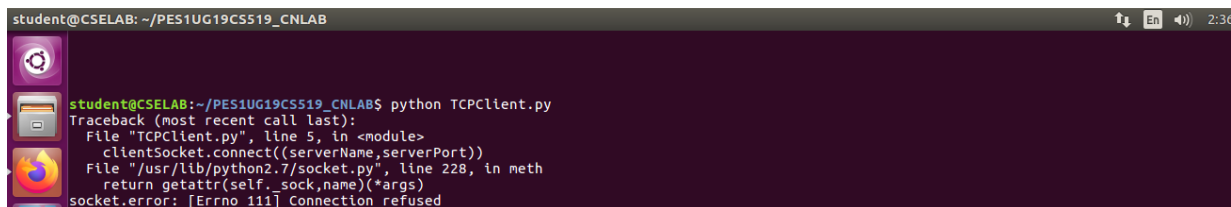


We can see both source and destination being 127.0.0.1(localhost). We can see that the dst port for request is 12000 and src port of response packet is 12000 as set in python code as serverPort. UDP is used and we can also see a data field in the packet inspection.

Problems:

1. Suppose you run TCPClient before you run TCPServer. What happens? Why?

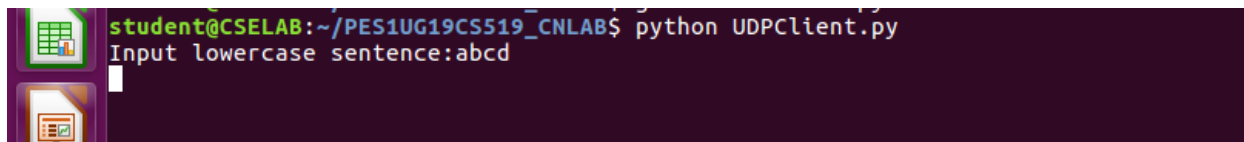
We get a **connection refused error**. A TCP connection can be established only when a host machine listens to a request for a client on a given IP address and port number. Then it accepts connections only at the specified address and port number.



```
student@CSELAB: ~/PES1UG19CS519_CNLAB
student@CSELAB:~/PES1UG19CS519_CNLAB$ python TCPClient.py
Traceback (most recent call last):
  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
```

2. Suppose you run UDPClient before you run UDPServer. What happens? Why?

There is **no error** as UDP is a **connectionless protocol** which is prone to data integrity issues like loss of packets. Hence here the packets are lost forever and we don't get any required response.



```
student@CSELAB:~/PES1UG19CS519_CNLAB$ python UDPClient.py
Input lowercase sentence:abcd
```

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

There is a **connection refused error** for TCP setup. The reason is same as Q1 and similar screenshot to Q1 is seen.

There is **no error** for UDP setup, but we don't get any response as there is no as such server mentioned in client and hence no connection was established and packets are lost forever. The reason is same as Q2 and similar screenshot to Q2 is seen.

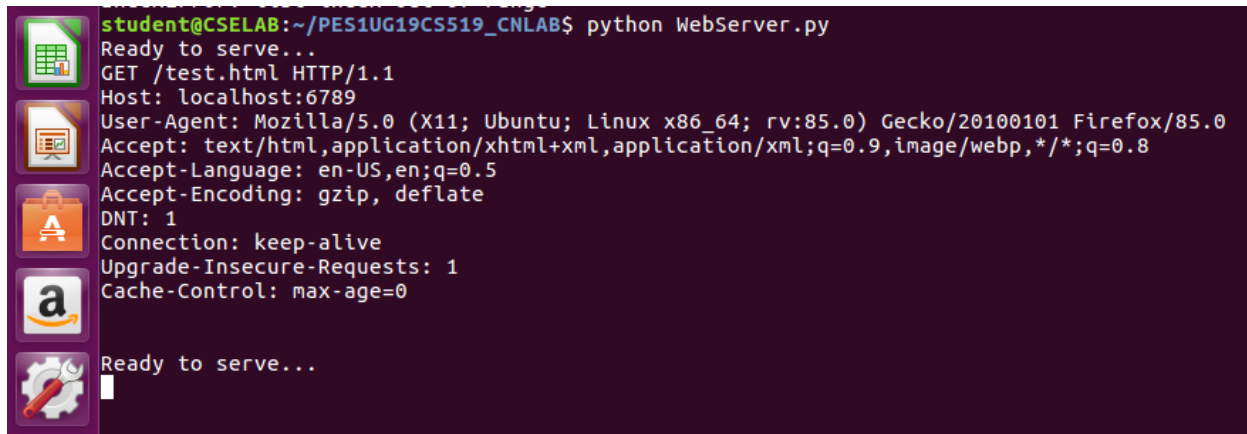
Task2: Web Server

We write a python script to setup a server by creating a **TCP** socket setup using python's socket module. We use **AF_INET** for **IPv4** use, **SOCK_STREAM** for **TCP**. Then we use **localhost(127.0.0.1)** as **server IP** and **6789** as **portnumber** where the server should listen for requests from client. We use **bind()** to put together the port and host/servername and build a socket. Then we continuously listen to request using a **while true loop** which runs infinitely. Then we use a **try except block**. In the try block if we don't get any error try block will be

executed and we extract the path of the file request using **split()** function from the GET header's first line and open the file on the server side and render it with a success response of **"HTTP/1.1 200 OK\r\n\r\n"**. If we get any errors like file not found on the server etc, except block will be executed and we send a **"HTTP/1.1 404 Not Found\r\n\r\n"** response and render a html page with content as

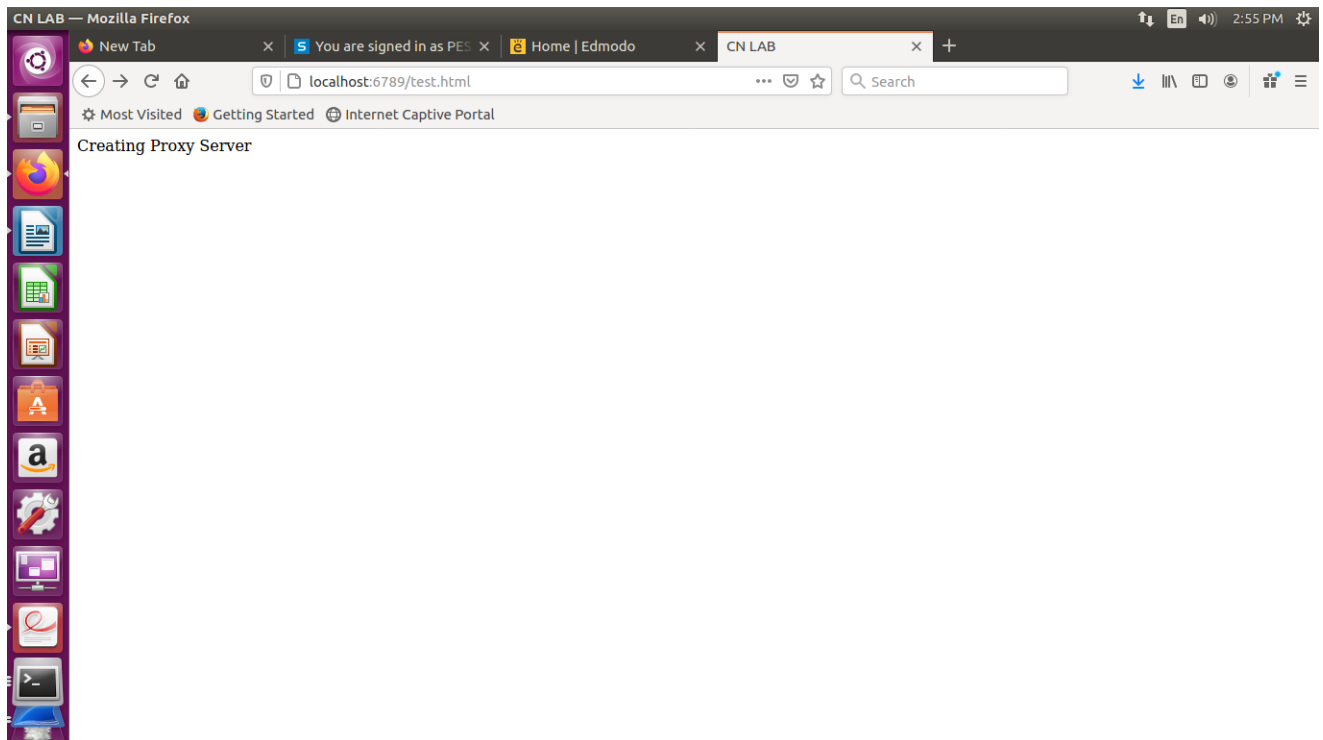
"<html><head></head><body><h1>404 Not Found</h1></body></html>\r\n". Then again the server goes on listening mode until we interrupt.

When we print the request on the terminal we get the following:

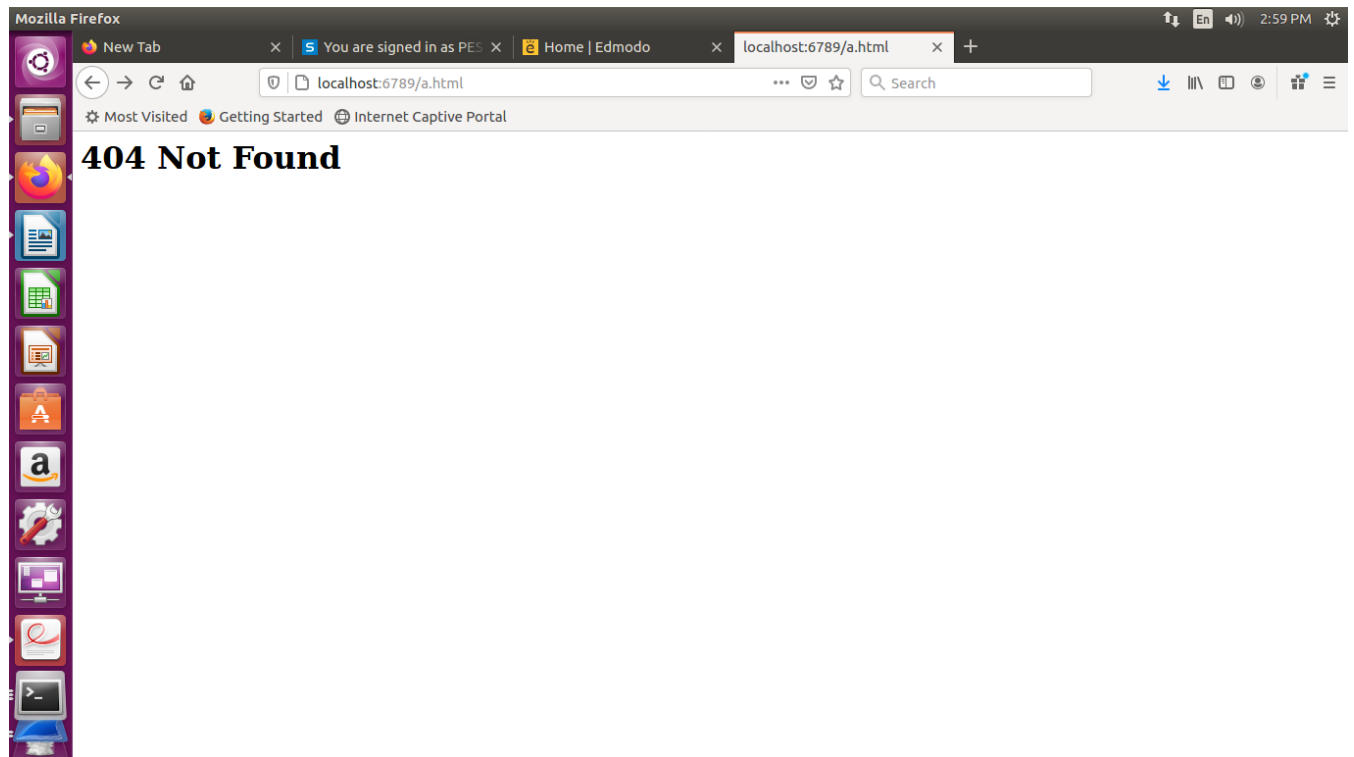


```
student@CSELAB:~/PES1UG19CS519_CNLAB$ python WebServer.py
Ready to serve...
GET /test.html HTTP/1.1
Host: localhost:6789
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:85.0) Gecko/20100101 Firefox/85.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
Ready to serve...
```

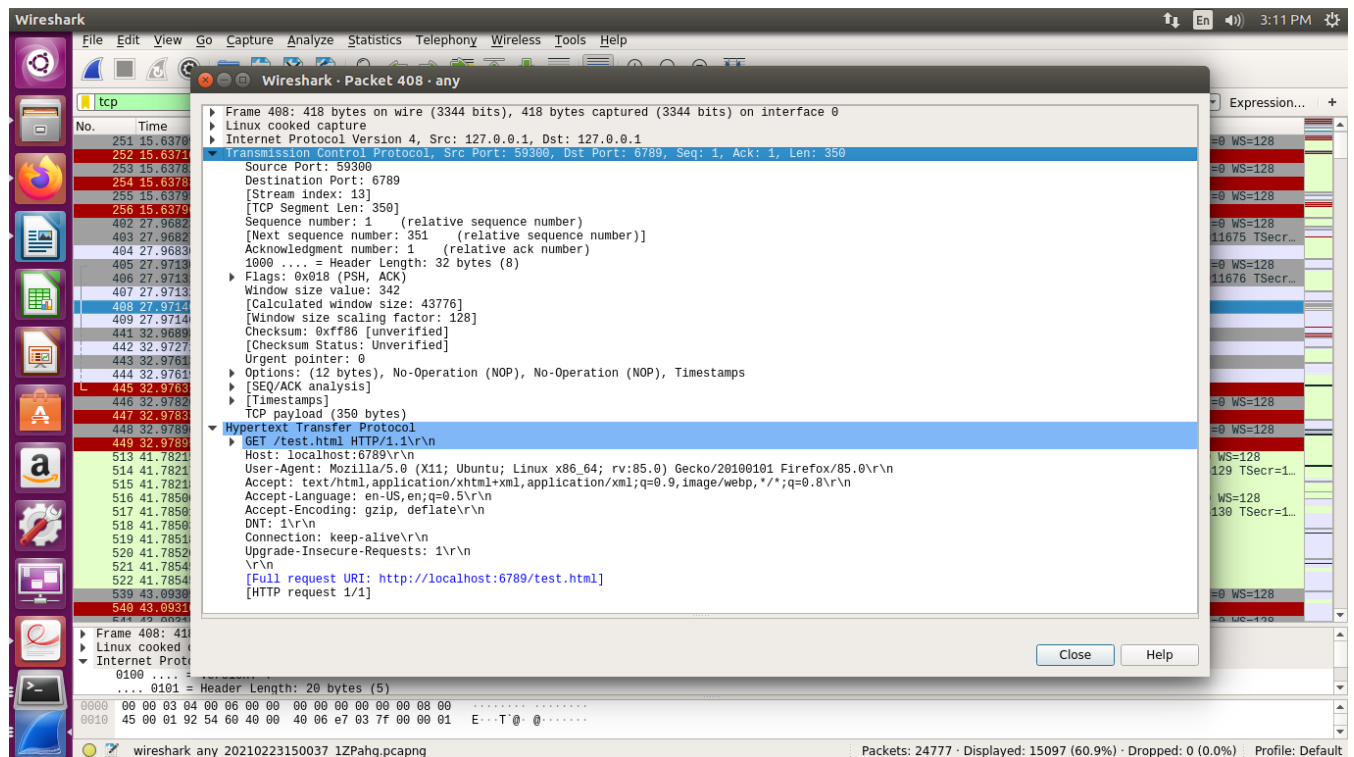
When we request for **test.html** with random content – **Creating Proxy Server** and title **CN Lab** which is present on the server side we get:



When we request a file **a.html** that is not there in the server side:



We capture the following get request packet on successful rendering on wireshark:



We see the **destination port** being **6789**(as it is a request packet) which we have set and it used **TCP** transmission. We see the IP to be **127.0.0.1** which is localhost as set by us. Similarly we get a response packet with **200 OK** response with **6789** as **src port**.

We capture the following response packet on wireshark on unsuccessful request:

```
▼ Transmission Control Protocol, Src Port: 6789, Dst Port: 60258, Seq: 1, Ack: 432, Len: 26
  Source Port: 6789
  Destination Port: 60258
  [Stream index: 40]
  [TCP Segment Len: 26]
  Sequence number: 1 (relative sequence number)
  [Next sequence number: 27 (relative sequence number)]
  Acknowledgment number: 432 (relative ack number)
  1000 .... = Header Length: 32 bytes (8)
  ► Flags: 0x018 (PSH, ACK)
  Window size value: 512
  [Calculated window size: 65536]
  [Window size scaling factor: 128]
  Checksum: 0x3ed8 [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  ► Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ► [SEQ/ACK analysis]
  ► [Timestamps]
  TCP payload (26 bytes)
▼ Hypertext Transfer Protocol
  HTTP/1.1 404 Not Found\r\n
  ► [Expert Info (Chat/Sequence): HTTP/1.1 404 Not Found\r\n]
  Response Version: HTTP/1.1
  Status Code: 404
  [Status Code Description: Not Found]
  Response Phrase: Not Found
\r\n
[HTTP response 1/1]
[Time since request: 0.000087285 seconds]
[Request in frame: 1099]
[Request URI: http://10.2.21.74:6789/hi.html]
```

We see the **source port** being **6789**(as it is a response packet) which we have set and it used **TCP** transmission. We get a **404 not found** response as we requested for a file **./a.html** which isn't present on the server. Similarly we get a request packet with **6789** as **dst port**.