

Computer Networks Lab

Week 5

Abhishek Aditya BS

PES1UG19CS019

Section A

1. Socket Programming

1. Create an application that will
 - Convert lowercase letters to uppercase e.g. [a...z] to [A...Z]
code will not change any special characters, e.g. &*!
 - 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 UDP Connection

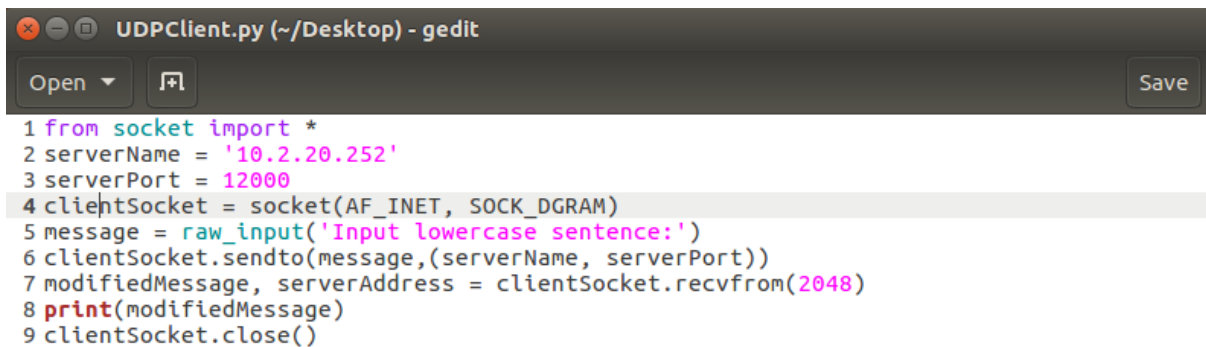
- A UDP connection can be made between two machines with the help of a socket interface using the `socket` library on Python.
- To create a UDP socket interface, the type of socket needs to be set as `SOCK_DGRAM`.
- The type of addresses needs to be set as `AF_INET` which corresponds to IPv4.
- Once the server socket application is created, it needs to be hosted and hence needs to bind to a host IP and port number using the `bind()` 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 UDP Server



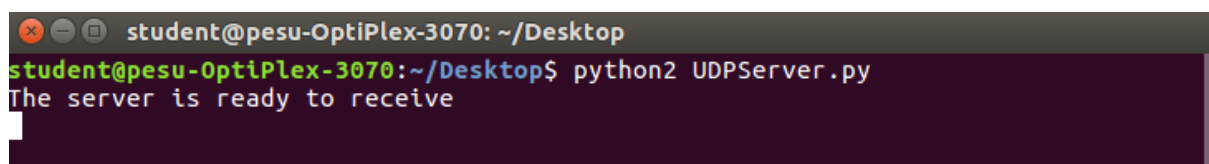
```
UDPServer.py (~/Desktop) - gedit
Open Save
1 from socket import *
2 serverPort = 12000
3 serverSocket = socket(AF_INET, SOCK_DGRAM)
4 serverSocket.bind(('', serverPort))
5 print("The server is ready to receive")
6 while 1:
7     message, clientAddress = serverSocket.recvfrom(2048)
8     modifiedMessage = message.upper()
9     serverSocket.sendto(modifiedMessage, clientAddress)
```

1.1.2 UDP Client



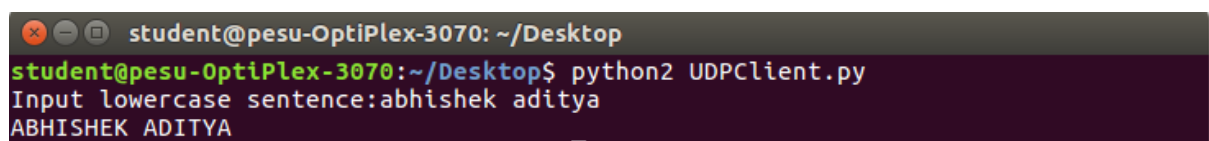
```
UDPClient.py (~/Desktop) - gedit
Open Save
1 from socket import *
2 serverName = '10.2.20.252'
3 serverPort = 12000
4 clientSocket = socket(AF_INET, SOCK_DGRAM)
5 message = raw_input('Input lowercase sentence:')
6 clientSocket.sendto(message, (serverName, serverPort))
7 modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
8 print(modifiedMessage)
9 clientSocket.close()
```

1.1.3 UDP Connection between Server and Client



```
student@pesu-OptiPlex-3070: ~/Desktop
student@pesu-OptiPlex-3070:~/Desktop$ python2 UDPServer.py
The server is ready to receive
```

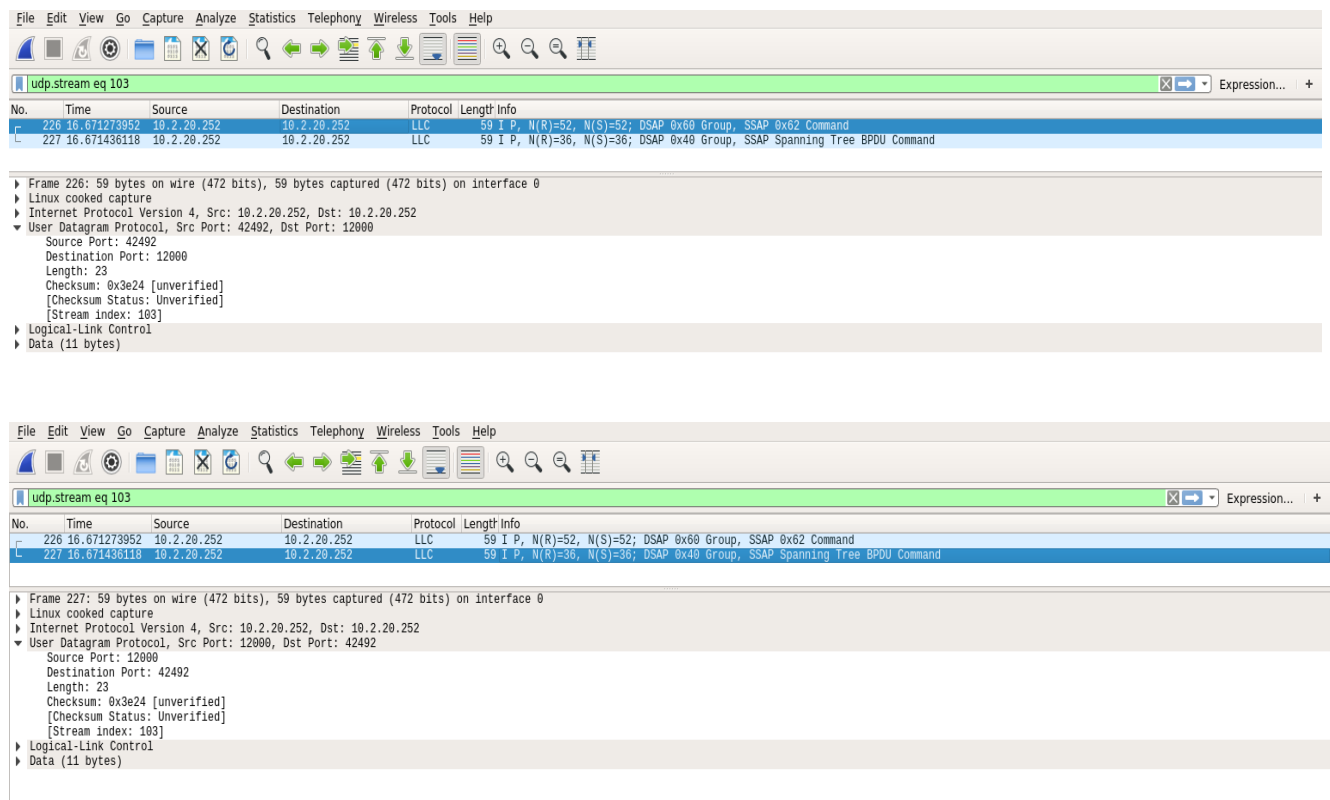
UDP Server



```
student@pesu-OptiPlex-3070: ~/Desktop
student@pesu-OptiPlex-3070:~/Desktop$ python2 UDPClient.py
Input lowercase sentence:abhishek aditya
ABHISHEK ADITYA
```

UDP Client

1.1.4 Wireshark Packet Capture for UDP Connection



1.2 TCP Connection

- A TCP connection can be made between two machines with the help of a socket interface using the socket library on Python.
- 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 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 the `bind()` 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 TCP Server

```
TCPServer.py (~/Desktop/PES1UG19CS019/TCP) - gedit
Open Save
1 from socket import *
2 serverPort = 12000
3 serverSocket = socket(AF_INET, SOCK_STREAM)
4 serverSocket.bind(('', serverPort))
5 serverSocket.listen(1)
6 print('The server is ready to receive')
7 while 1:
8     connectionSocket, addr = serverSocket.accept()
9     sentence = connectionSocket.recv(1024)
10    capitalizedSentence = sentence.upper()
11    connectionSocket.send(capitalizedSentence)
12    connectionSocket.close()
```

1.2.2 TCP Client

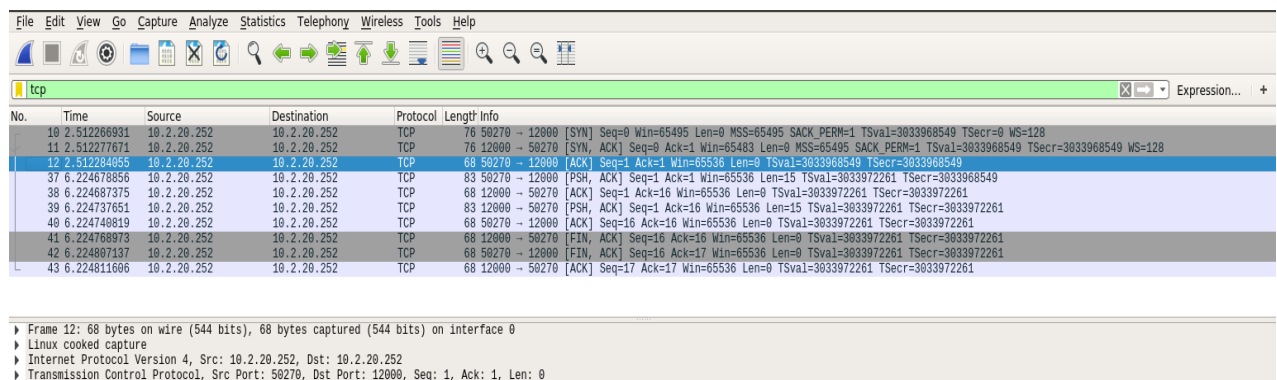
```
TCPClient.py (~/Desktop/PES1UG19CS019/TCP) - gedit
Open Save
1 from socket import *
2 serverName = '10.2.20.252'
3 serverPort = 12000
4 clientSocket = socket(AF_INET, SOCK_STREAM)
5 clientSocket.connect((serverName, serverPort))
6 sentence = raw_input('Input lowercase sentence:')
7 clientSocket.send(sentence)
8 modifiedSentence = clientSocket.recv(1024)
9 print 'From Server:', modifiedSentence
10 clientSocket.close()
```

1.2.3 TCP Connection between Server and Client

```
student@pesu-OptiPlex-3070: ~/Desktop
student@pesu-OptiPlex-3070:~/Desktop$ python2 TCPServer.py
The server is ready to receive

student@pesu-OptiPlex-3070: ~/Desktop
student@pesu-OptiPlex-3070:~/Desktop$ python2 TCPClient.py
Input lowercase sentence:abhishek aditya
From Server: ABHISHEK ADITYA
student@pesu-OptiPlex-3070:~/Desktop$
```

1.2.4 Wireshark Packet Capture for TCP Connection



No.	Time	Source	Destination	Protocol	Length	Info
10	2.512266931	10.2.20.252	10.2.20.252	TCP	76	50270 → 12000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3033968549 TSecr=0 WS=128
11	2.512277671	10.2.20.252	10.2.20.252	TCP	76	12000 → 50270 [SYN, ACK] Seq=0 Ack=1 Win=65493 Len=0 MSS=65495 SACK_PERM=1 TSval=3033968549 TSecr=3033968549 WS=128
12	2.512284055	10.2.20.252	10.2.20.252	TCP	68	50270 → 12000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3033968549 TSecr=3033968549
37	6.224678856	10.2.20.252	10.2.20.252	TCP	83	50270 → 12000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=15 TSval=3033972261 TSecr=3033968549
38	6.224687375	10.2.20.252	10.2.20.252	TCP	68	12000 → 50270 [ACK] Seq=1 Ack=16 Win=65536 Len=0 TSval=3033972261 TSecr=3033972261
39	6.224737651	10.2.20.252	10.2.20.252	TCP	83	12000 → 50270 [PSH, ACK] Seq=1 Ack=16 Win=65536 Len=15 TSval=3033972261 TSecr=3033972261
40	6.224740819	10.2.20.252	10.2.20.252	TCP	68	50270 → 12000 [ACK] Seq=16 Ack=16 Win=65536 Len=0 TSval=3033972261 TSecr=3033972261
41	6.224768973	10.2.20.252	10.2.20.252	TCP	68	12000 → 50270 [FIN, ACK] Seq=16 Ack=16 Win=65536 Len=0 TSval=3033972261 TSecr=3033972261
42	6.224807137	10.2.20.252	10.2.20.252	TCP	68	50270 → 12000 [FIN, ACK] Seq=16 Ack=17 Win=65536 Len=0 TSval=3033972261 TSecr=3033972261
43	6.224811606	10.2.20.252	10.2.20.252	TCP	68	12000 → 50270 [ACK] Seq=17 Ack=17 Win=65536 Len=0 TSval=3033972261 TSecr=3033972261

Frame 12: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.2.20.252, Dst: 10.2.20.252
Transmission Control Protocol, Src Port: 50270, Dst Port: 12000, Seq: 1, Ack: 1, Len: 0

1.3 Questions

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 set up 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

A simple Web server in Python that is capable of processing only one request

```
# 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 = 6789

# 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 content 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")

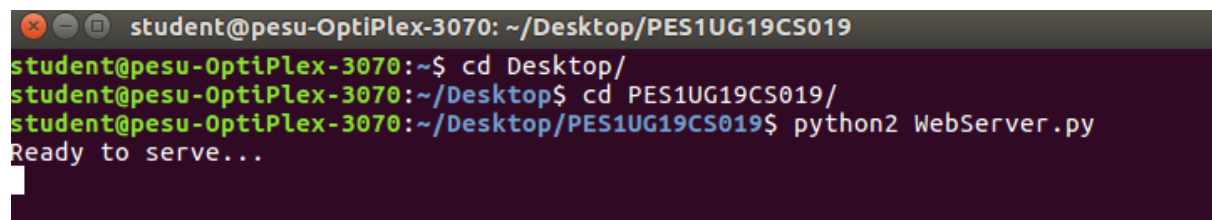
# Close the client connection socket
connectionSocket.close()

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><body><h1>404 Not Found</h1></body></html>\r\n")
    # Close the client connection socket
    connectionSocket.close()

serverSocket.close()

```

2.1 Running the Web Server

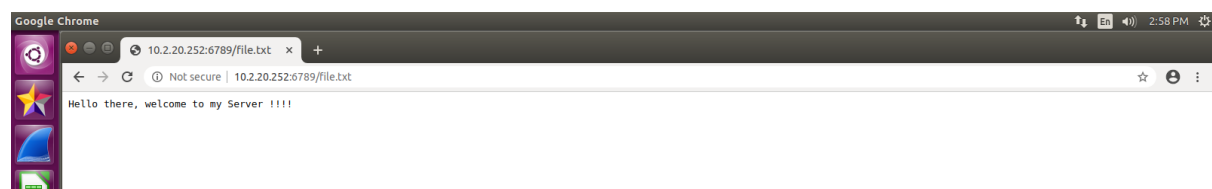


```

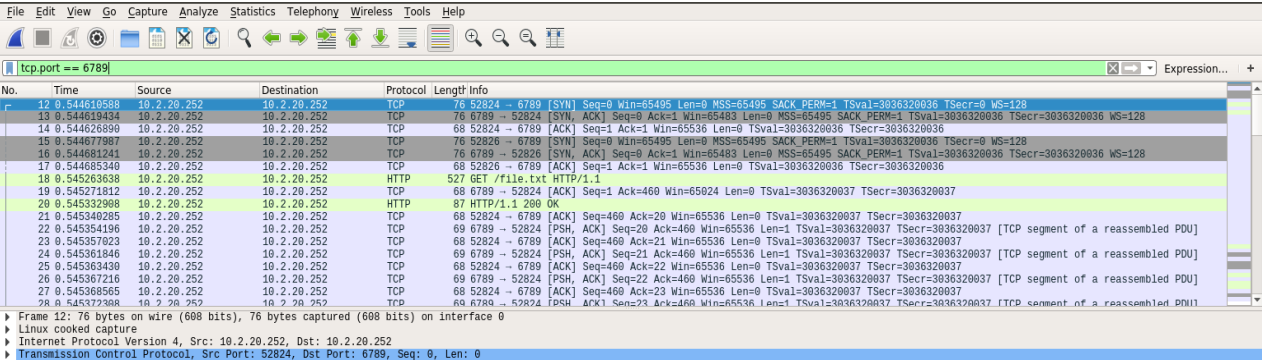
student@pesu-OptiPlex-3070: ~/Desktop/PES1UG19CS019
student@pesu-OptiPlex-3070:~$ cd Desktop/
student@pesu-OptiPlex-3070:~/Desktop$ cd PES1UG19CS019/
student@pesu-OptiPlex-3070:~/Desktop/PES1UG19CS019$ python2 WebServer.py
Ready to serve...

```

2.2 Accessing the file from Server's file system



2.3 Wireshark Capture



Wireshark packet capture showing a TCP stream to port 6789. The capture is filtered on 'tcp.port == 6789'. The table below shows the captured packets, highlighting the HTTP GET request for /file.txt.

No.	Time	Source	Destination	Protocol	Length	Info
12	0.544610588	10.2.20.252	10.2.20.252	TCP	76	52824 → 6789 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3036320026 TSecr=0 WS=128
13	0.544619434	10.2.20.252	10.2.20.252	TCP	76	6789 → 52824 [SYN, ACK] Seq=0 Ack=1 Win=65493 Len=0 MSS=65495 SACK_PERM=1 TSval=3036320036 TSecr=3036320036 WS=128
14	0.544626890	10.2.20.252	10.2.20.252	TCP	68	52824 → 6789 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3036320036 TSecr=3036320036
15	0.544677987	10.2.20.252	10.2.20.252	TCP	76	52826 → 6789 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3036320036 TSecr=0 WS=128
16	0.544681241	10.2.20.252	10.2.20.252	TCP	76	6789 → 52826 [SYN, ACK] Seq=0 Ack=1 Win=65493 Len=0 MSS=65495 SACK_PERM=1 TSval=3036320036 TSecr=3036320036 WS=128
17	0.544685340	10.2.20.252	10.2.20.252	TCP	68	52826 → 6789 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3036320036 TSecr=3036320036
18	0.545263638	10.2.20.252	10.2.20.252	HTTP	527	GET /file.txt HTTP/1.1
19	0.545271812	10.2.20.252	10.2.20.252	TCP	68	6789 → 52824 [ACK] Seq=1 Ack=460 Win=65024 Len=0 TSval=3036320037 TSecr=3036320037
20	0.545332908	10.2.20.252	10.2.20.252	HTTP	87	HTTP/1.1 200 OK
21	0.545340285	10.2.20.252	10.2.20.252	TCP	68	52824 → 6789 [ACK] Seq=460 Ack=20 Win=65536 Len=0 TSval=3036320037 TSecr=3036320037
22	0.545354196	10.2.20.252	10.2.20.252	TCP	69	6789 → 52824 [PSH, ACK] Seq=20 Ack=460 Win=65536 Len=1 TSval=3036320037 TSecr=3036320037 [TCP segment of a reassembled PDU]
23	0.545357023	10.2.20.252	10.2.20.252	TCP	68	52824 → 6789 [ACK] Seq=460 Ack=21 Win=65536 Len=0 TSval=3036320037 TSecr=3036320037
24	0.545361846	10.2.20.252	10.2.20.252	TCP	69	6789 → 52824 [PSH, ACK] Seq=21 Ack=460 Win=65536 Len=1 TSval=3036320037 TSecr=3036320037 [TCP segment of a reassembled PDU]
25	0.545363430	10.2.20.252	10.2.20.252	TCP	68	52824 → 6789 [ACK] Seq=460 Ack=22 Win=65536 Len=0 TSval=3036320037 TSecr=3036320037
26	0.545367216	10.2.20.252	10.2.20.252	TCP	69	6789 → 52824 [PSH, ACK] Seq=22 Ack=460 Win=65536 Len=1 TSval=3036320037 TSecr=3036320037 [TCP segment of a reassembled PDU]
27	0.545368565	10.2.20.252	10.2.20.252	TCP	68	52824 → 6789 [ACK] Seq=460 Ack=23 Win=65536 Len=0 TSval=3036320037 TSecr=3036320037
28	0.545377388	10.2.20.252	10.2.20.252	TCP	69	6789 → 52824 [PSH, ACK] Seq=23 Ack=460 Win=65536 Len=1 TSval=3036320037 TSecr=3036320037 [TCP segment of a reassembled PDU]

Frame 12: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.2.20.252, Dst: 10.2.20.252
Transmission Control Protocol, Src Port: 52824, Dst Port: 6789, Seq: 0, Len: 0

2.4 TCP Stream of the request to the server



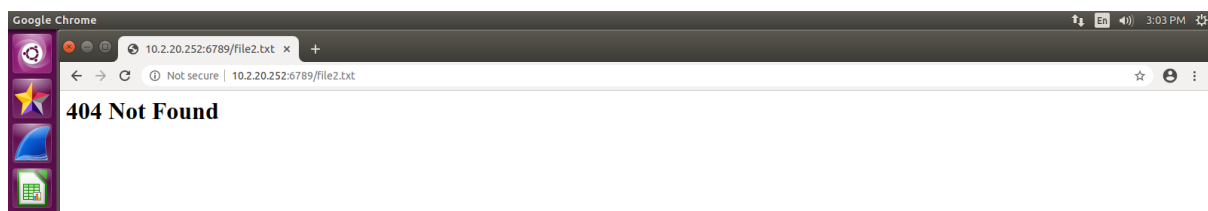
Wireshark Follow TCP Stream (tcp.stream eq 0) - any

```
GET /file.txt HTTP/1.1
Host: 10.2.20.252:6789
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.88 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8

HTTP/1.1 200 OK

Hello there, welcome to my Server !!!!
```

2.5 Requesting a file not Present in the Server



Server returns a “404 Not Found” error message