CN Lab Report – Week 5

PES1201800366

Aditeya Baral

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

1.1.1 TCP Server

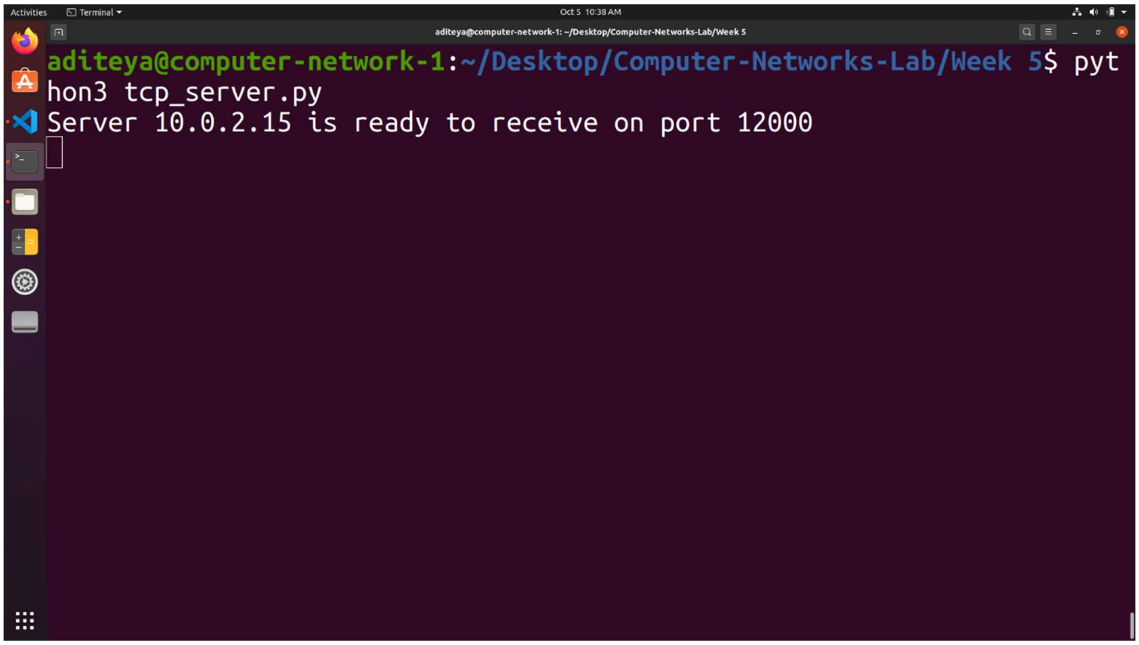
```
from socket import socket, AF_INET, SOCK_STREAM

server_name = "10.0.2.15"
server_port = 12000
server_socket = socket(AF_INET, SOCK_STREAM)
server_socket.bind((server_name, server_port))
server_socket.listen(1)

print(f"Server 10.0.2.15 is ready to receive on port {server_port}")
while True:
    connection_socket, address = server_socket.accept()
    sentence = connection_socket.recv(1024)
    sentence = sentence.upper()
    connection_socket.send(sentence)
    connection_socket.close()
```

1.1.2 TCP Client

```
import sys
from socket import socket, AF_INET, SOCK_STREAM

server_name = sys.argv[1].encode()
server_port = int(sys.argv[2])
client_socket = socket(AF_INET, SOCK_STREAM)
client_socket.connect((server_name, server_port))

sentence = input("\nEnter sentence: ").encode()
client_socket.send(sentence)
```

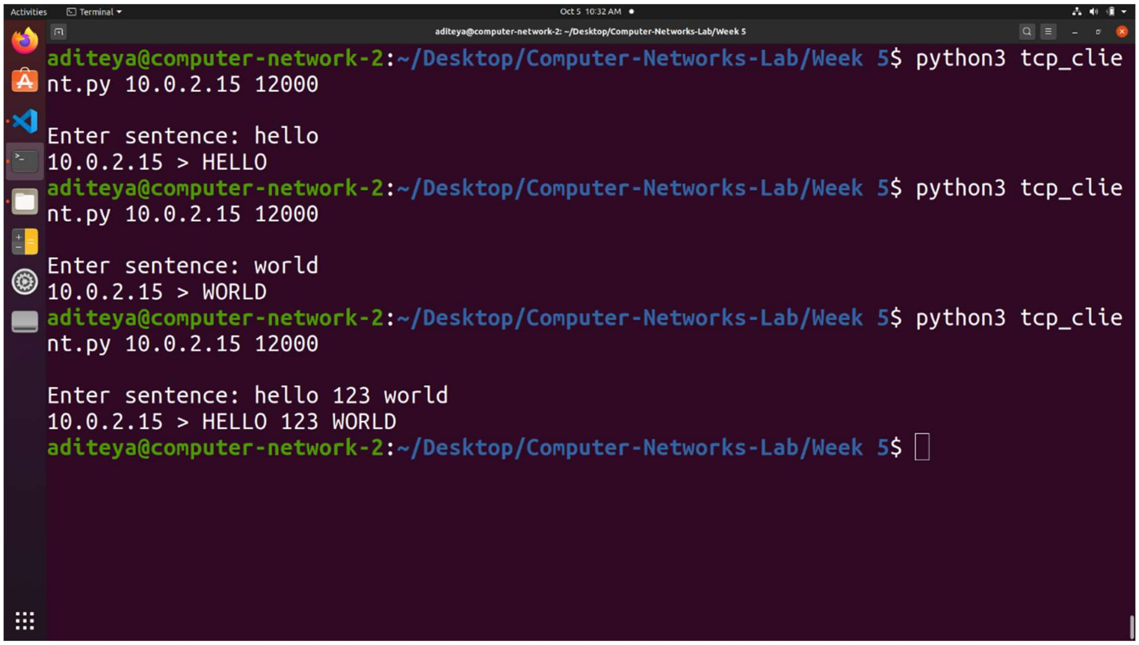
```
modified_sentence = client_socket.recv(1024)
print(f"{server_name.decode()} > {modified_sentence.decode()}")
client_socket.close()
```

1.1.3 TCP Connection between Server and Client


Activities
Terminal
Oct 5 10:38 AM
aditeya@computer-network-1: ~/Desktop/Computer-Networks-Lab/Week 5

```
aditeya@computer-network-1:~/Desktop/Computer-Networks-Lab/Week 5$ pyt
hon3 tcp_server.py
Server 10.0.2.15 is ready to receive on port 12000
```

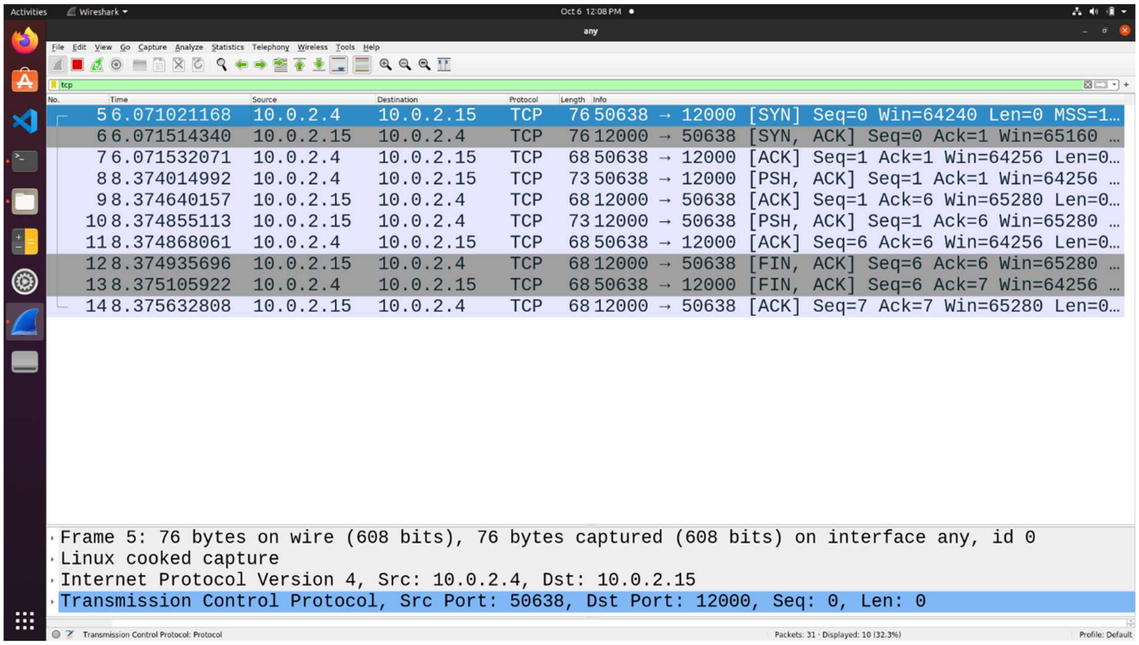
TCP Server


Activities
Terminal
Oct 5 10:32 AM
aditeya@computer-network-2: ~/Desktop/Computer-Networks-Lab/Week 5

```
aditeya@computer-network-2:~/Desktop/Computer-Networks-Lab/Week 5$ python3 tcp_clie
nt.py 10.0.2.15 12000

Enter sentence: hello
10.0.2.15 > HELLO
aditeya@computer-network-2:~/Desktop/Computer-Networks-Lab/Week 5$ python3 tcp_clie
nt.py 10.0.2.15 12000

Enter sentence: world
10.0.2.15 > WORLD
aditeya@computer-network-2:~/Desktop/Computer-Networks-Lab/Week 5$ python3 tcp_clie
nt.py 10.0.2.15 12000

Enter sentence: hello 123 world
10.0.2.15 > HELLO 123 WORLD
aditeya@computer-network-2:~/Desktop/Computer-Networks-Lab/Week 5$
```

TCP Client

1.1.4 Wireshark Capture for TCP Connection


Activities
Wireshark
Oct 6 12:08 PM
any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp

```
No.  Time         Source     Destination Protocol Length Info
   5 6.071021168  10.0.2.4   10.0.2.15   TCP      76 50638 → 12000 [SYN] Seq=0 Win=64240 Len=0 MSS=1…
   6 6.071514340  10.0.2.15  10.0.2.4    TCP      76 12000 → 50638 [SYN, ACK] Seq=0 Ack=1 Win=65160 …
   7 6.071532071  10.0.2.4   10.0.2.15   TCP      68 50638 → 12000 [ACK] Seq=1 Ack=1 Win=64256 Len=0…
   8 8.374014992  10.0.2.4   10.0.2.15   TCP      73 50638 → 12000 [PSH, ACK] Seq=1 Ack=1 Win=64256 …
   9 8.374640157  10.0.2.15  10.0.2.4    TCP      68 12000 → 50638 [ACK] Seq=1 Ack=6 Win=65280 Len=0…
  10 8.374855113  10.0.2.15  10.0.2.4    TCP      73 12000 → 50638 [PSH, ACK] Seq=1 Ack=6 Win=65280 …
  11 8.374868061  10.0.2.4   10.0.2.15   TCP      68 50638 → 12000 [ACK] Seq=6 Ack=6 Win=64256 Len=0…
  12 8.374935696  10.0.2.15  10.0.2.4    TCP      68 12000 → 50638 [FIN, ACK] Seq=6 Ack=6 Win=65280 …
  13 8.375105922  10.0.2.4   10.0.2.15   TCP      68 50638 → 12000 [FIN, ACK] Seq=6 Ack=7 Win=64256 …
  14 8.375632808  10.0.2.15  10.0.2.4    TCP      68 12000 → 50638 [ACK] Seq=7 Ack=7 Win=65280 Len=0…
```

```
Frame 5: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface any, id 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.15
Transmission Control Protocol, Src Port: 50638, Dst Port: 12000, Seq: 0, Len: 0
```

Transmission Control Protocol: Protocol   Packets: 31 · Displayed: 10 (32.3%)   Profile: Default

1.2 UDP Connection

1.2.1 UDP Server

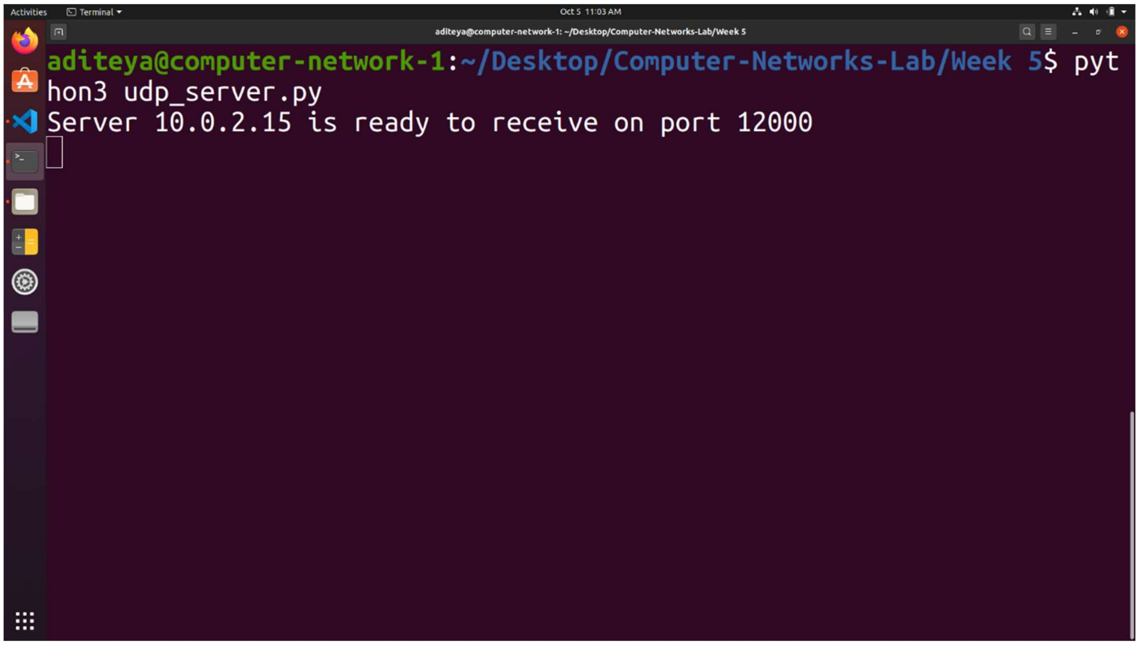
```python
import sys
from socket import socket, AF_INET, SOCK_DGRAM

server_name = "10.0.2.15"
server_port = 12000
server_socket = socket(AF_INET, SOCK_DGRAM)
server_socket.bind((server_name, server_port))

print(f"Server 10.0.2.15 is ready to receive on port {server_port}")

while True:
    message, client_address = server_socket.recvfrom(2048)
    message = message.upper()
    server_socket.sendto(message, client_address)
```

1.2.2 UDP Client

```python
import sys
from socket import socket, AF_INET, SOCK_STREAM

server_name = sys.argv[1].encode()
server_port = int(sys.argv[2])
client_socket = socket(AF_INET, SOCK_STREAM)
client_socket.connect((server_name, server_port))

sentence = input("\nEnter sentence: ").encode()
client_socket.send(sentence)
```

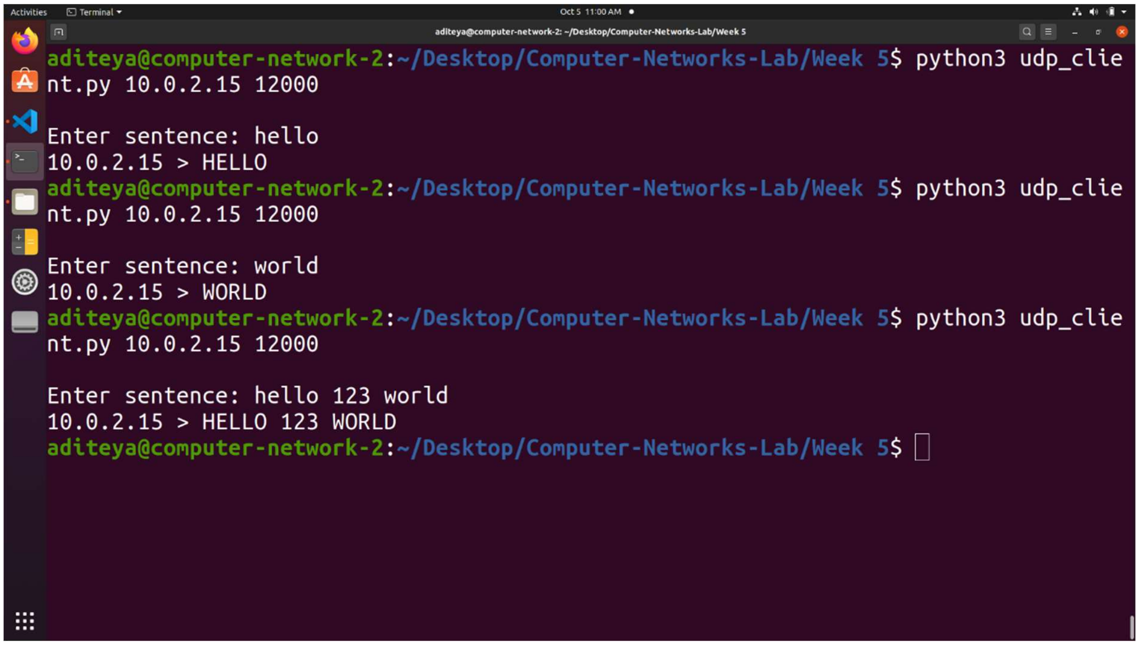
```
modified_sentence = client_socket.recv(1024)
print(f"{server_name.decode()} > {modified_sentence.decode()}")
client_socket.close()
```

1.2.3 UDP Connection between Server and Client


Activities
Terminal
Oct 5 11:03 AM
aditeya@computer-network-1: ~/Desktop/Computer-Networks-Lab/Week 5

```
aditeya@computer-network-1:~/Desktop/Computer-Networks-Lab/Week 5$ pyt
hon3 udp_server.py
Server 10.0.2.15 is ready to receive on port 12000
```

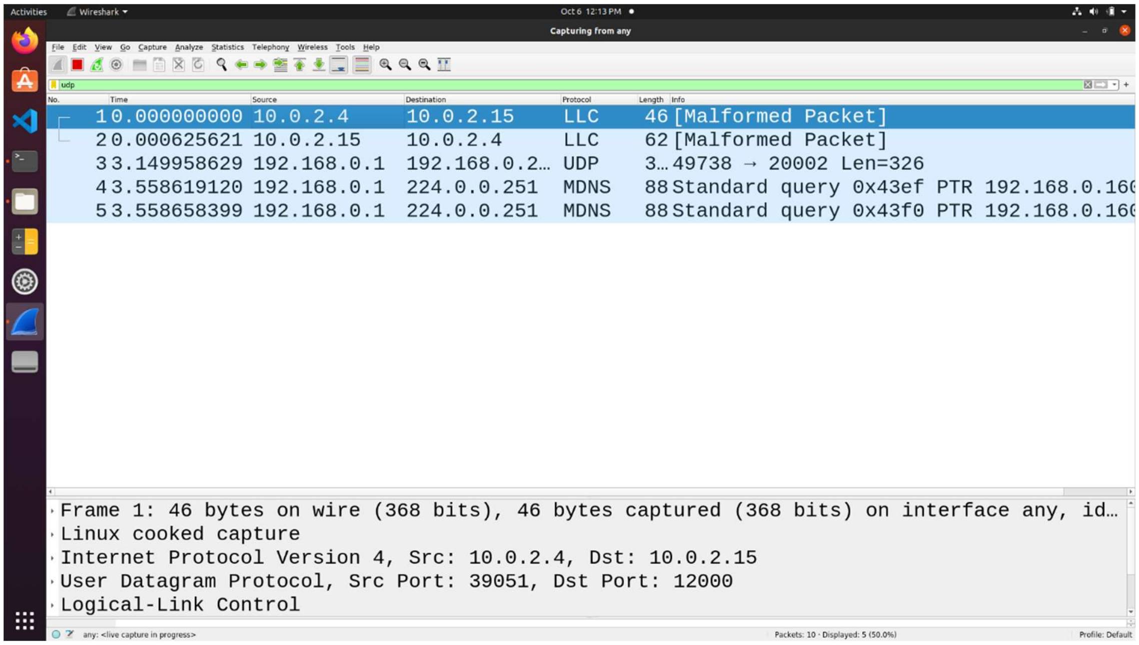
UDP Server


Activities
Terminal
Oct 5 11:00 AM
aditeya@computer-network-2: ~/Desktop/Computer-Networks-Lab/Week 5

```
aditeya@computer-network-2:~/Desktop/Computer-Networks-Lab/Week 5$ python3 udp_clie
nt.py 10.0.2.15 12000

Enter sentence: hello
10.0.2.15 > HELLO
aditeya@computer-network-2:~/Desktop/Computer-Networks-Lab/Week 5$ python3 udp_clie
nt.py 10.0.2.15 12000

Enter sentence: world
10.0.2.15 > WORLD
aditeya@computer-network-2:~/Desktop/Computer-Networks-Lab/Week 5$ python3 udp_clie
nt.py 10.0.2.15 12000

Enter sentence: hello 123 world
10.0.2.15 > HELLO 123 WORLD
aditeya@computer-network-2:~/Desktop/Computer-Networks-Lab/Week 5$
```

UDP Client

1.2.4 Wireshark Capture for UDP Connection


Activities
Wireshark
Oct 6 12:13 PM
Capturing from any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
udp
any: <live capture in progress>
Packets: 10 · Displayed: 5 (50.0%)
Profile: Default

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.4	10.0.2.15	LLC	46	[Malformed Packet]
2	0.000625621	10.0.2.15	10.0.2.4	LLC	62	[Malformed Packet]
3	3.149958629	192.168.0.1	192.168.0.2…	UDP	3…	49738 → 20002 Len=326
4	3.558619120	192.168.0.1	224.0.0.251	MDNS	88	Standard query 0x43ef PTR 192.168.0.16
5	3.558658399	192.168.0.1	224.0.0.251	MDNS	88	Standard query 0x43f0 PTR 192.168.0.16

```
Frame 1: 46 bytes on wire (368 bits), 46 bytes captured (368 bits) on interface any, id…
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.15
User Datagram Protocol, Src Port: 39051, Dst Port: 12000
Logical-Link Control
```

2. Task 3 – Multi Threaded Web Proxy

In this assignment, you will develop a Web proxy. When your proxy receives an HTTP request for an object from a browser, it generates a new HTTP request for the same object and sends it to the origin server. When the proxy receives the corresponding HTTP response with the object from the origin server, it creates a new HTTP response, including the object, and sends it to the client. This proxy will be multi-threaded, so that it will be able to handle multiple requests at the same time.

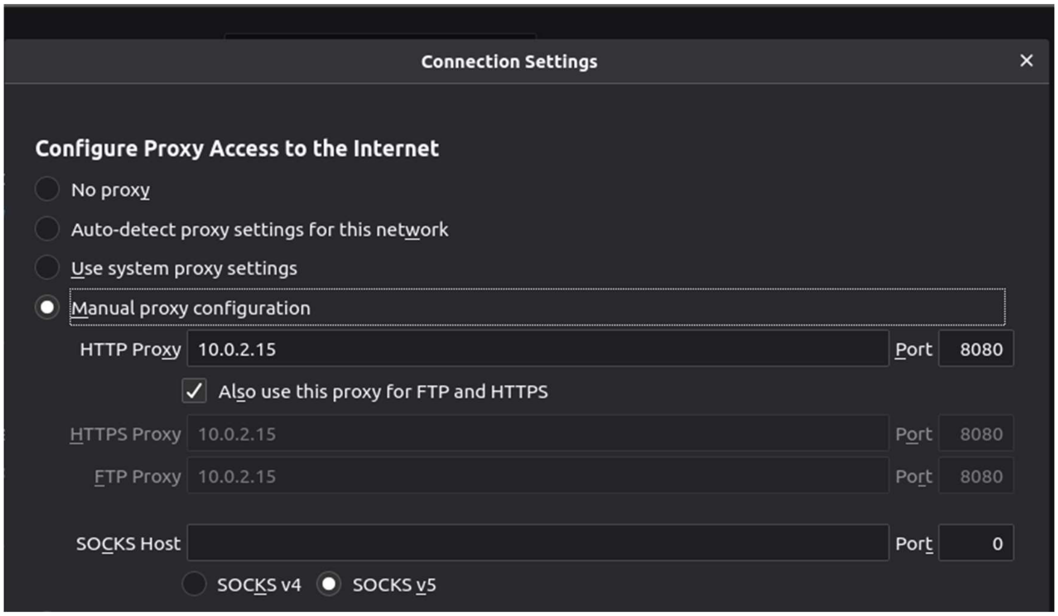
2.1 Setting up a Web Proxy Server

- We first set up a web proxy server using Python3 which is capable of handling multiple requests at the same time.
- This is done with the help of the **`socket`** and **`threading`** libraries which are built in libraries included with the language.
- The **`socket`** library is used to create a connection between the proxy server and client machine.
- The **`threading`** library is used to spawn a new thread for every connection made between the client machine and the proxy server. This new thread is used to generate a HTTP request to the destination server and receive the corresponding HTTP response from the server machine.
- This entire process is run iteratively in an endless **`while`** loop so that it can handle multiple requests at the same time.

```
import os
import sys
import threading
from socket import socket, AF_INET, SOCK_STREAM, error

NUM_REQS = 50
BUF_SIZE = 999999


def proxy_server_thread(client_conn, client_addr):
    request = client_conn.recv(BUF_SIZE)
    request_first_line = request.decode().split("\n")[0]
    url = request_first_line.split(" ")[1]
    print("From", "\t", client_addr[0], "\t", "Request", "\t",
request_first_line)

    http_pos = url.find("://")
    if http_pos == -1:
        temp = url
    else:
        temp = url[(http_pos + 3) :]

    port_pos = temp.find(":")

    webserver_pos = temp.find("/")
    if webserver_pos == -1:
        webserver_pos = len(temp)

    webserver = ""
    port = -1
    if port_pos == -1 or webserver_pos < port_pos:
        port = 80
        webserver = temp[:webserver_pos]
    else:
        port = int((temp[(port_pos + 1) :])[: webserver_pos - port_pos -
1])
        webserver = temp[:port_pos]

    try:
        s = socket(AF_INET, SOCK_STREAM)
        s.connect((webserver, port))
        s.send(request)
        while 1:
            response = s.recv(BUF_SIZE)
            response_first_line = response.decode("utf8",
"ignore").partition("\n")[0]
            print(
                "To", "\t", client_addr[0], "\t", "Response", "\t",
response_first_line
            )
            if len(response) > 0:
                client_conn.send(response)
            else:
                break
        s.close()
        client_conn.close()
    except error:
        if s:
            s.close()
        if client_conn:
            client_conn.close()
        print(client_addr[0], "\t", "Peer reset", "\t", request_first_line)
        sys.exit(1)


def proxy_server():
    if len(sys.argv) < 2:
        print("Using Default port 8080 since no port was mentioned.")
        port = 8080
    else:
        port = int(sys.argv[1])
    host = ""
    print("Proxy server Running on localhost :", port)
    try:
        s = socket(AF_INET, SOCK_STREAM)
        s.bind((host, port))
        s.listen(NUM_REQS)
    except error:
        if s:
            s.close()
        print("Could not open socket:")
        sys.exit(1)
    while 1:
        client_conn, client_addr = s.accept()
        threading._start_new_thread(proxy_server_thread, (client_conn,
client_addr))
    s.close()


if __name__ == "__main__":
    proxy_server()
```

2.2 Configuring Browser

- The browser needs to be configured to use the socket application as a multi-threaded web proxy.
- This is done by adding the IP address of the host machine and the port number at which the socket application is being hosted on.



Connection Settings

Configure Proxy Access to the Internet

○ No proxy

○ Auto-detect proxy settings for this network

○ Use system proxy settings

● Manual proxy configuration

HTTP Proxy 10.0.2.15 Port 8080

☑ Also use this proxy for FTP and HTTPS

HTTPS Proxy 10.0.2.15 Port 8080

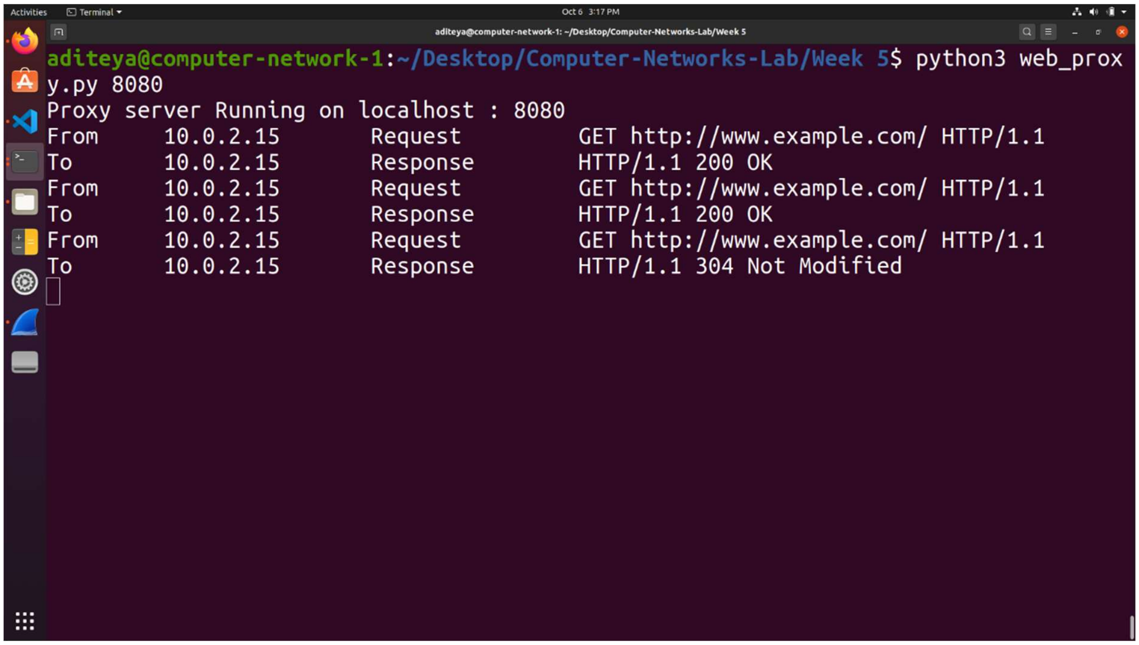
FTP Proxy 10.0.2.15 Port 8080

SOCKS Host Port 0

○ SOCKS v4 ● SOCKS v5

Browser Configuration Settings for Web Proxy

2.3 Web Proxy Server

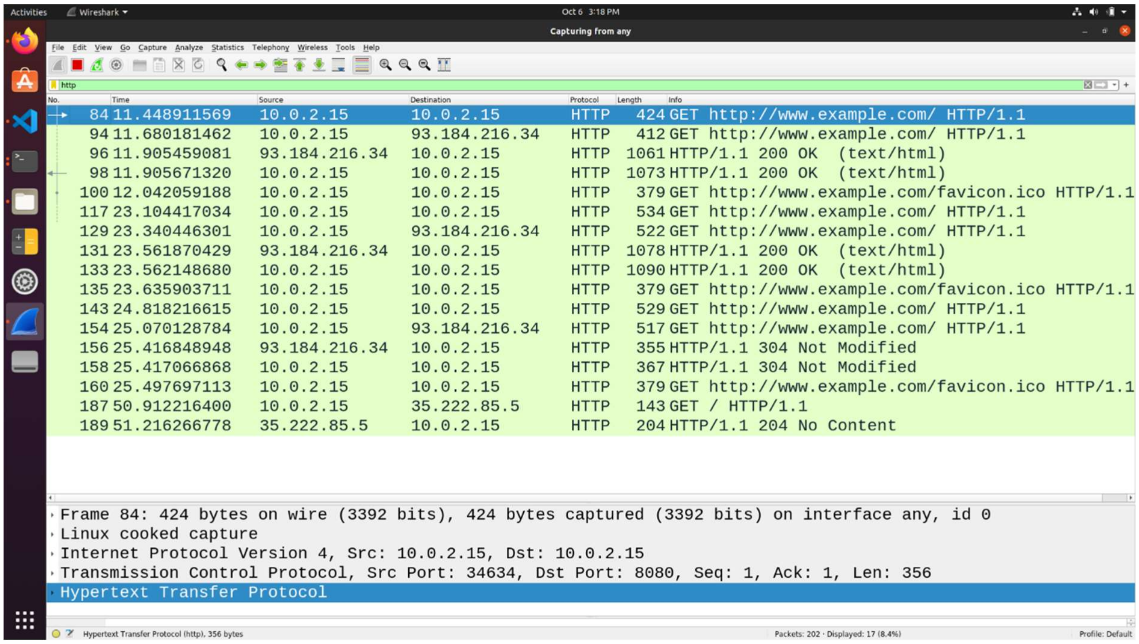


Activities
Terminal
Oct 6 3:17 PM
aditeya@computer-network-1: ~/Desktop/Computer-Networks-Lab/Week 5

```
aditeya@computer-network-1:~/Desktop/Computer-Networks-Lab/Week 5$ python3 web_prox
y.py 8080
Proxy server Running on localhost : 8080
From     10.0.2.15       Request         GET http://www.example.com/ HTTP/1.1
To       10.0.2.15       Response        HTTP/1.1 200 OK
From     10.0.2.15       Request         GET http://www.example.com/ HTTP/1.1
To       10.0.2.15       Response        HTTP/1.1 200 OK
From     10.0.2.15       Request         GET http://www.example.com/ HTTP/1.1
To       10.0.2.15       Response        HTTP/1.1 304 Not Modified
```

Request and Response Packets handled by Web Proxy

2.3 Wireshark Capture for Web Proxy


Activities
Wireshark
Oct 6 3:18 PM
Capturing from any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
http
```

```
No.  Time          Source          Destination     Protocol Length Info
  84 11.448911569  10.0.2.15       10.0.2.15       HTTP      424 GET http://www.example.com/ HTTP/1.1
  94 11.680181462  10.0.2.15       93.184.216.34   HTTP      412 GET http://www.example.com/ HTTP/1.1
  96 11.905459081  93.184.216.34   10.0.2.15       HTTP     1061 HTTP/1.1 200 OK  (text/html)
  98 11.905671320  10.0.2.15       10.0.2.15       HTTP     1073 HTTP/1.1 200 OK  (text/html)
 100 12.042059188  10.0.2.15       10.0.2.15       HTTP      379 GET http://www.example.com/favicon.ico HTTP/1.1
 117 23.104417034  10.0.2.15       10.0.2.15       HTTP      534 GET http://www.example.com/ HTTP/1.1
 129 23.340446301  10.0.2.15       93.184.216.34   HTTP      522 GET http://www.example.com/ HTTP/1.1
 131 23.561870429  93.184.216.34   10.0.2.15       HTTP     1078 HTTP/1.1 200 OK  (text/html)
 133 23.562148680  10.0.2.15       10.0.2.15       HTTP     1090 HTTP/1.1 200 OK  (text/html)
 135 23.635903711  10.0.2.15       10.0.2.15       HTTP      379 GET http://www.example.com/favicon.ico HTTP/1.1
 143 24.818216615  10.0.2.15       10.0.2.15       HTTP      529 GET http://www.example.com/ HTTP/1.1
 154 25.070128784  10.0.2.15       93.184.216.34   HTTP      517 GET http://www.example.com/ HTTP/1.1
 156 25.416848948  93.184.216.34   10.0.2.15       HTTP      355 HTTP/1.1 304 Not Modified
 158 25.417066868  10.0.2.15       10.0.2.15       HTTP      367 HTTP/1.1 304 Not Modified
 160 25.497697113  10.0.2.15       10.0.2.15       HTTP      379 GET http://www.example.com/favicon.ico HTTP/1.1
 187 50.912216400  10.0.2.15       35.222.85.5     HTTP      143 GET / HTTP/1.1
 189 51.216266778  35.222.85.5     10.0.2.15       HTTP      204 HTTP/1.1 204 No Content
```

```
Frame 84: 424 bytes on wire (3392 bits), 424 bytes captured (3392 bits) on interface any, id 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.15
Transmission Control Protocol, Src Port: 34634, Dst Port: 8080, Seq: 1, Ack: 1, Len: 356
Hypertext Transfer Protocol
```

```
Hypertext Transfer Protocol (http), 356 bytes          Packets: 202 · Displayed: 17 (8.4%)          Profile: Default
```