CN Lab Report – Week 5
PES1UG 21CS361
Nandan N

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 `socket` library 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_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 TCP Server

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

server_name = "10.0.4.26"
server_port = 12000
server_socket = socket(AF_INET, SOCK_STREAM)
server_socket.bind((server_name, server_port))
server_socket.listen(1)

print(f"Server {server_name} 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()
```
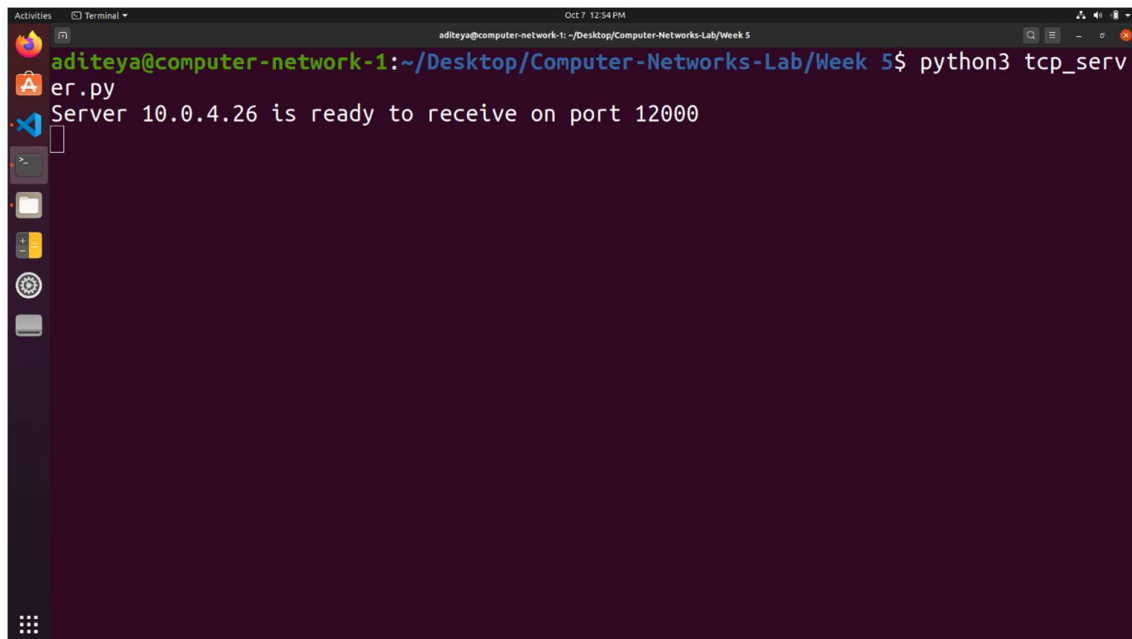
1.1.2 TCP Client

```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()
```

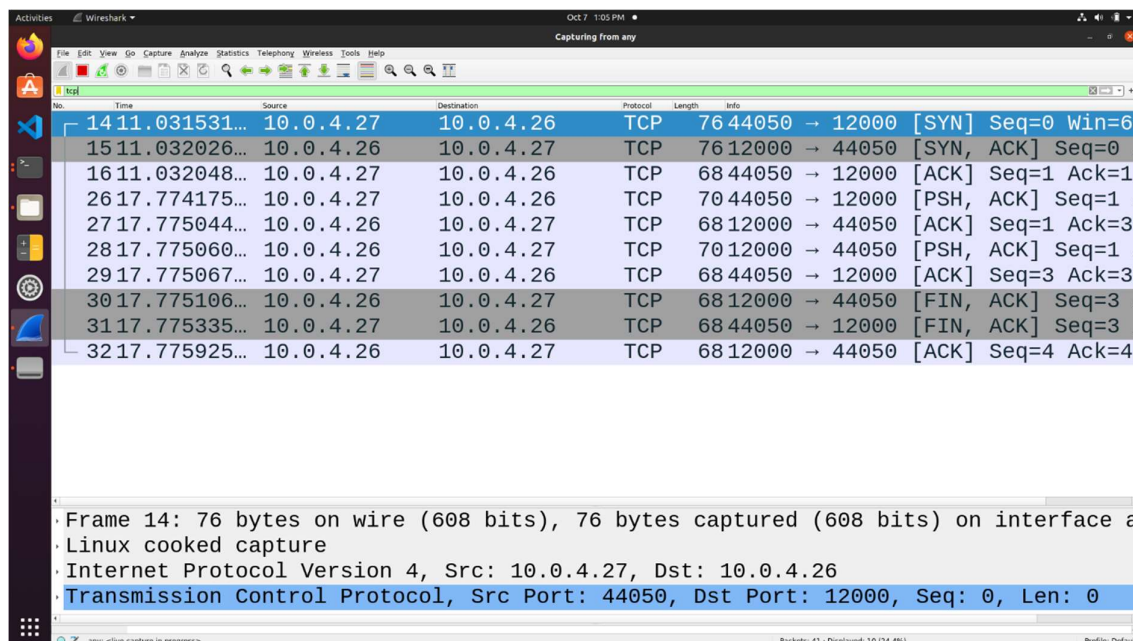### 1.1.3 TCP Connection between Server and Client

TCP Server

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 `socket` library 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_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 UDP Server

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

server_name = "10.0.4.26"
server_port = 12000
server_socket = socket(AF_INET, SOCK_DGRAM)
server_socket.bind((server_name, server_port))

print(f"Server {server_name} 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)
```
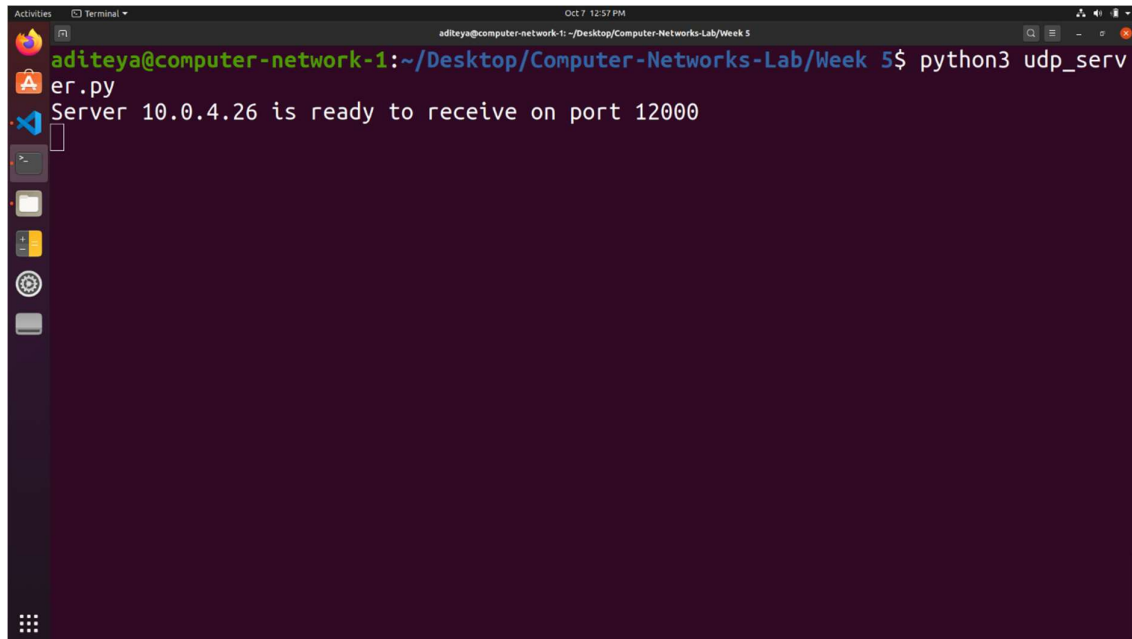
1.2.2 UDP Client

```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()
```
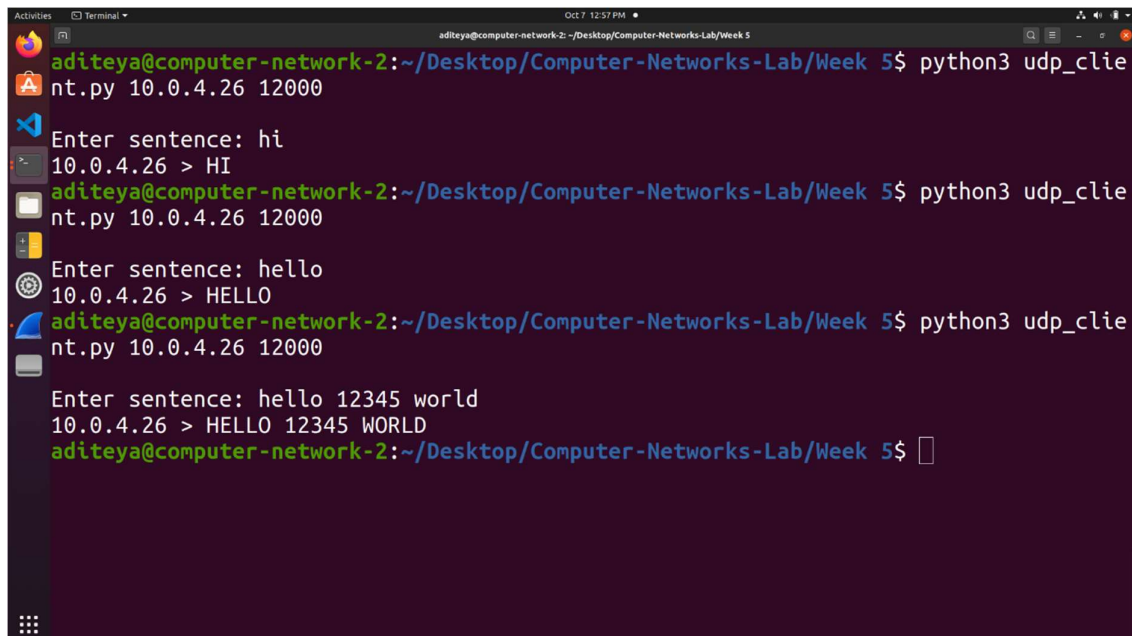
1.2.3 UDP Connection between Server and Client

UDP Server

UDP Client

1.2.4 Wireshark Capture for UDP Connection
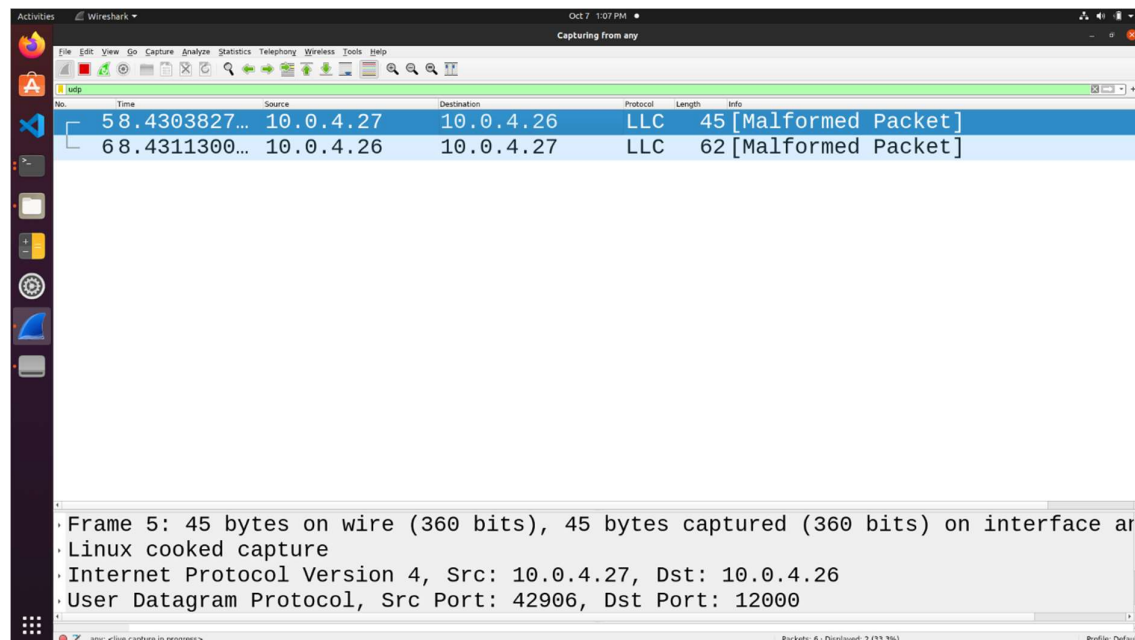
1.3 Problems

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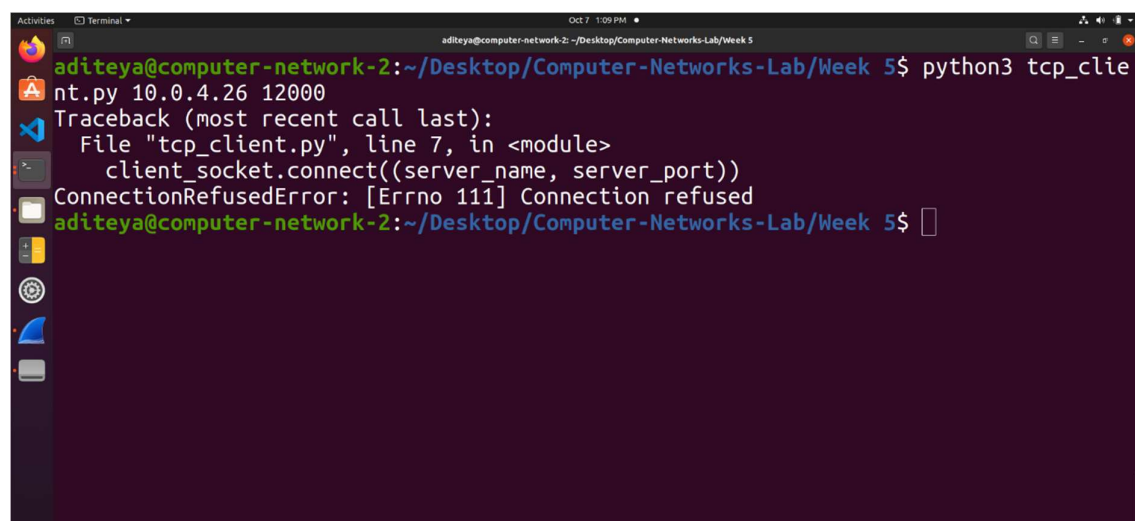
Q2. *Suppose you run UDPClient before you run UDPServer. What happens? Why?*

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 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.

```python
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]
```

```python
            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 server machine and the port number at which the socket application is being hosted on.

Browser Configuration Settings for Web Proxy

2.3 Web Proxy Server

- The website `www.example.com` is visited via the client browser.
- As shown below, the web proxy server receives the request instead from the client machine and forwards it to the destination server.
- Similarly, it receives a response packet from `www.example.com` and returns the same to the client machine.
- Hence, 3 pairs of request and response packets are received for each connection established between the client, the web proxy, and the server
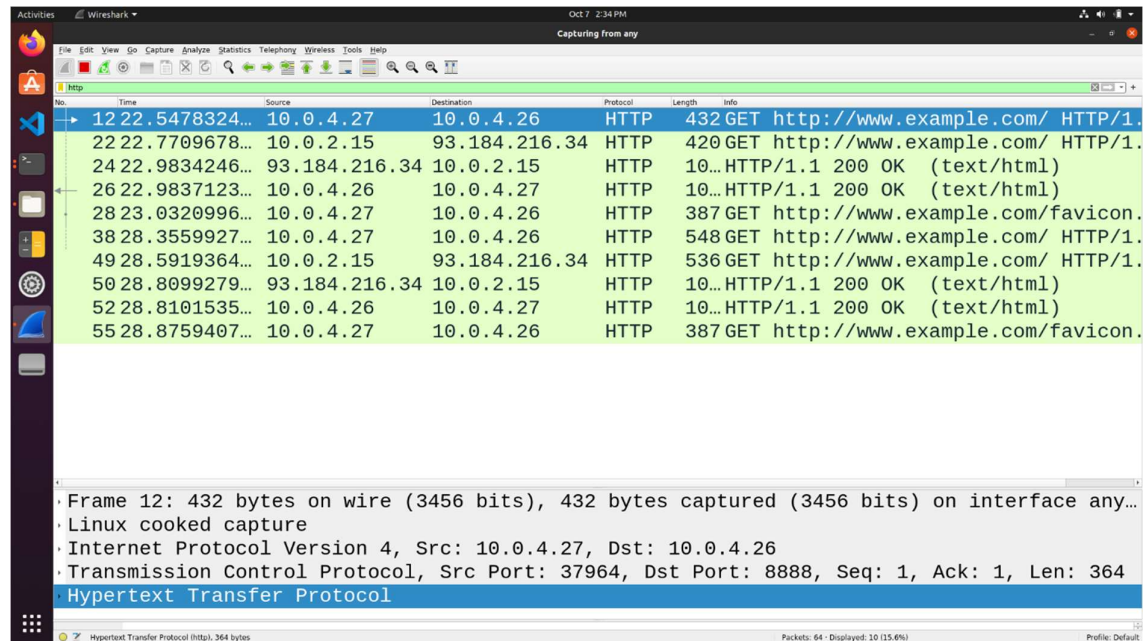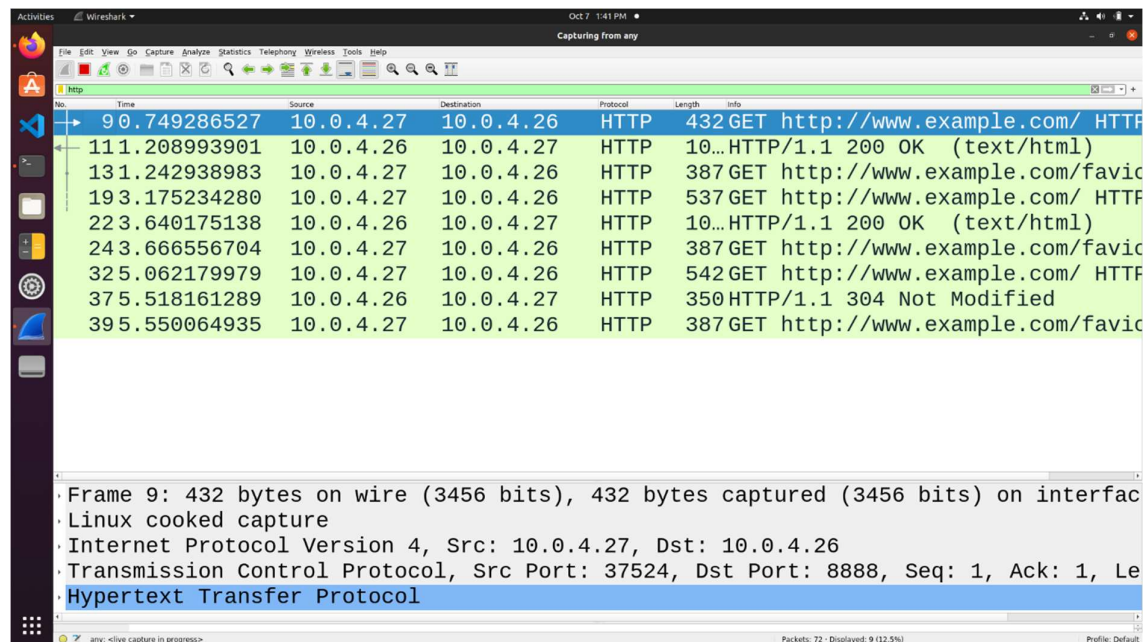

Request and Response Packets handled by Web Proxy

2.3 Wireshark Capture for Web Proxy

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

Wireshark Capture for Server Machine

Wireshark Capture for Client Machine