CN Lab Report – Week 5

PES1201800366

Aditeya Baral

**1. Socket Programming**

1. Create an application that will
   1. Convert lowercase letters to uppercase
      * e.g. [a…z] to [A…Z]
      * code will not change any special characters, e.g. &\*!
   2. 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**

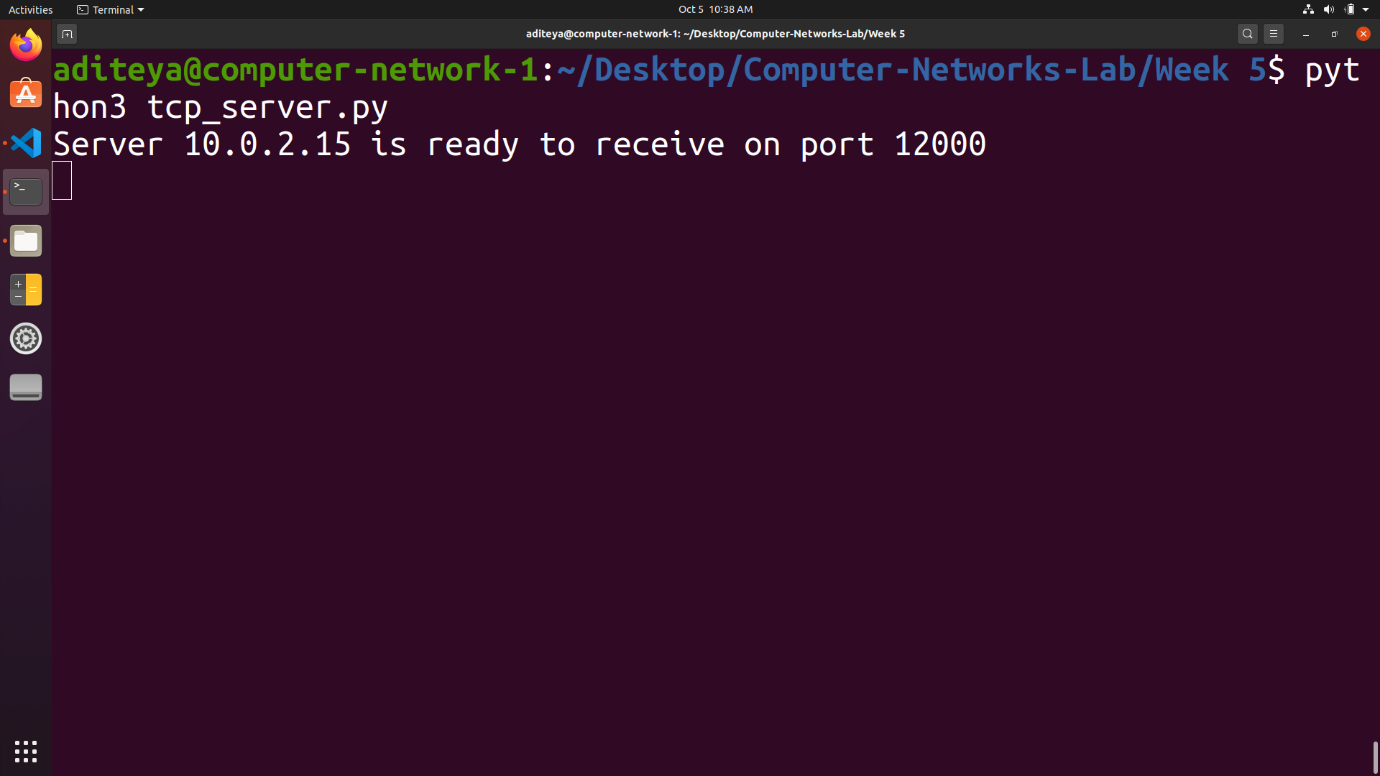
**1.1.1 TCP Server**

**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()

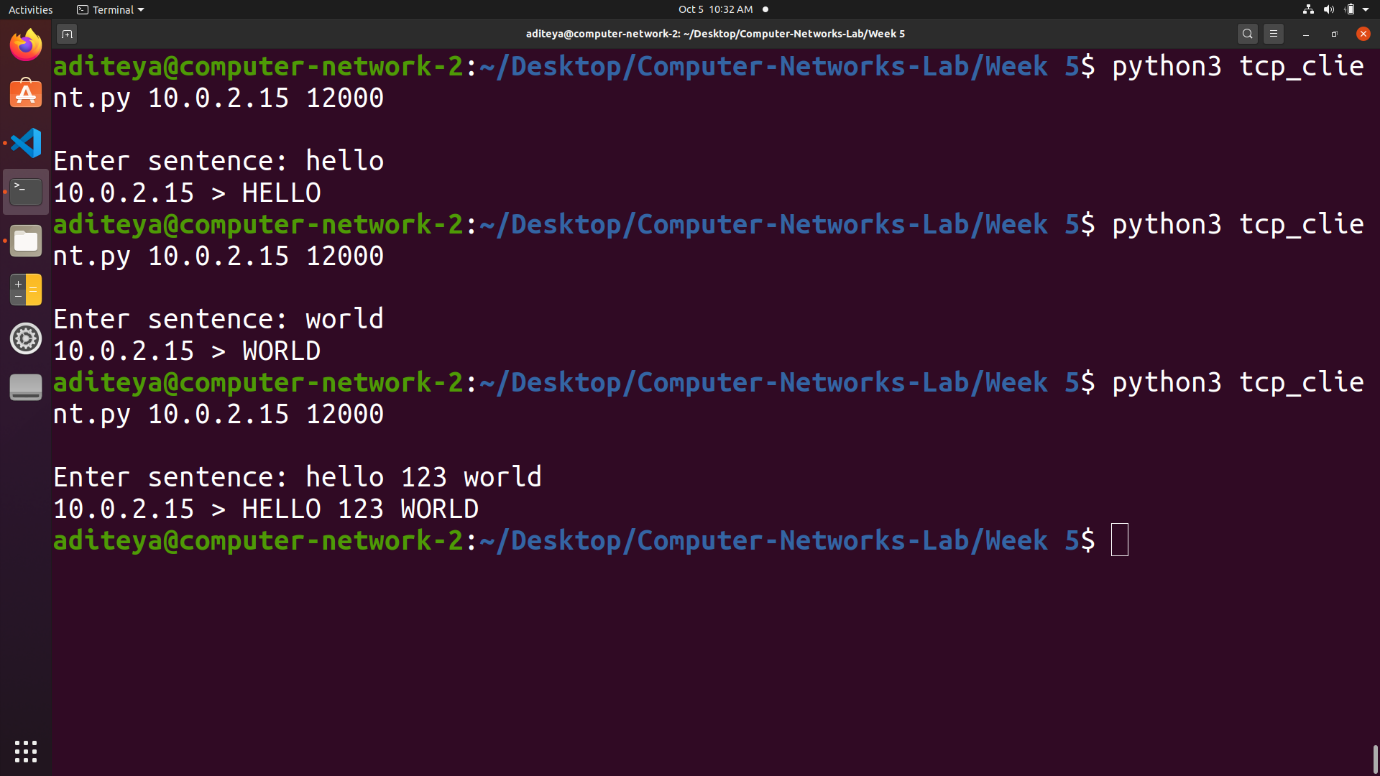
**1.1.2 TCP Client**

**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**

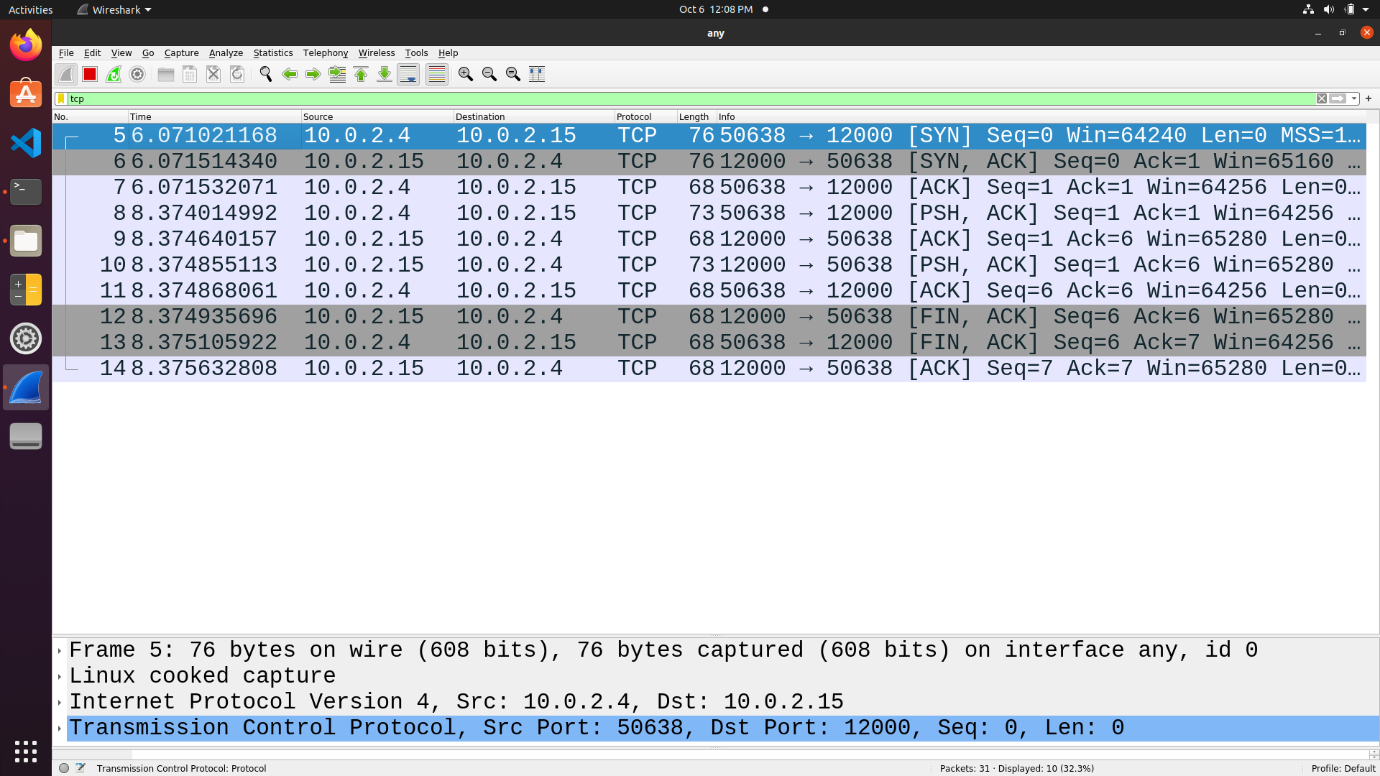


TCP Server



TCP Client

**1.1.4 Wireshark Capture for TCP Connection**



**1.2 UDP Connection**

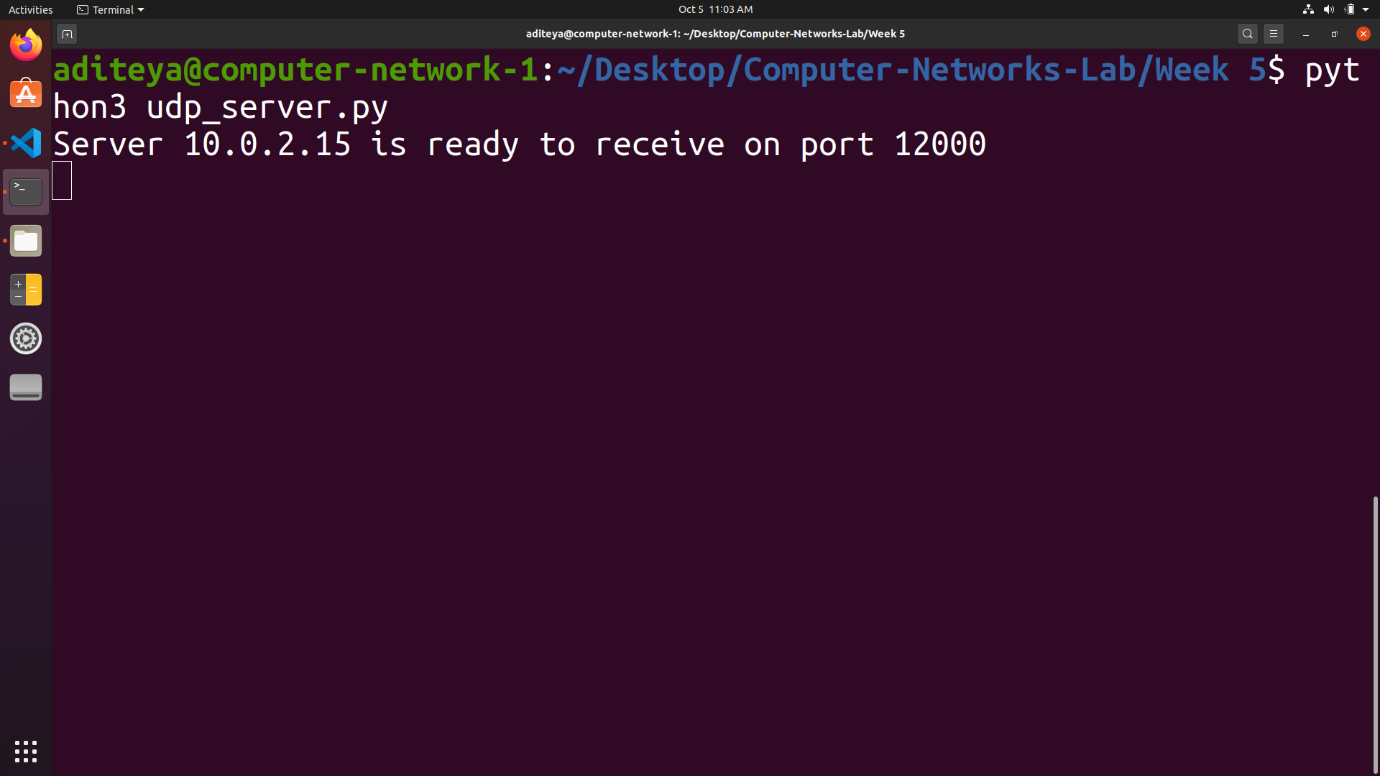
**1.2.1 UDP Server**

**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)

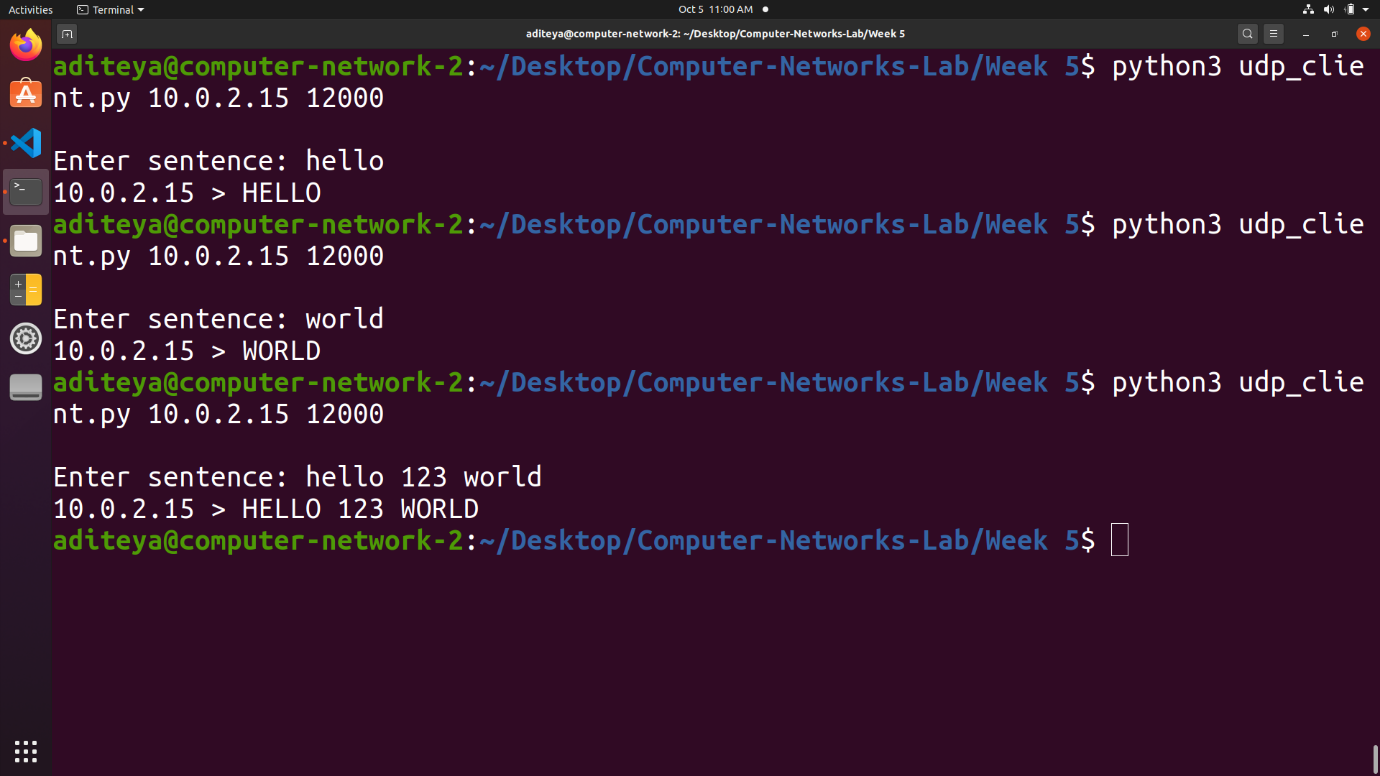
**1.2.2 UDP Client**

**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**

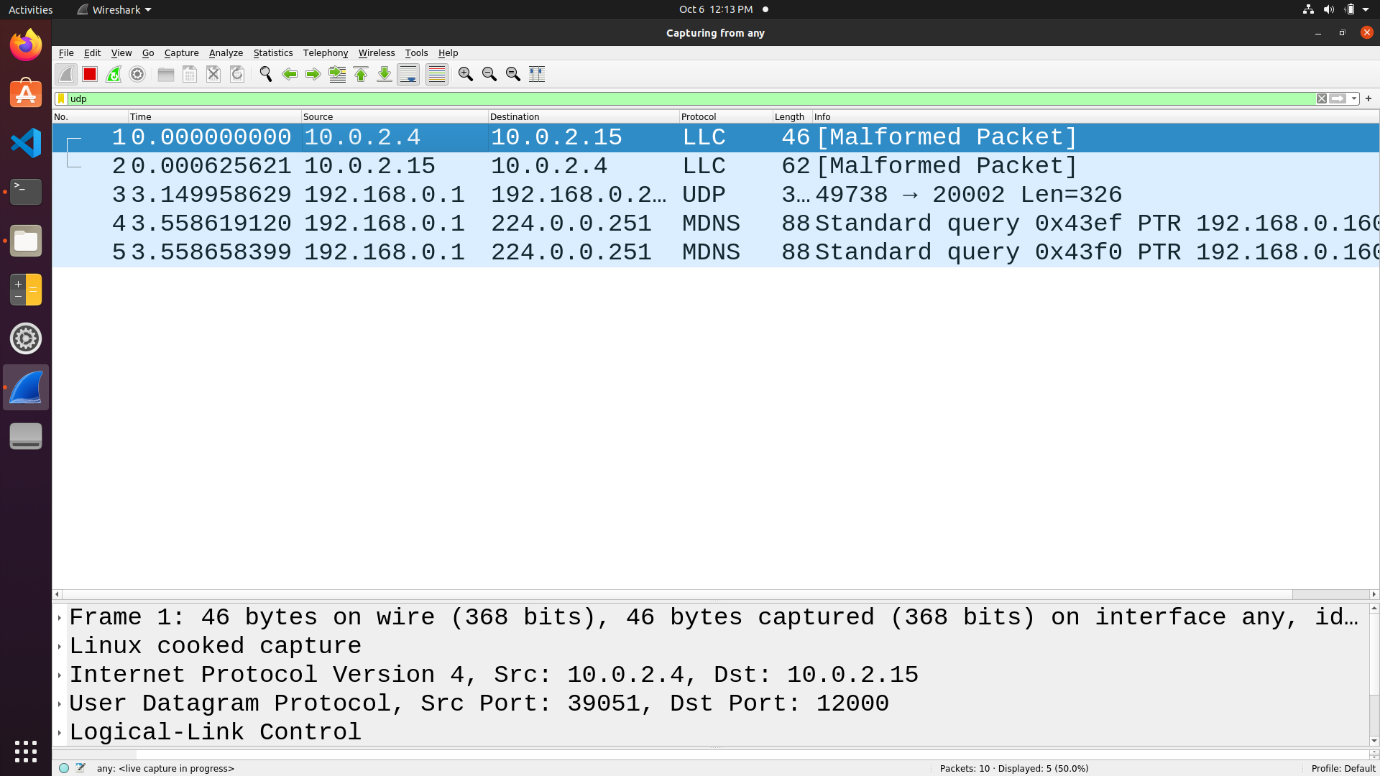


UDP Server



UDP Client

**1.2.4 Wireshark Capture for UDP Connection**



**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.

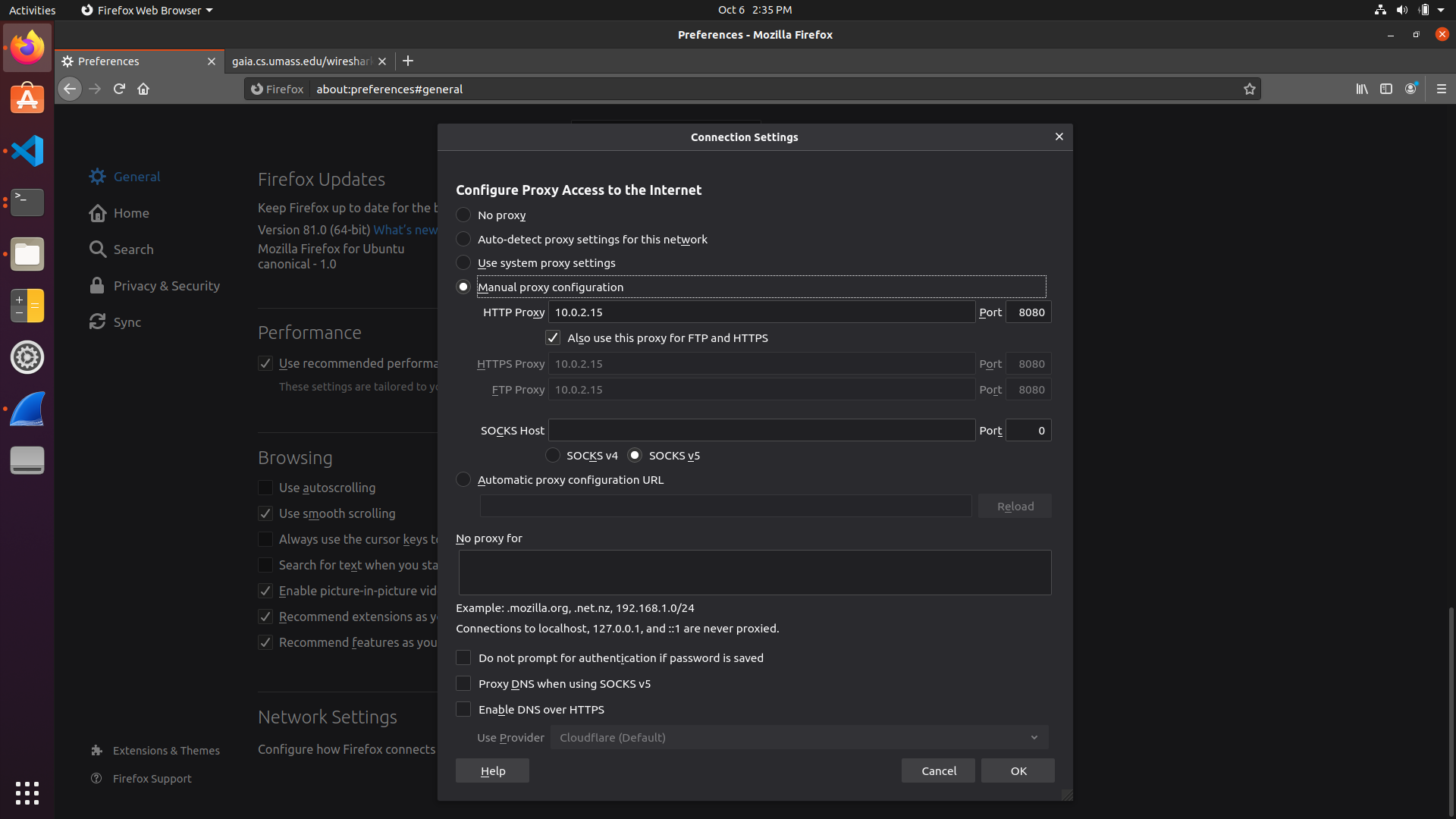
**2.1 Setting up a Web Proxy Server**

* 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()

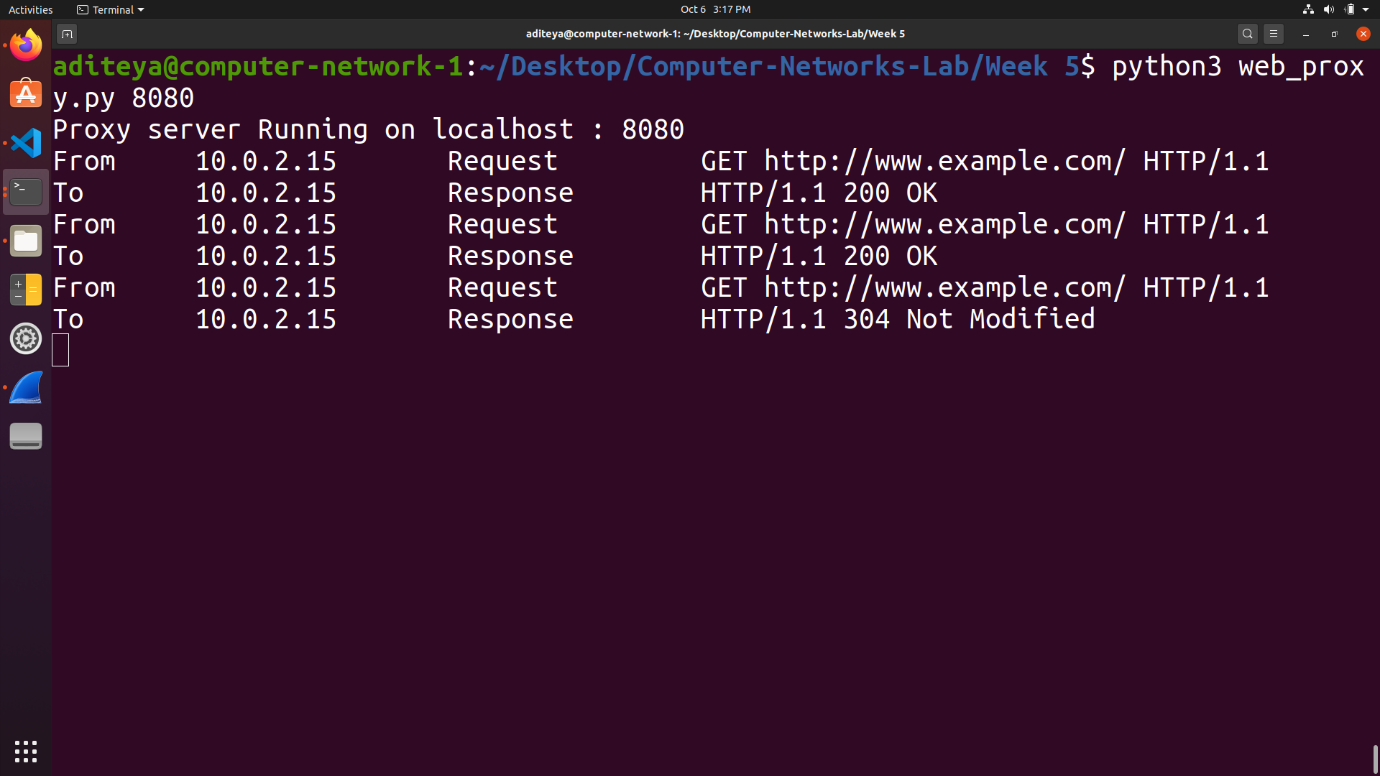
**2.2 Configuring Browser**

* 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.



Browser Configuration Settings for Web Proxy

**2.3 Web Proxy Server**



Request and Response Packets handled by Web Proxy

**2.3 Wireshark Capture for Web Proxy**

