جامعة بيرزيت

BIRZEIT UNIVERSITY

Faculty of Engineering & Technology
Electrical & Computer Engineering Department

Computer Networks ENCS3320

Socket Programming

Leen Alqazaqi	1220380
Tasneem Abusara	1220446
Moayyad Maraaba	1220887

Instructor: Dr.Ibrahim Nemer

Section: 1

Date: 15/8/2024

1 Table of Contents

2 Table of Figures

3 Introduction

This project explores multiple concepts in computer networks through three different tasks. Each task helps us understand and master different concepts. Task one focuses on some fundamental commands in computer networks; ping, nslookup, tracert, and telnet. This task helps us understand the use of each one in detail and gives a general idea about theWireShark tool. Task two focuses on the two types of connections, TCP and UDP, providing solid knowledge about their main differences, in sockets and code. Finally, the third task, which implements a web server that responds according to the HTTP request sent to it. This task provides us with background knowledge about web servers, HTML, and HTTP requests and responses.

4 Theory

- **Wireshark:** Wireshark is a powerful tool that uses sniffing to track and analyze all network activities related to our device.

- **TCP/UDP**: TCP and UDP are two different protocols in the transport layer. Each with different approaches to establishing a connection. TCP connections begin with a request to connect, known as a three-way handshake. This request guarantees that the server is available and ready to receive from the client. in contrast, a UDP connection does not involve a connection request so no guarantee.

- **Socket programming:** sockets are communication endpoints in the application layer that allow processes to communicate with each other. Socket programming shows how to establish these connections over the network.

- **HTTP (Hypertext Transfer Protocol)**: is a fundamental protocol in the application layer that supports transferring data over the web, between the server and the web browser. HTTP supports various methods and versions.

- **Web servers**: web servers are specialized servers that provide web browsers (clients) with requested data such as text, images, and others.

5 Procedure

Task 1

In this task, we learned about the commands “ping”, “nslookup”, “tracert” and “telnet” .We executed them using terminal on both Windows and Linux, then we looked up some information about the website www.ox.ac.uk using the “bgpview" website and bgp tool. Finally, we used the tool Wireshark to capture some DNS messages.

Task 2

We created two applications one is aserver-client TCP connection and the other uses a server-client approach where all peers can send and receive. The main difference between UDP and TCP is in the sockets and their communication (shown clearly in the code). We used PYcharm compiler . The flow chart below shows how TCP and UDP connections are [1]

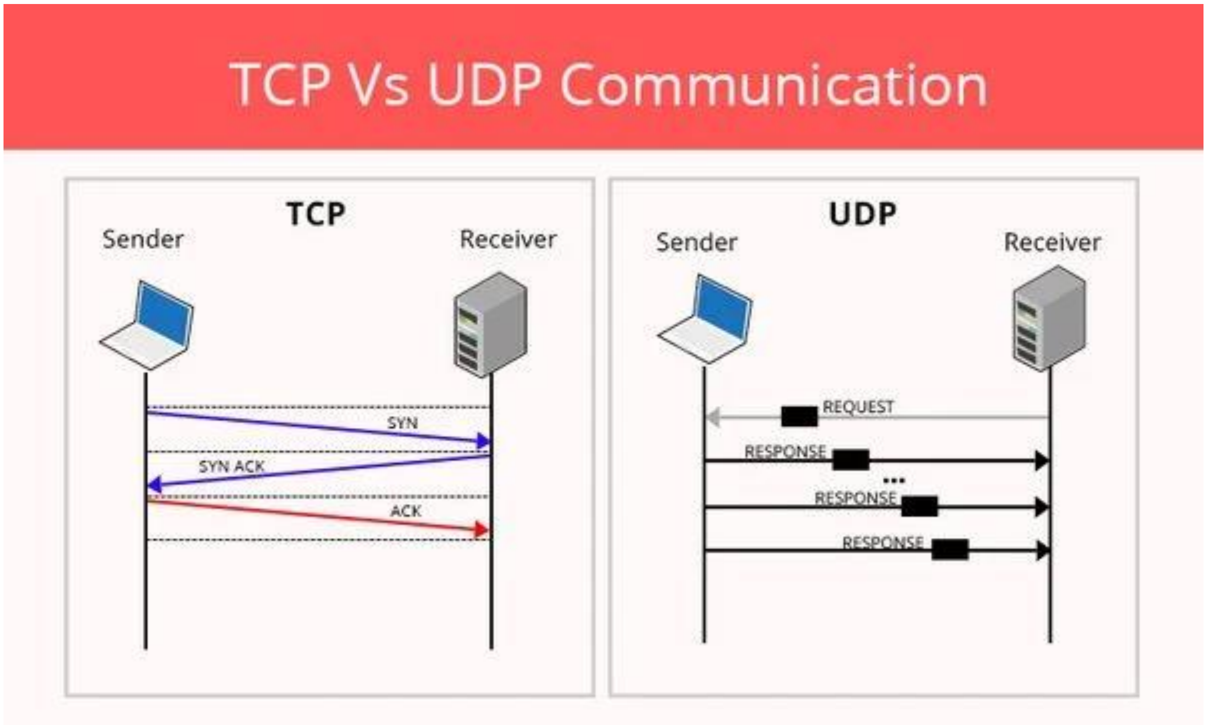


Figure 5.1: diagram showing TCP vs UDP

Task 3

In this task, we built a web server and a web page. for the front end, we used html and some JavaScript to control certain elements such as the fading-out buttons. For the backend, we used Python.

6 Results and Discussions

Task 1: commands & Wireshark

1. What are ping, tracert, nslookup, and telnet

ping: a command to find the time taken from the source (my device) to the destination (the web name or IP specified).

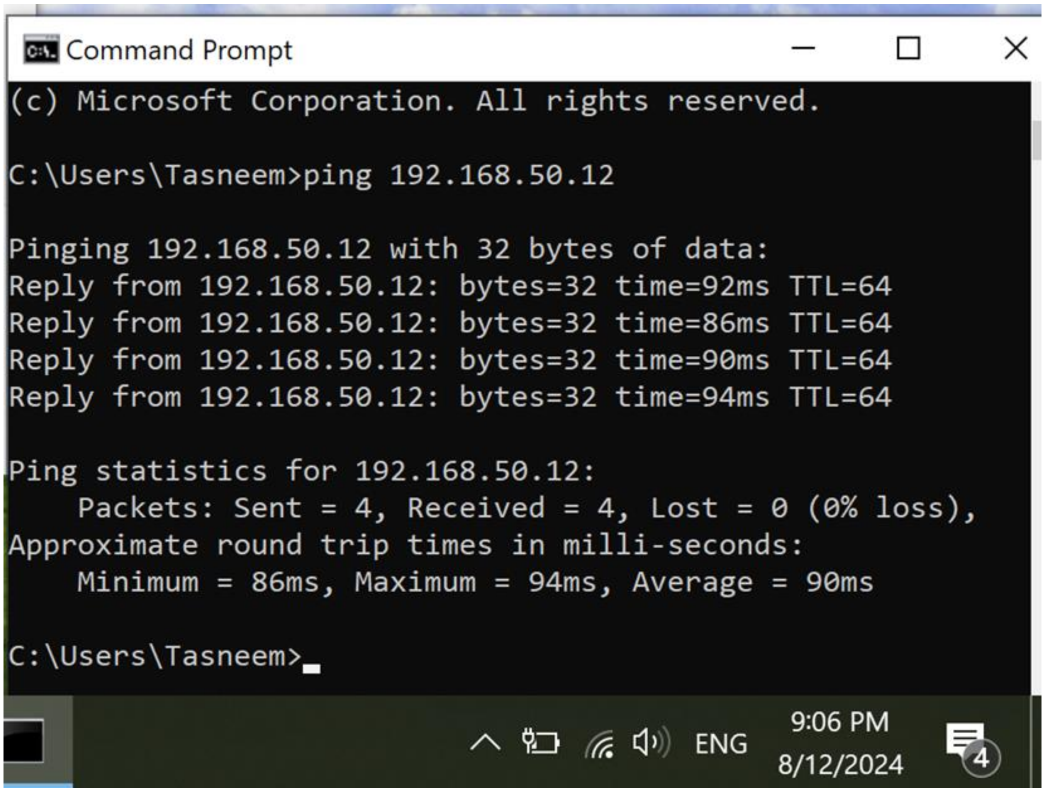
tracert: a command that tracks and lists the path of routers taken by the data to travel from the source to the destination.

lookup: a command that looks up the IP address for a web and vice versa, similar to what the DNS does.

telnet: a command that gives me the ability to send HTTP requests directly by writing them after establishing a telnet connection to the website

2. running some commands

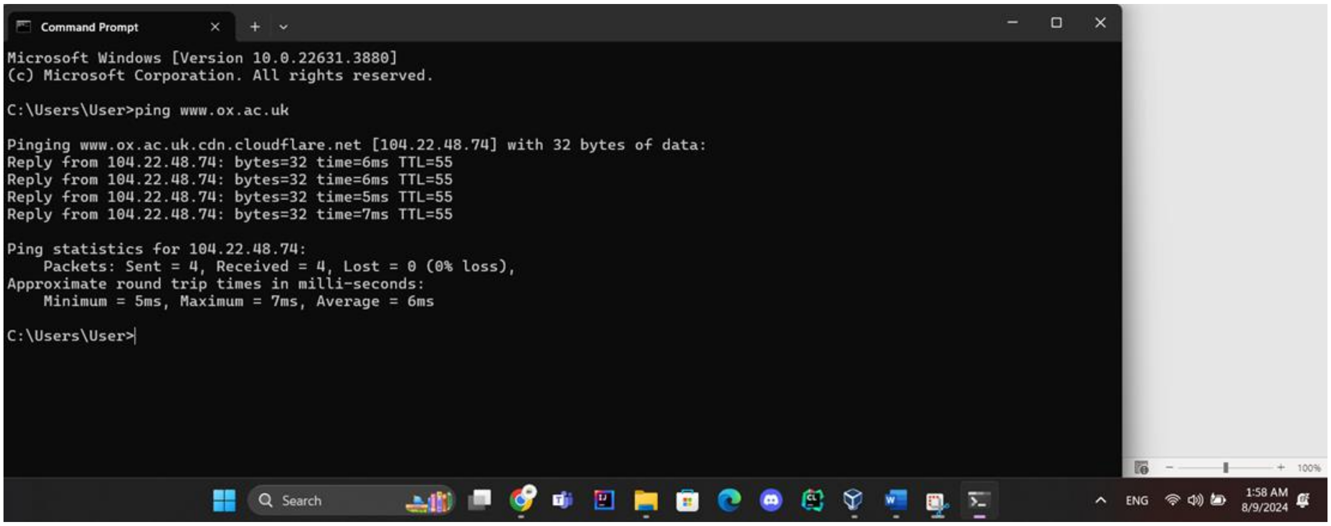
a. ping from laptop to smartphone



```
(c) Microsoft Corporation. All rights reserved.

C:\Users\Tasneem>ping 192.168.50.12

Pinging 192.168.50.12 with 32 bytes of data:
Reply from 192.168.50.12: bytes=32 time=92ms TTL=64
Reply from 192.168.50.12: bytes=32 time=86ms TTL=64
Reply from 192.168.50.12: bytes=32 time=90ms TTL=64
Reply from 192.168.50.12: bytes=32 time=94ms TTL=64

Ping statistics for 192.168.50.12:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 86ms, Maximum = 94ms, Average = 90ms

C:\Users\Tasneem>_
```

Figure 6.1: ping another device on the same network (cmd)

b. ping www.ox.ac.uk



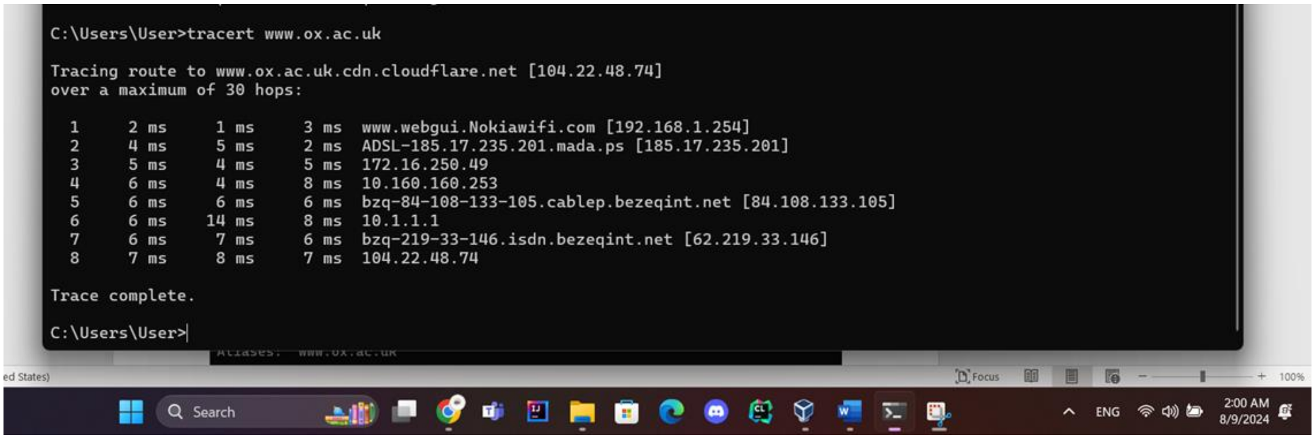
```
Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>ping www.ox.ac.uk

Pinging www.ox.ac.uk.cdn.cloudflare.net [104.22.48.74] with 32 bytes of data:
Reply from 104.22.48.74: bytes=32 time=6ms TTL=55
Reply from 104.22.48.74: bytes=32 time=6ms TTL=55
Reply from 104.22.48.74: bytes=32 time=5ms TTL=55
Reply from 104.22.48.74: bytes=32 time=7ms TTL=55

Ping statistics for 104.22.48.74:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 5ms, Maximum = 7ms, Average = 6ms

C:\Users\User>
```

Figure 6.2: ping website(cmd)

c. specifying the location of the server from where we got the response

in the first part we received the response from the same server. in the second part, the response was from a device with 104.22.49.74 IP address, looking up this IP, we found that the response was from United States San Jose Cloudflare Inc.

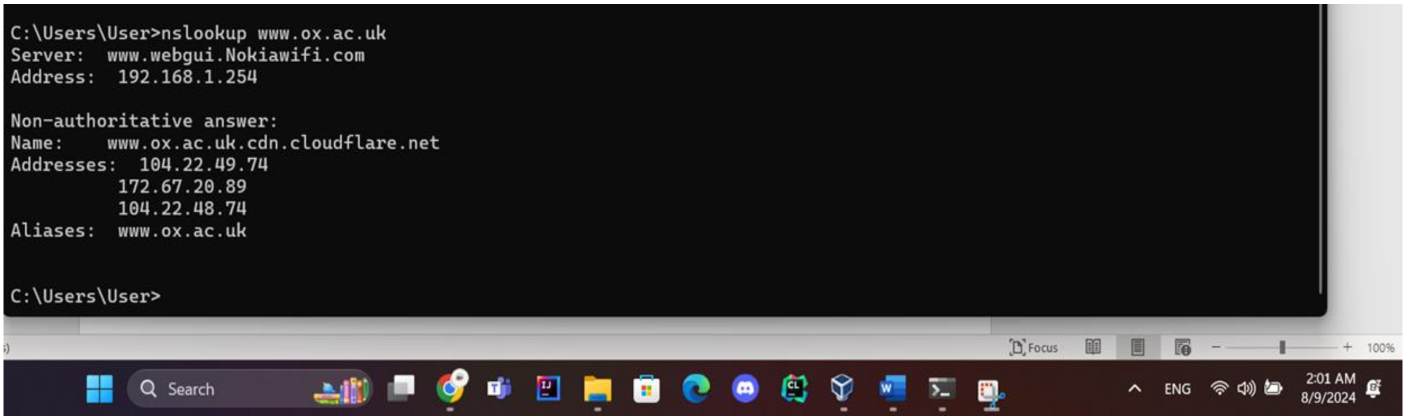
d. tracert www.ox.ac.uk



```
C:\Users\User>tracert www.ox.ac.uk

Tracing route to www.ox.ac.uk.cdn.cloudflare.net [104.22.48.74]
over a maximum of 30 hops:

  1     2 ms     1 ms     3 ms  www.webgui.Nokiawifi.com [192.168.1.254]
  2     4 ms     5 ms     2 ms  ADSL-185.17.235.201.mada.ps [185.17.235.201]
  3     5 ms     4 ms     5 ms  172.16.250.49
  4     6 ms     4 ms     8 ms  10.160.160.253
  5     6 ms     6 ms     6 ms  bzq-84-108-133-105.cablep.bezeqint.net [84.108.133.105]
  6     6 ms    14 ms     8 ms  10.1.1.1
  7     6 ms     7 ms     6 ms  bzq-219-33-146.isdn.bezeqint.net [62.219.33.146]
  8     7 ms     8 ms     7 ms  104.22.48.74

Trace complete.

C:\Users\User>
```

Figure 6.3: tracert command (cmd)

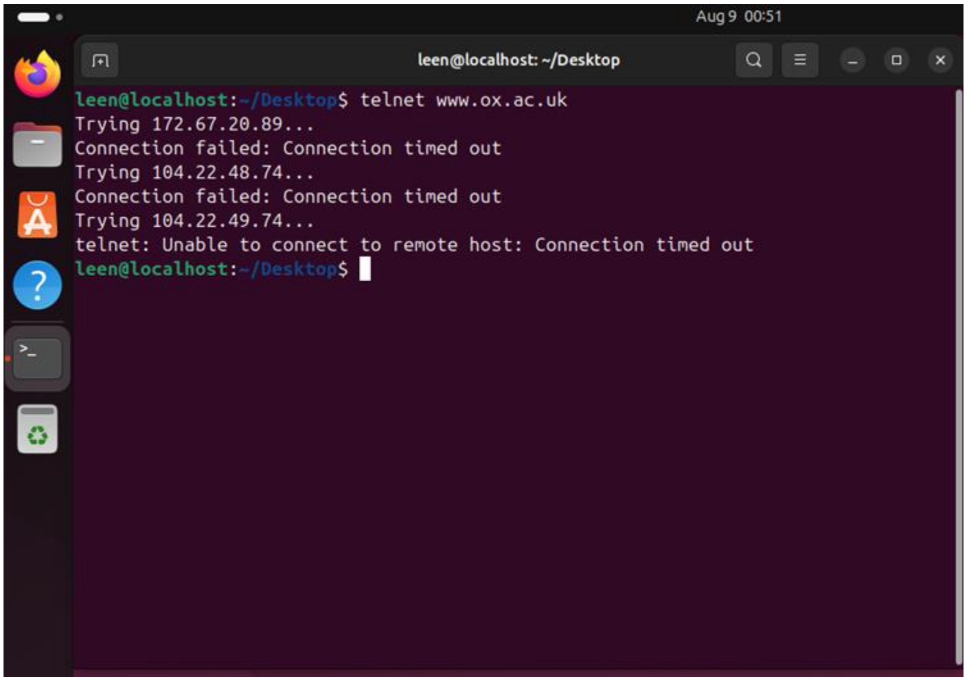
e. nslookup www.ox.ac.uk



```
C:\Users\User>nslookup www.ox.ac.uk
Server:  www.webgui.Nokiawifi.com
Address:  192.168.1.254

Non-authoritative answer:
Name:    www.ox.ac.uk.cdn.cloudflare.net
Addresses:  104.22.49.74
          172.67.20.89
          104.22.48.74
Aliases:  www.ox.ac.uk


C:\Users\User>
```

Figure 6.4: Figure 6.4: nslookup on cmd

f. telnet www.ox.ac.uk



```
leen@localhost:~/Desktop$ telnet www.ox.ac.uk
Trying 172.67.20.89...
Connection failed: Connection timed out
Trying 104.22.48.74...
Connection failed: Connection timed out
Trying 104.22.49.74...
telnet: Unable to connect to remote host: Connection timed out
leen@localhost:~/Desktop$
```

Figure 6.5: telnet command (linux teminal)

- Most probably the router does not allow Telnet connections

3. some details about www.ox.ac.uk.

www.ox.ac.uk : 104.22.48.74

Autonomous system (AS) number: AS13335 //it’s like an ID for the network

Number of IP’s = 4,096

parent Prefix: 104.22.48.0/20

prefix :104.16.0.0/12

number of prefixes: 3662

number of peers: 1296

Tier-1-ISP name

Any of the ISPs below can be considered Tier-1 ISP:

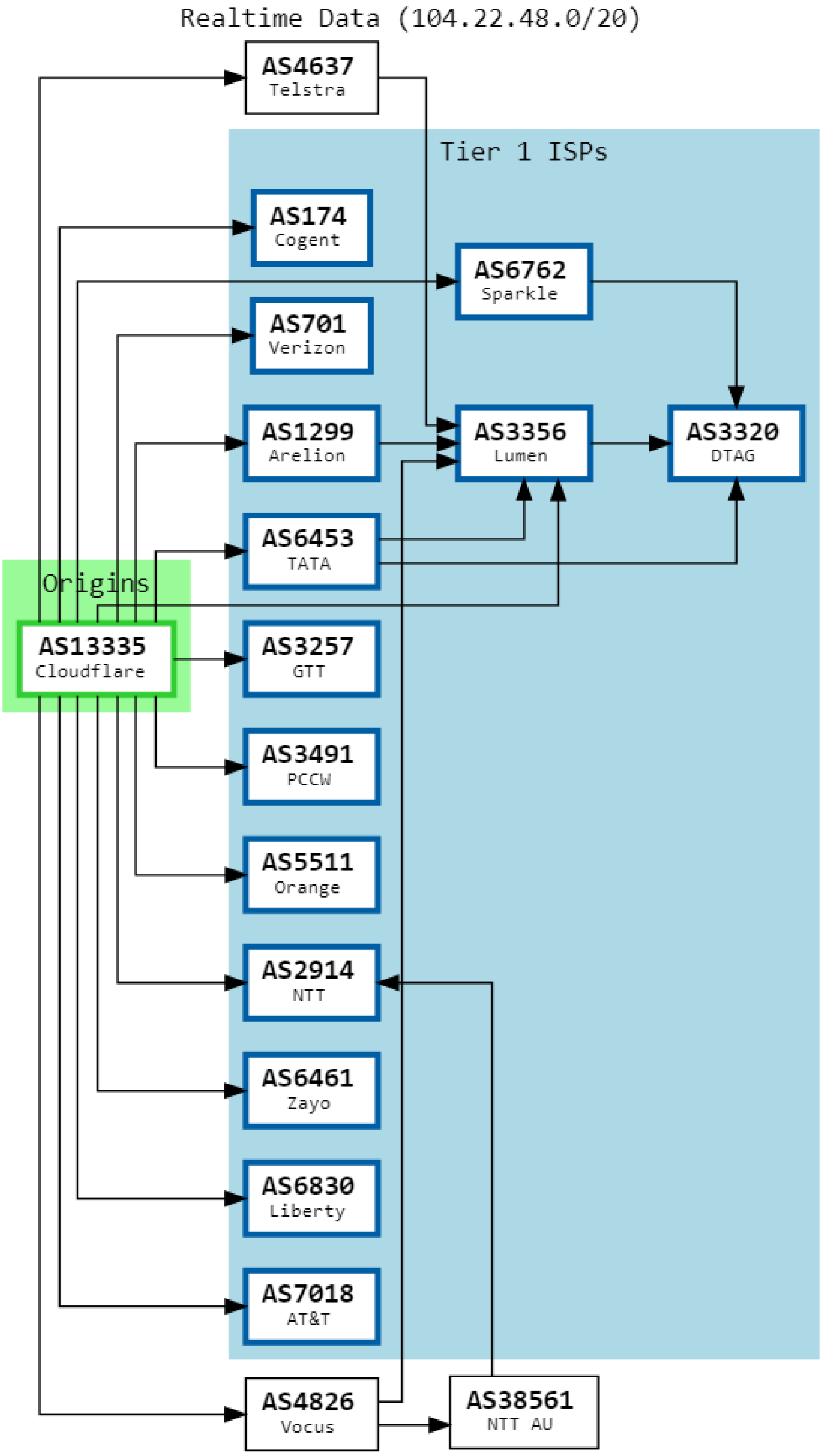
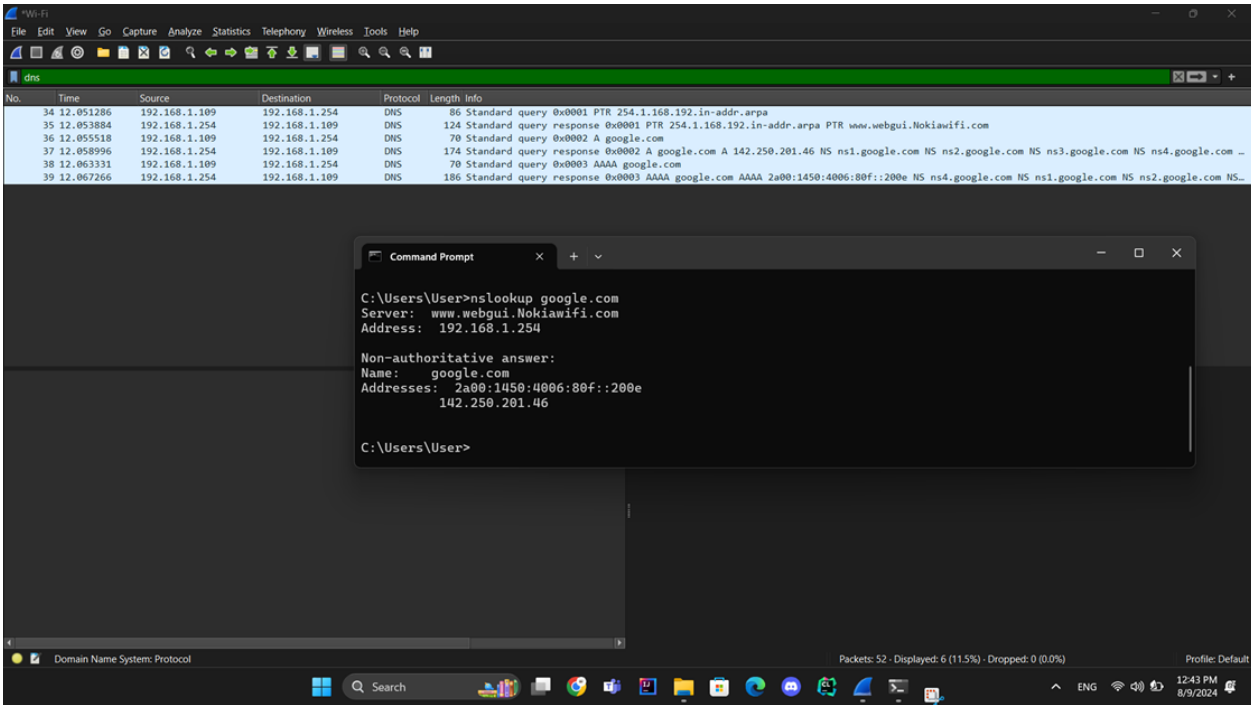


Figure 6.6: Website tier ISPs 1 [2].

4. Wireshark capturing some DNS messages


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
dns

No.	Time	Source	Destination	Protocol	Length	Info
34	12.051286	192.168.1.109	192.168.1.254	DNS	86	Standard query 0x0001 PTR 254.1.168.192.in-addr.arpa
35	12.053884	192.168.1.254	192.168.1.109	DNS	124	Standard query response 0x0001 PTR 254.1.168.192.in-addr.arpa PTR www.webgui.Nokiawifi.com
36	12.055518	192.168.1.109	192.168.1.254	DNS	70	Standard query 0x0002 A google.com
37	12.058996	192.168.1.254	192.168.1.109	DNS	174	Standard query response 0x0002 A google.com A 142.250.201.46 NS ns1.google.com NS ns2.google.com NS ns3.google.com NS ns4.google.com …
38	12.063331	192.168.1.109	192.168.1.254	DNS	70	Standard query 0x0003 AAAA google.com
39	12.067266	192.168.1.254	192.168.1.109	DNS	186	Standard query response 0x0003 AAAA google.com AAAA 2a00:1450:4006:80f::200e NS ns4.google.com NS ns1.google.com NS ns2.google.com NS…

```
C:\Users\User>nslookup google.com
Server:  www.webgui.Nokiawifi.com
Address:  192.168.1.254

Non-authoritative answer:
Name:    google.com
Addresses:  2a00:1450:4006:80f::200e
          142.250.201.46


C:\Users\User>
```

Domain Name System: Protocol
Packets: 52 · Displayed: 6 (11.5%) · Dropped: 0 (0.0%)
Profile: Default
Search
ENG
12:43 PM
8/9/2024

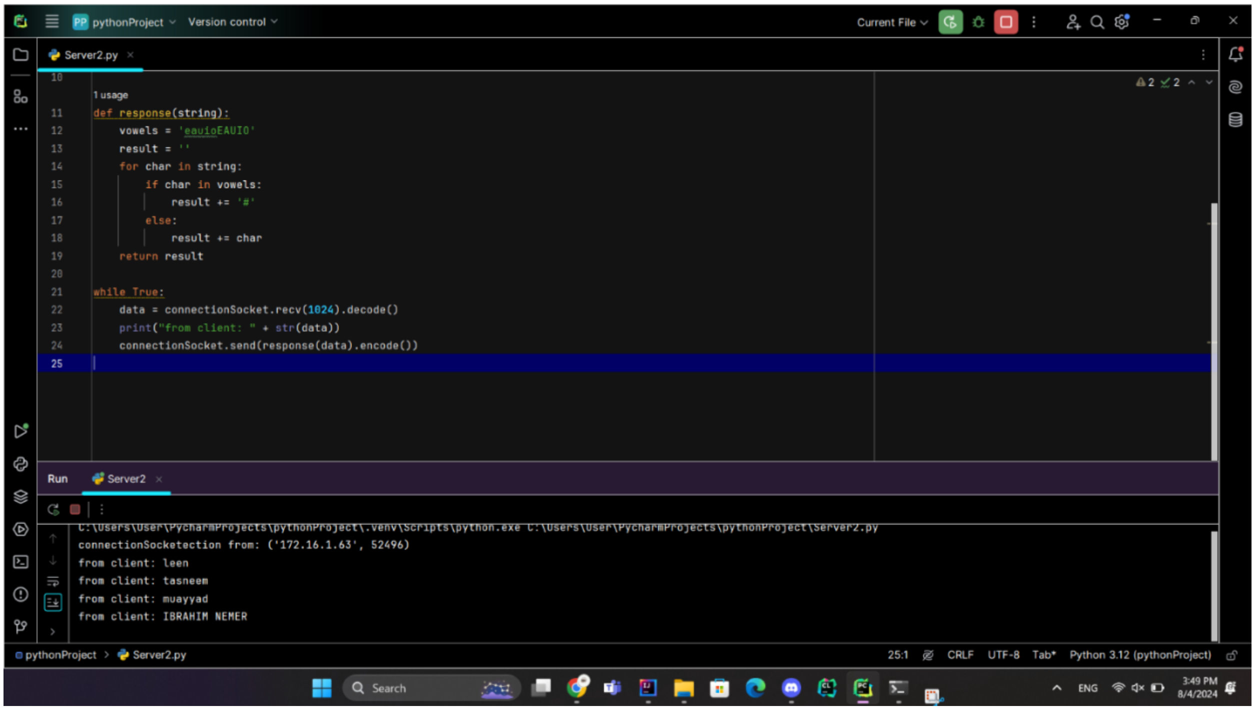
Figure 6.7: captured DNS messages

as shown above, Wireshark is a tool that tracks all the network activities related to my device, here we focused on capturing the DNS messages.

Task 2: socket programming (TCP&UDP)

1. In this task, the client sends words to the server through a TCP connection, and the server processes the data by replacing any vowel with a # symbol. The modified text is then sent back to the client

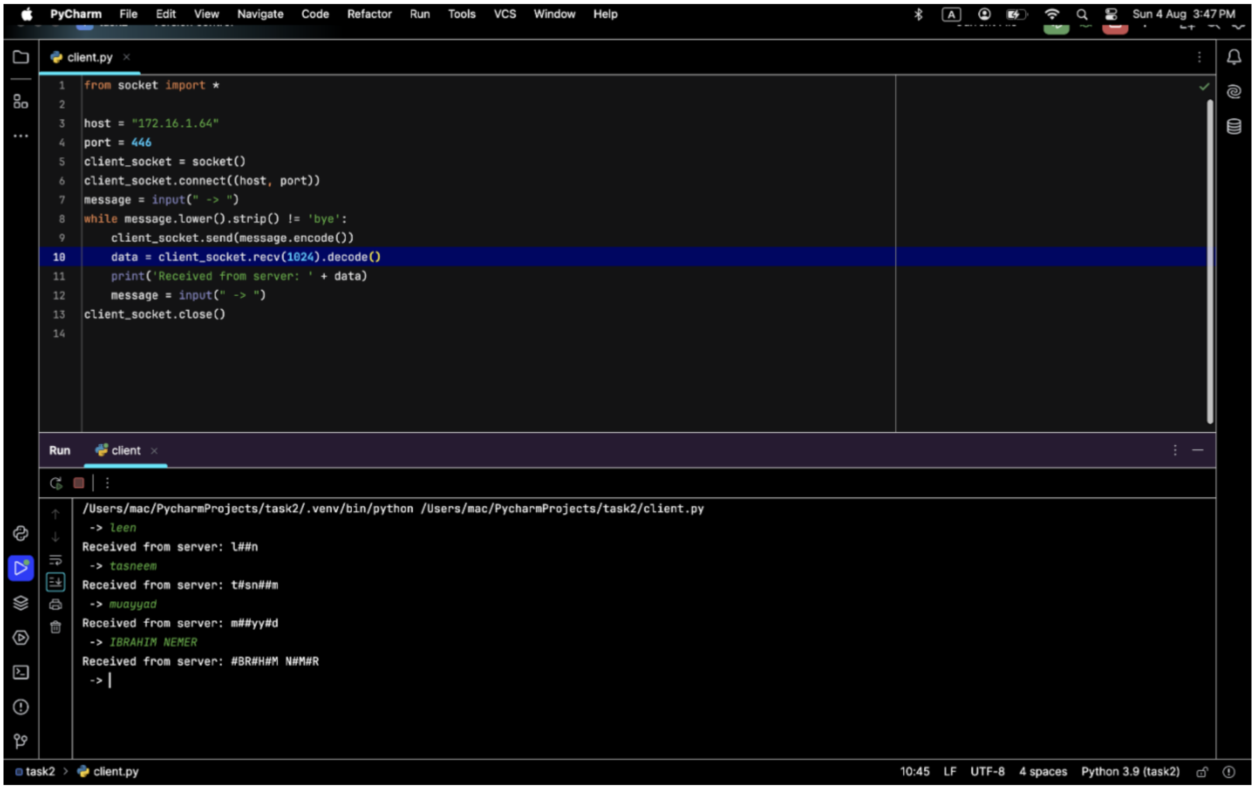
   The snapshot below is for a server and a client, as observed, any vowel in the input (sent by the client) is replaced with the # symbol by the server and sent back to the client.



```
1 usage
def response(string):
    vowels = 'eauioEAUIO'
    result = ''
    for char in string:
        if char in vowels:
            result += '#'
        else:
            result += char
    return result

while True:
    data = connectionSocket.recv(1024).decode()
    print("from client: " + str(data))
    connectionSocket.send(response(data).encode())

```

```
C:\Users\User\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\User\PycharmProjects\pythonProject\Server2.py
connectionSocketection from: ('172.16.1.63', 52496)
from client: leen
from client: tasneem
from client: muayyad
from client: IBRAHIM NEMER
```

Figure 6.8: server hosting device



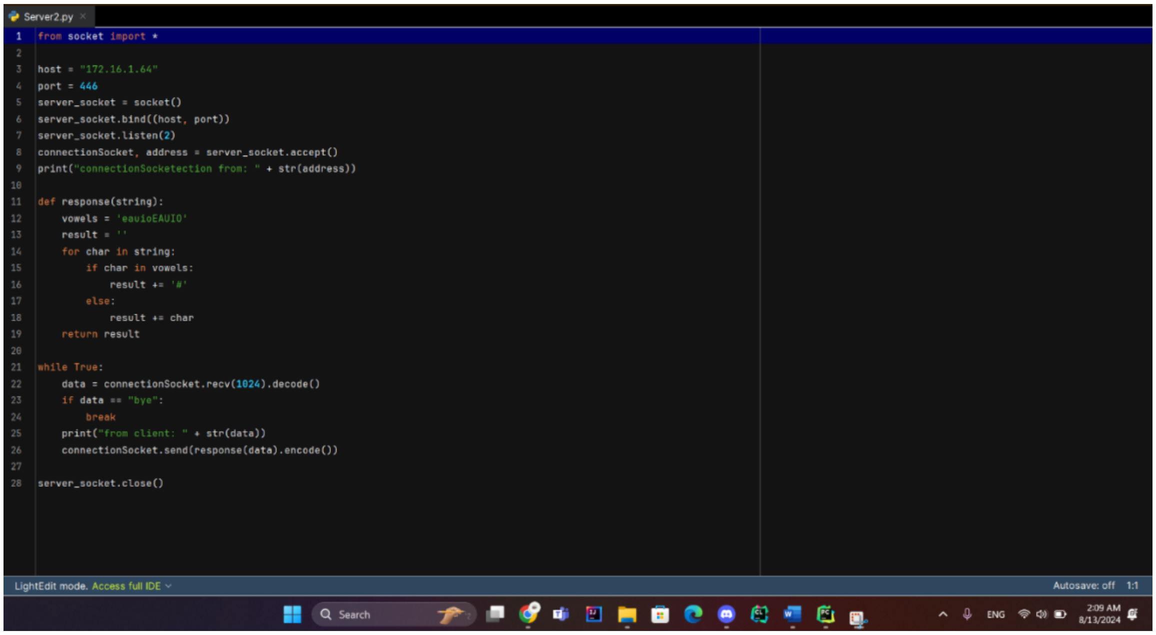
```
from socket import *

host = "172.16.1.64"
port = 446
client_socket = socket()
client_socket.connect((host, port))
message = input(" -> ")
while message.lower().strip() != 'bye':
    client_socket.send(message.encode())
    data = client_socket.recv(1024).decode()
    print('Received from server: ' + data)
    message = input(" -> ")
client_socket.close()

```

```
/Users/mac/PycharmProjects/task2/.venv/bin/python /Users/mac/PycharmProjects/task2/client.py
 -> leen
Received from server: l##n
 -> tasneem
Received from server: t#sn##m
 -> muayyad
Received from server: m##yy#d
 -> IBRAHIM NEMER
Received from server: #BR#H#M N#M#R
 -> 
```

Figure 6.9: client device

Here is the code for the previous server-client approach

Server's code



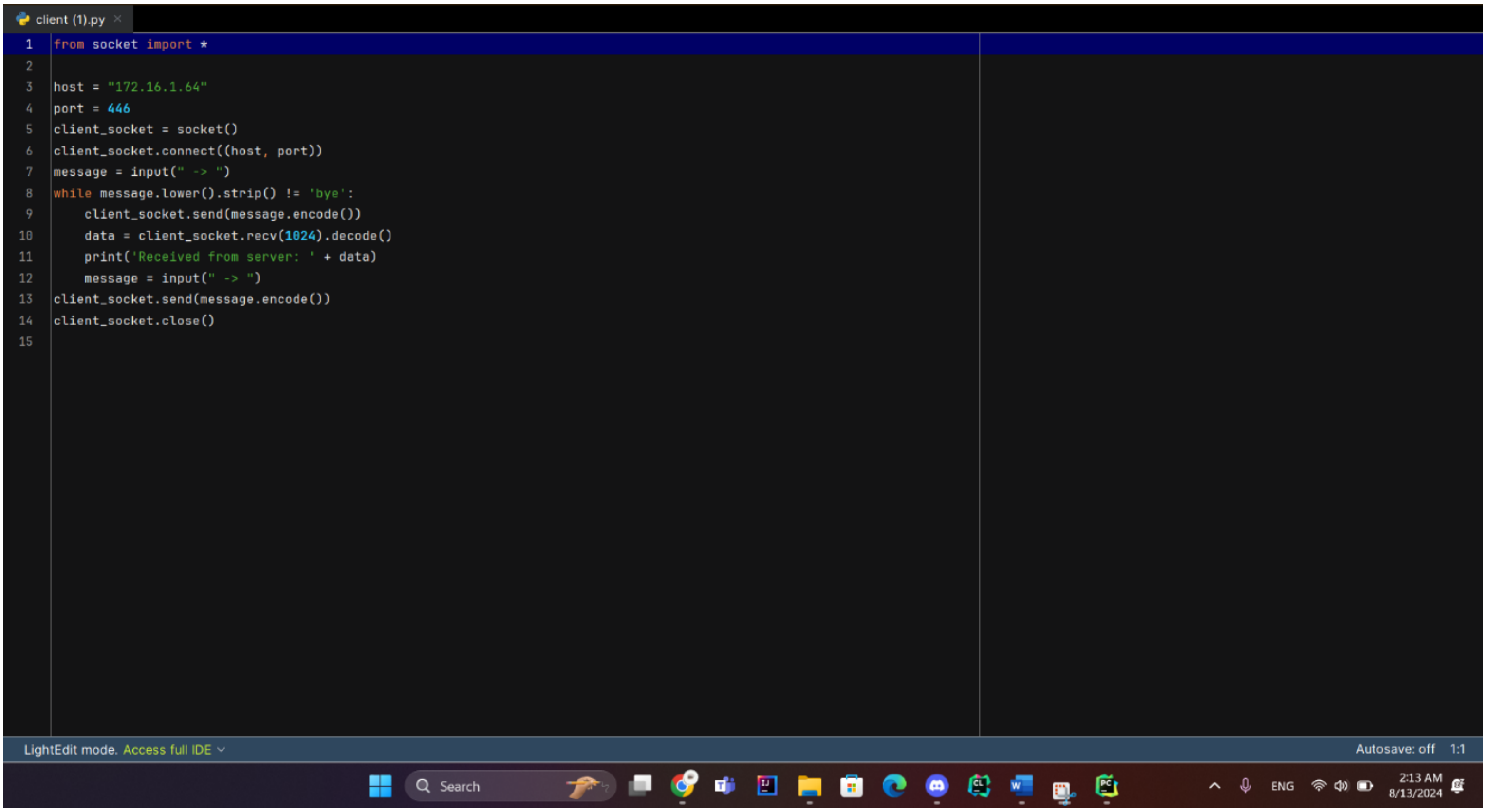
```python
from socket import *

host = "172.16.1.64"
port = 446
server_socket = socket()
server_socket.bind((host, port))
server_socket.listen(2)
connectionSocket, address = server_socket.accept()
print("connectionSocketection from: " + str(address))

def response(string):
    vowels = 'eauioEAUIO'
    result = ''
    for char in string:
        if char in vowels:
            result += '#'
        else:
            result += char
    return result

while True:
    data = connectionSocket.recv(1024).decode()
    if data == "bye":
        break
    print("from client: " + str(data))
    connectionSocket.send(response(data).encode())

server_socket.close()
```

Figure 6.10: Python code for server host

As shown, we connected to port 446 and used functions from the socket package as “listen”, “bind”, “recv”, and others. All of these functions are specifically designed for **TCP connection.**

The server receives from the client and replaces vowels with # before sending it back to it.

Client's code



```python
from socket import *

host = "172.16.1.64"
port = 446
client_socket = socket()
client_socket.connect((host, port))
message = input(" -> ")
while message.lower().strip() != 'bye':
    client_socket.send(message.encode())
    data = client_socket.recv(1024).decode()
    print('Received from server: ' + data)
    message = input(" -> ")
client_socket.send(message.encode())
client_socket.close()
```

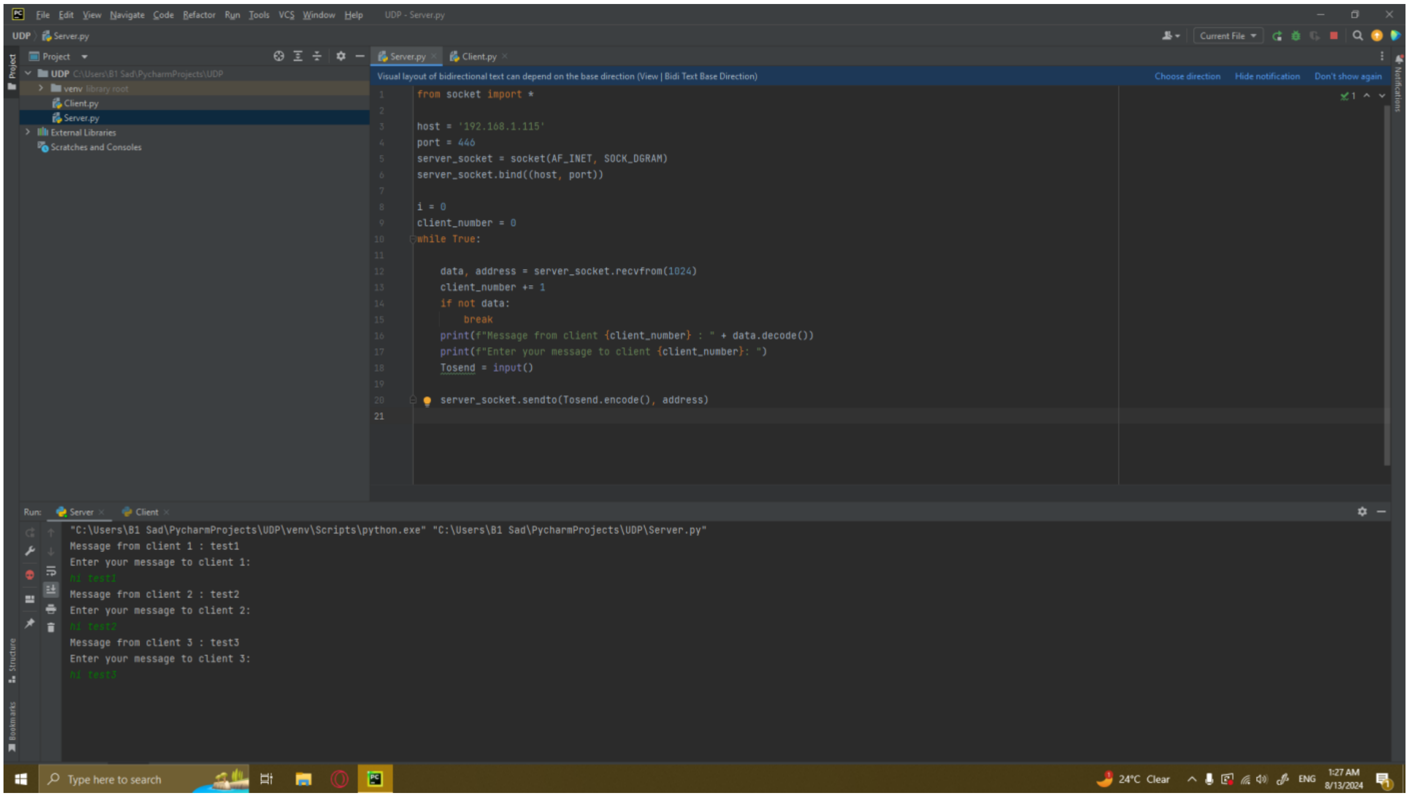
Figure 6.11: Python code for client side

The client also uses some functions that are for TCP as send. It sends a word and then waits for a response from the server. The connection is open until the client sends ‘bye’. Meanwhile, the server is always on.

2. In this task, using a UDP connection, multiple clients can connect to a single server, all peers can send and receive (server and clients). The communication begins with a client sending to the server then the server should reply.

   The snapshots below show a connection between a server and several clients (3 clients). As shown, all connections are clear on the server side, while the client can only see its own connection with the server.

Server's side:



```python
from socket import *

host = '192.168.1.115'
port = 446
server_socket = socket(AF_INET, SOCK_DGRAM)
server_socket.bind((host, port))

i = 0
client_number = 0
while True:

    data, address = server_socket.recvfrom(1024)
    client_number += 1
    if not data:
        break
    print(f"Message from client {client_number} : " + data.decode())
    print(f"Enter your message to client {client_number}: ")
    Tosend = input()

    server_socket.sendto(Tosend.encode(), address)
```

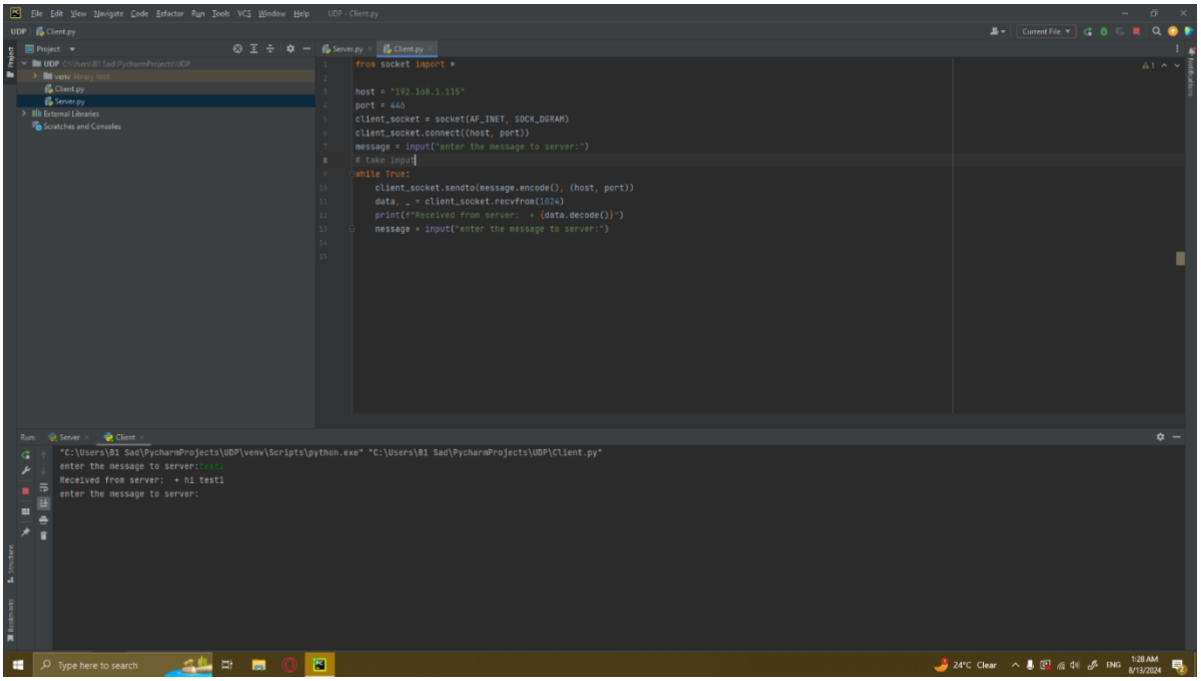
```
"C:\Users\B1 Sad\PycharmProjects\UDP\venv\Scripts\python.exe" "C:\Users\B1 Sad\PycharmProjects\UDP\Server.py"
Message from client 1 : test1
Enter your message to client 1:
hi test1
Message from client 2 : test2
Enter your message to client 2:
hi test2
Message from client 3 : test3
Enter your message to client 3:
hi test3
```

Figure 6.12: Python code for UDP server hosting

It connects to port 446, and uses some UDP functions such as “recvFrom”. The server receives messages from clients and responds freely (as a peer-to-peer approach). The server prints all incoming messages from the clients as well as its responses.

Two clients' sides

Client #1

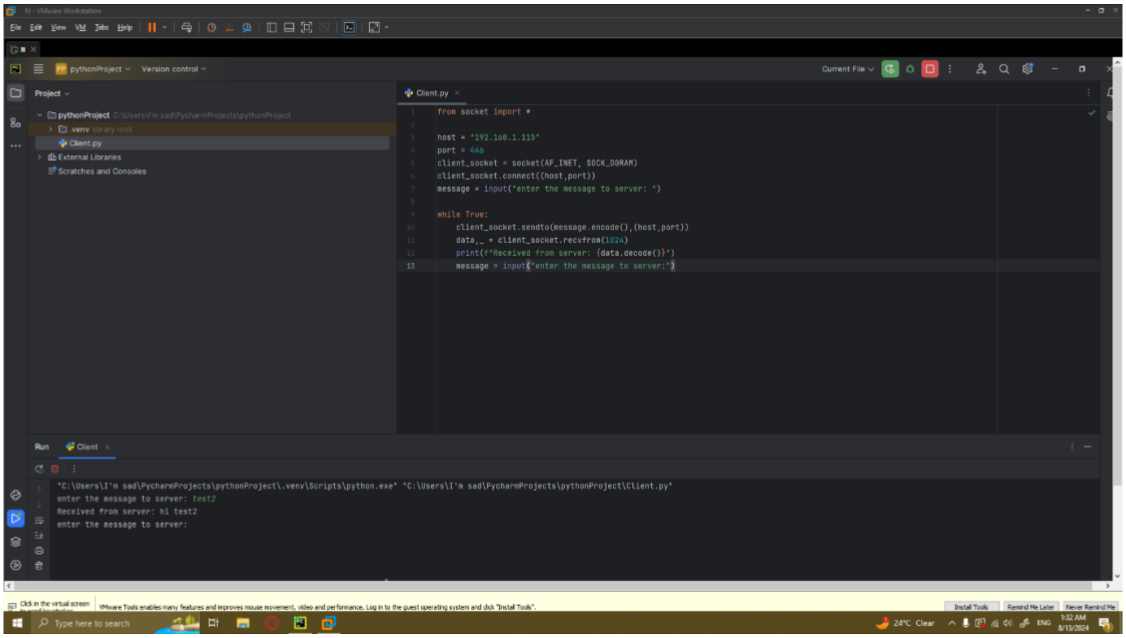


```
from socket import *

host = "192.168.1.115"
port = 446
client_socket = socket(AF_INET, SOCK_DGRAM)
client_socket.connect((host, port))
message = input("enter the message to server:")
# take input
while True:
    client_socket.sendto(message.encode(), (host, port))
    data, _ = client_socket.recvfrom(1024)
    print(f"Received from server:  + {data.decode()}")
    message = input("enter the message to server:")
```

```
"C:\Users\B1 Sad\PycharmProjects\UDP\venv\Scripts\python.exe" "C:\Users\B1 Sad\PycharmProjects\UDP\Client.py"
enter the message to server:test1
Received from server:  + hi test1
enter the message to server:
```

Figure 6.13: Client #1 console

Client #2



```
from socket import *

host = "192.168.1.115"
port = 446
client_socket = socket(AF_INET, SOCK_DGRAM)
client_socket.connect((host,port))
message = input("enter the message to server: ")

while True:
    client_socket.sendto(message.encode(),(host,port))
    data,_ = client_socket.recvfrom(1024)
    print(f"Received from server: {data.decode()}")
    message = input("enter the message to server:")
```

```
"C:\Users\I'm sad\PycharmProjects\pythonProject\.venv\Scripts\python.exe" "C:\Users\I'm sad\PycharmProjects\pythonProject\Client.py"
enter the message to server: test2
Received from server: hi test2
enter the message to server:
```

VMware Tools enables many features and improves mouse movement, video and performance. Log in to the guest operating system and click "Install Tools".

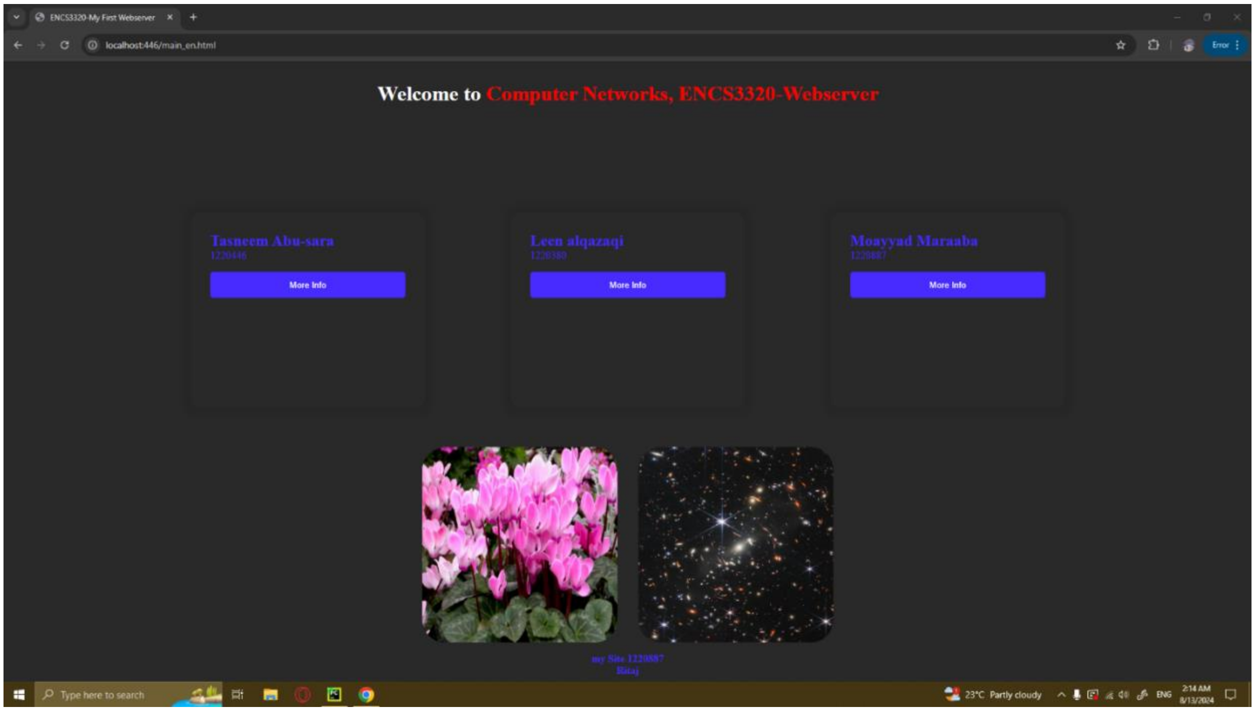
Figure 6.14: Client #2 console

Codes are shown in the snapshots above. The connection is always on. Also, the client uses some specified functions for UDP, such as sendto().

Task 3: web server

In this task, we built a web server and a web page. The web page contains several objects each of which can be stored on different web servers (they are on the same server in our case though). The web browser makes a request using the URL for the wanted object, and the server responds with the requested file or data.

Here is a snapshot of the main page of the web page. The HTML web page (the base file) being in the main_en.html file is noteworthy. There is a link to a local html file is in the main page (labeled as my site 1220887). A link to Ritaj is also provided at the bottom of the page. In addition to our names, numbers, and hobbies when clicking "More Info".



Welcome to Computer Networks, ENCS3320-Webserver

Tasneem Abu-sara
1220446
More Info

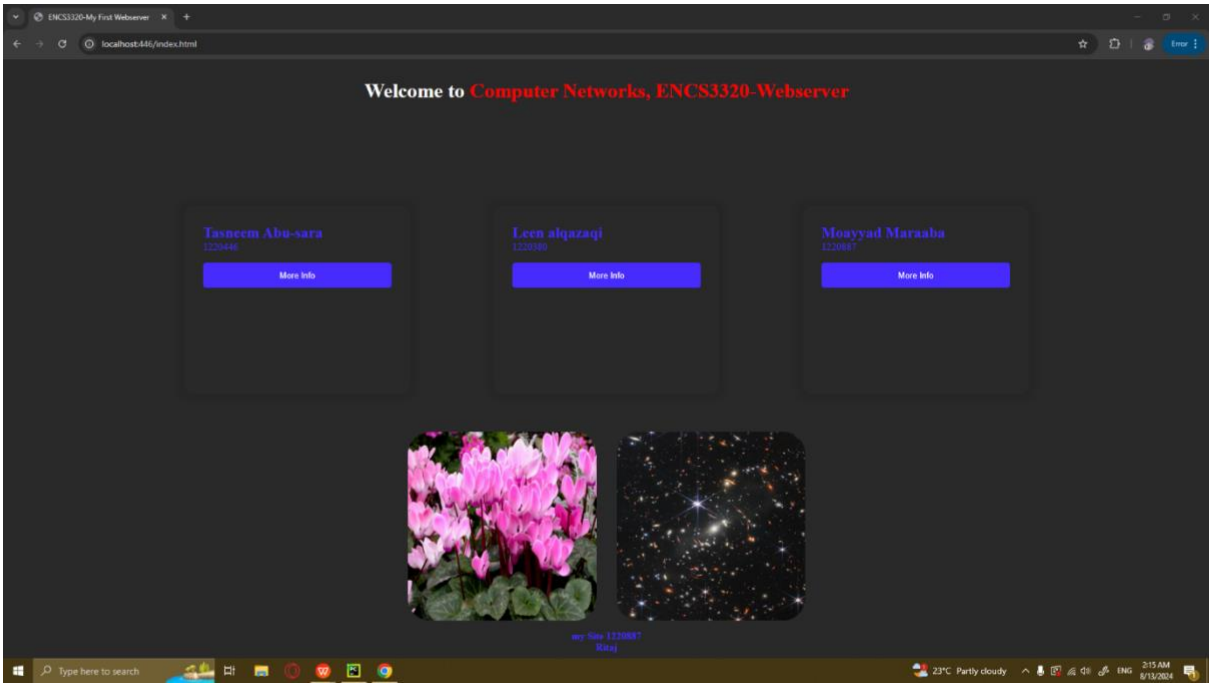
Leen alqazaqi
1220380
More Info

Moayyad Maraaba
1220887
More Info

my Site 1220887
Ritaj

Figure 6.15: webpage design

As requested, using main_en.html or index.html will open the file in main_en.html



Welcome to Computer Networks, ENCS3320-Webserver

Tasneem Abu-sara
1220446

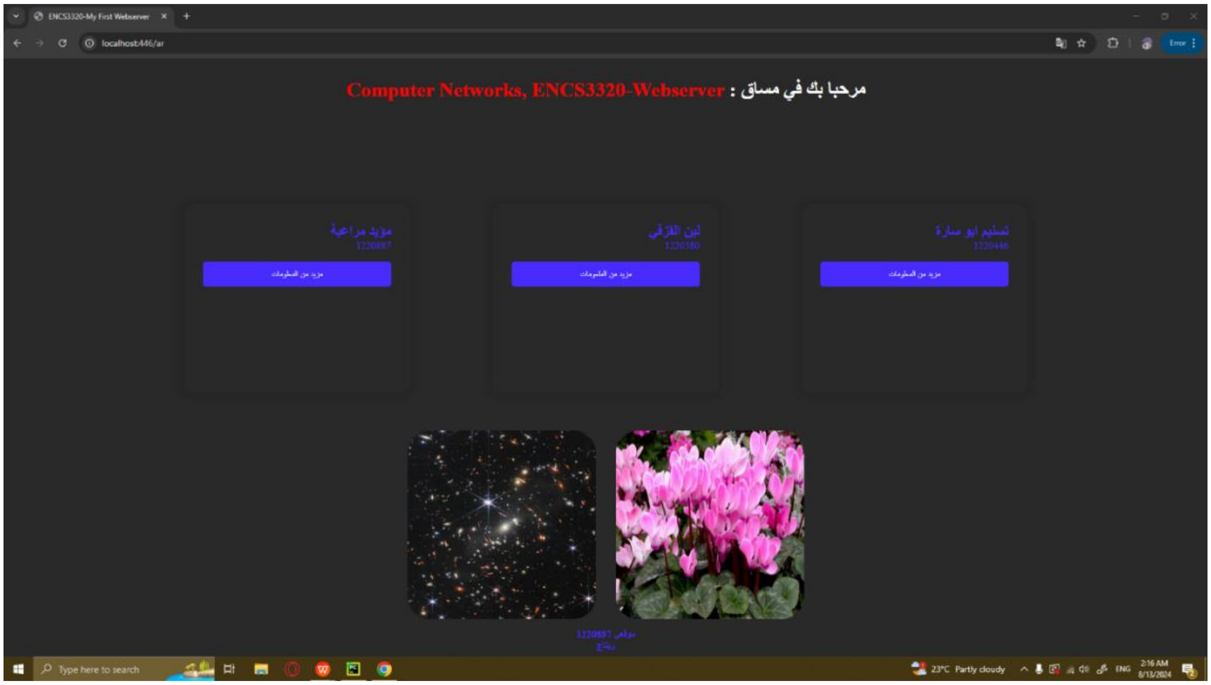
Leen alqazaqi
1220380

Moayyad Maraaba
1220887

my Site 1220887
Ritaj

Figure 6.16: /main_en.html request

When the request is /ar then the server will respond with main_ar.html as shown below which is the Arabic version of main_en.html.



Computer Networks, ENCS3320-Webserver : مرحبا بك في مساق

مويد مراعبة
1220887

لين القزقي
1220380

تسنيم ابو سارة
1220446

Figure 6.17: /ar request

If the request is .jpg the server sends the jpg image with content type “image /jpg”.

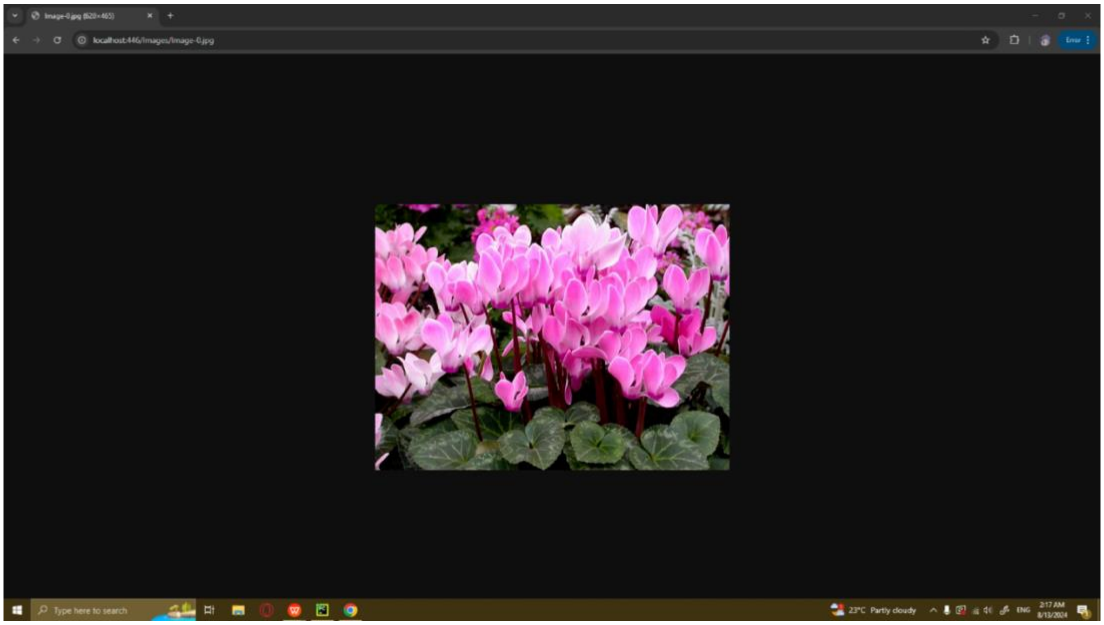


Figure 6.18: /image.jpg request

If the request is .png the server sends the png image with the content type “image /png”.

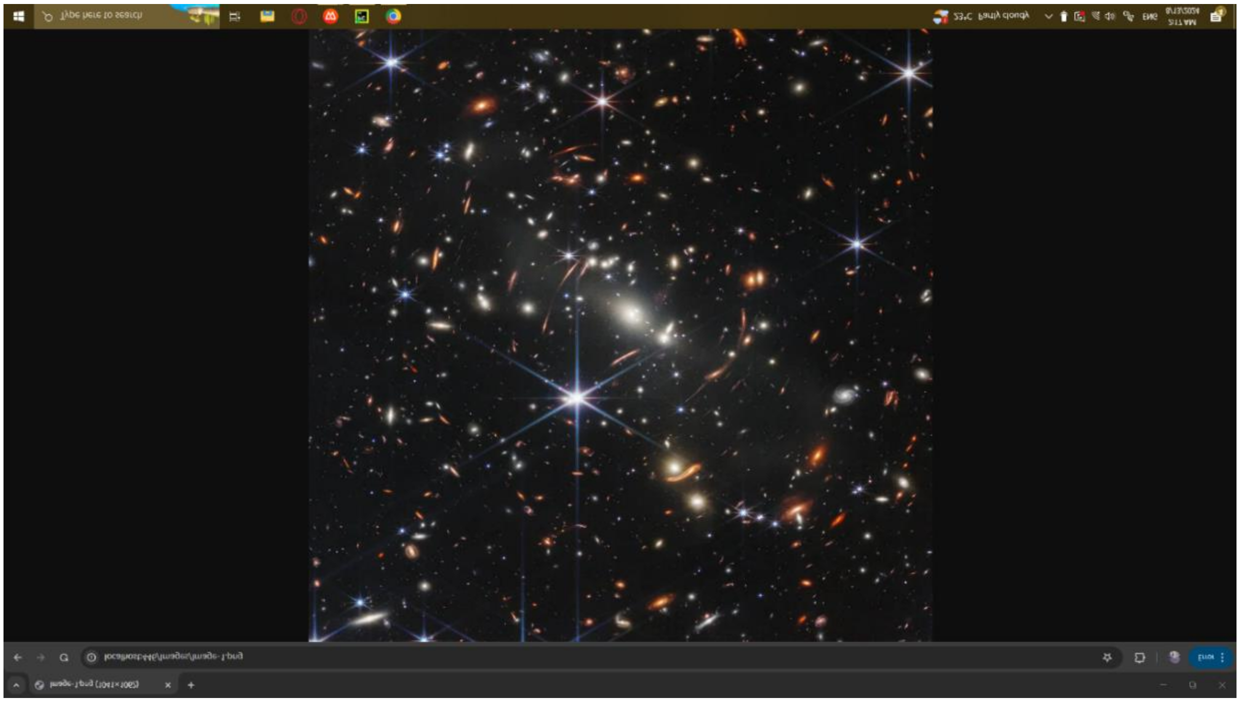
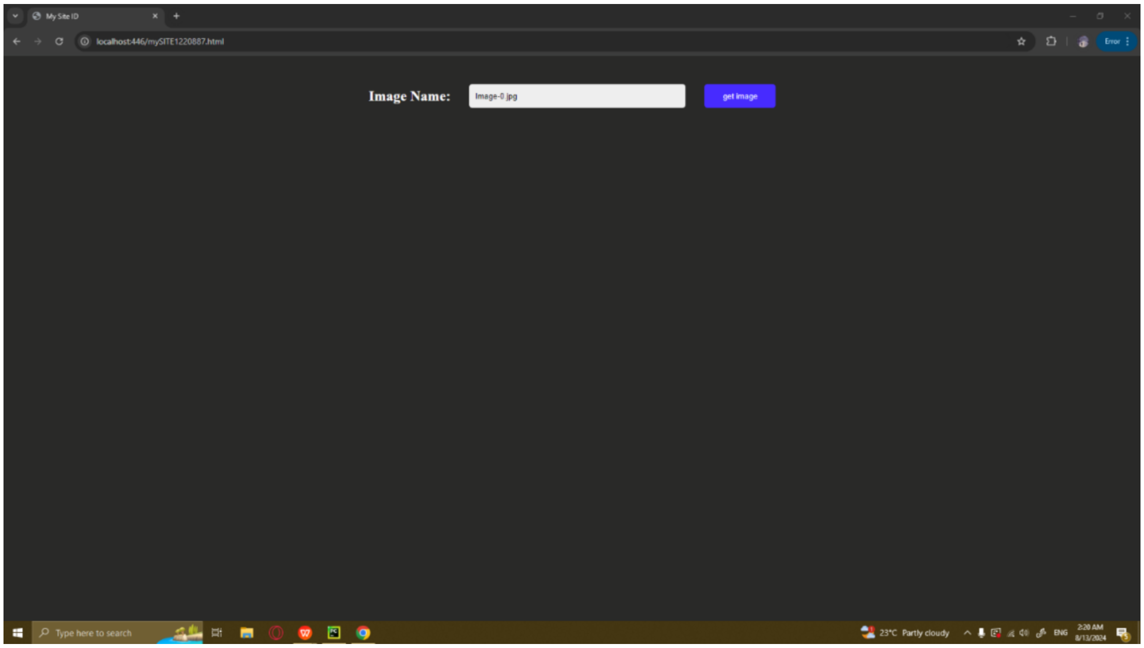


Figure 6.19: /image.png request

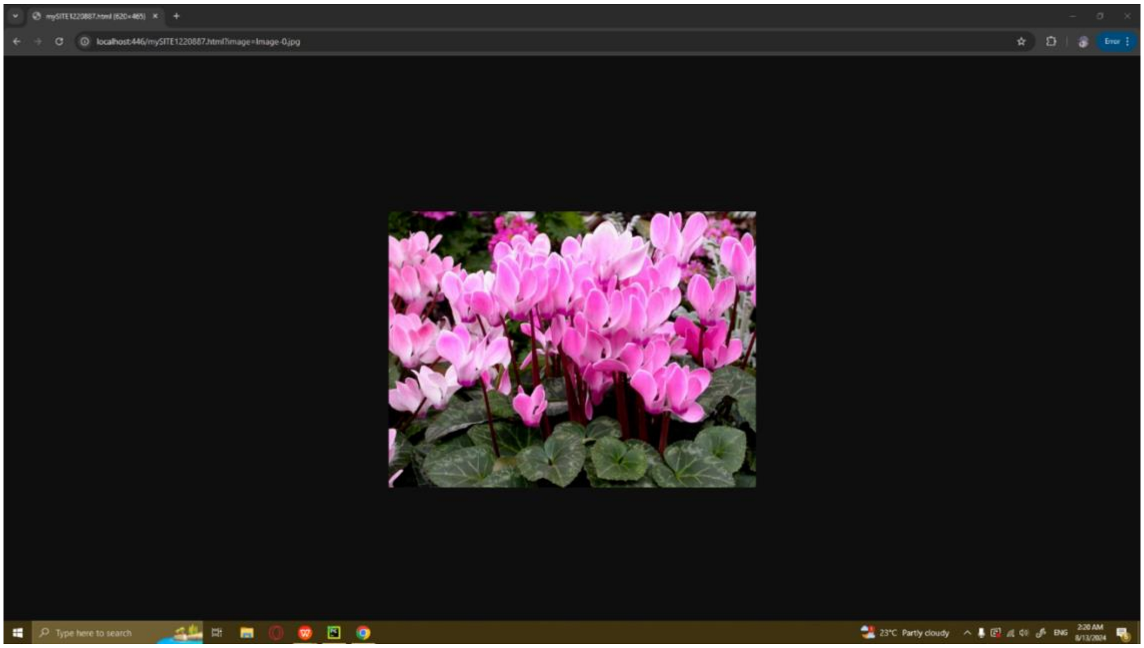
Additionally, we can use mySite1220887.html, to search for an image by typing its name in the box and the server will look for it in the images folder on the server.



localhost:446/mySITE1220887.html

Image Name: Image-0.jpg

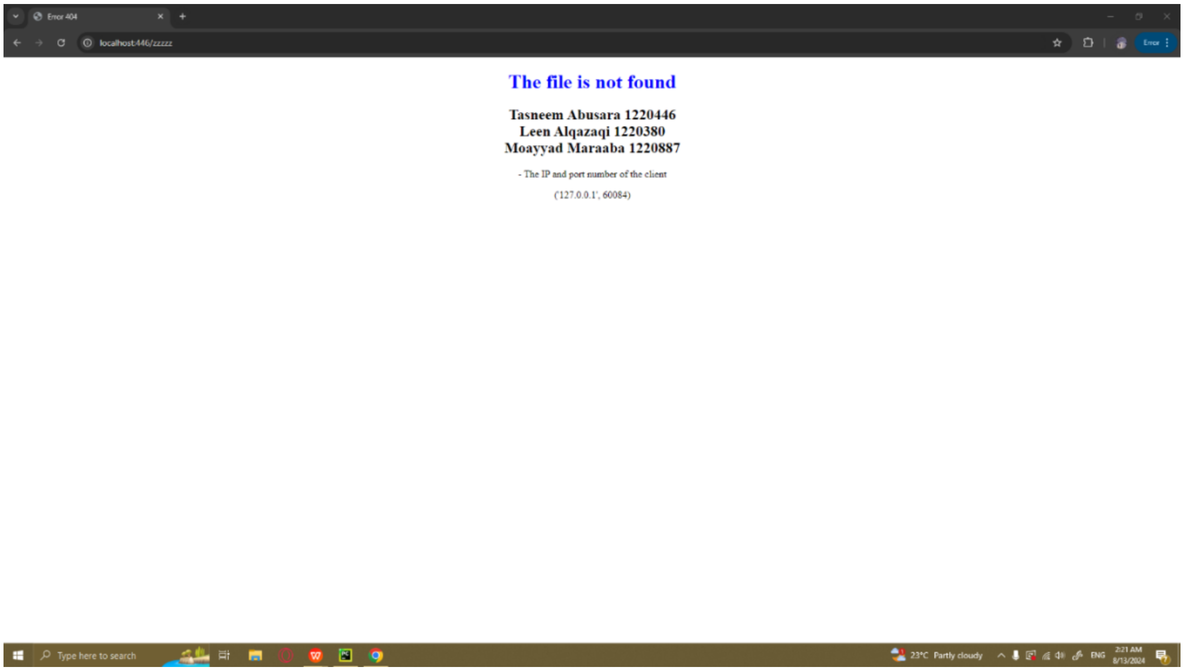
Figure 6.20: mySite1220887.html



localhost:446/mySITE1220887.html?image=Image-0.jpg

Figure 6.21: display the requested image

When the request is wrong or the file does not exist, the webpage will appear as in the snapshot below.



The file is not found

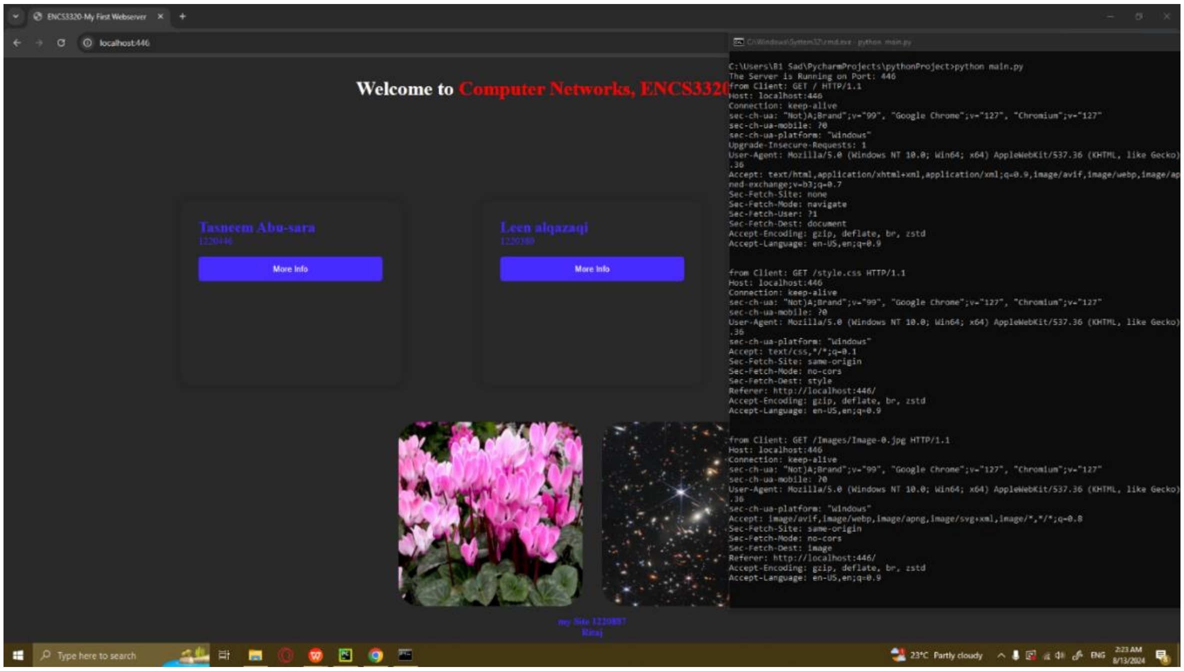
Tasneem Abusara 1220446
Leen Alqazaqi 1220380
Moayyad Maraaba 1220887

- The IP and port number of the client

('127.0.0.1', 60084)

Figure 6.22: notfound.html

The snapshot below shows the HTTP requests on the terminal window



Welcome to Computer Networks, ENCS3320

Tasneem Abu-sara
1220446

Leen alqazaqi
1220380

my Site 1220887
Ritaj

```
C:\Users\B1 Sad\PycharmProjects\pythonProject>python main.py
The Server is Running on Port: 446
from Client: GET / HTTP/1.1
Host: localhost:446
Connection: keep-alive
sec-ch-ua: "Not)A;Brand";v="99", "Google Chrome";v="127", "Chromium";v="127"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/ap
ned-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9


from Client: GET /style.css HTTP/1.1
Host: localhost:446
Connection: keep-alive
sec-ch-ua: "Not)A;Brand";v="99", "Google Chrome";v="127", "Chromium";v="127"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
.36
sec-ch-ua-platform: "Windows"
Accept: text/css,*/*;q=0.1
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: style
Referer: http://localhost:446/
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9


from Client: GET /Images/Image-0.jpg HTTP/1.1
Host: localhost:446
Connection: keep-alive
sec-ch-ua: "Not)A;Brand";v="99", "Google Chrome";v="127", "Chromium";v="127"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
.36
sec-ch-ua-platform: "Windows"
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: http://localhost:446/
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
```

Figure 6.23: HTTP requests on the terminal window

If the request is /so then it redirects to stackoverflow.com website. Similarly, when the request is /itc, the localhost will redirect to itc.birzeit.edu.

After creating a firewall rule to allow connection through the port ‘446’ and changing the IP of the hosting server to 0.0.0.0, we can connect from other devices on the same network as below

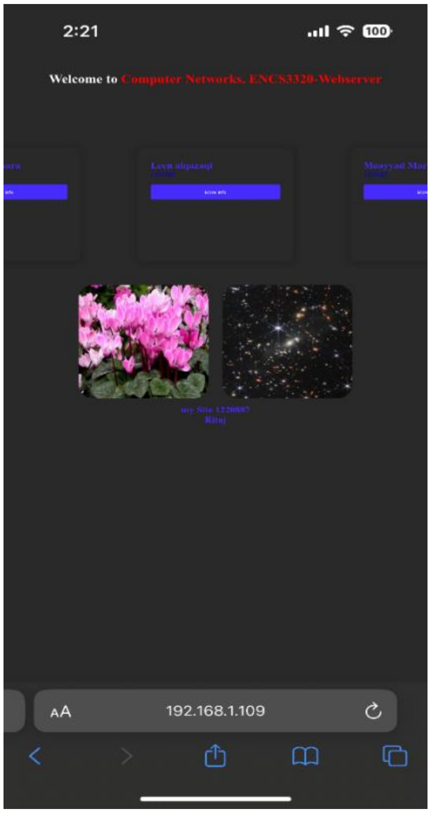
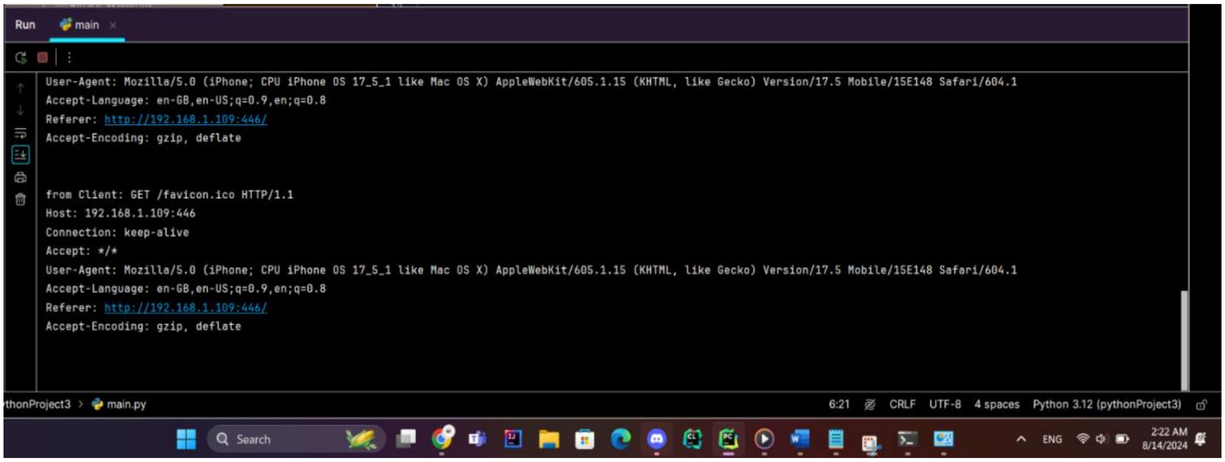


Figure 6.24: server accessed from a device on the same network



```
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 17_5_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/17.5 Mobile/15E148 Safari/604.1
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Referer: http://192.168.1.109:446/
Accept-Encoding: gzip, deflate


from Client: GET /favicon.ico HTTP/1.1
Host: 192.168.1.109:446
Connection: keep-alive
Accept: */*
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 17_5_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/17.5 Mobile/15E148 Safari/604.1
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Referer: http://192.168.1.109:446/
Accept-Encoding: gzip, deflate
```

Figure 6.25: Request from mentioned device

7 Alternatives Solutions, Issues and Limitations

Task1:

Issues:

- Telnet is not allowed on all servers or routers, and in our case, it did not connect. If it had connected as expected, we would have had the ability to send HTTP requests directly.

- In general, some security protocols limit our freedom on the internet and limit the use of some commands.

Task2:

Alternative solutions:

- We can write the code in multiple ways using different functions.

Issues:

- No fixed IP for the server so the code is not general.

Task3:

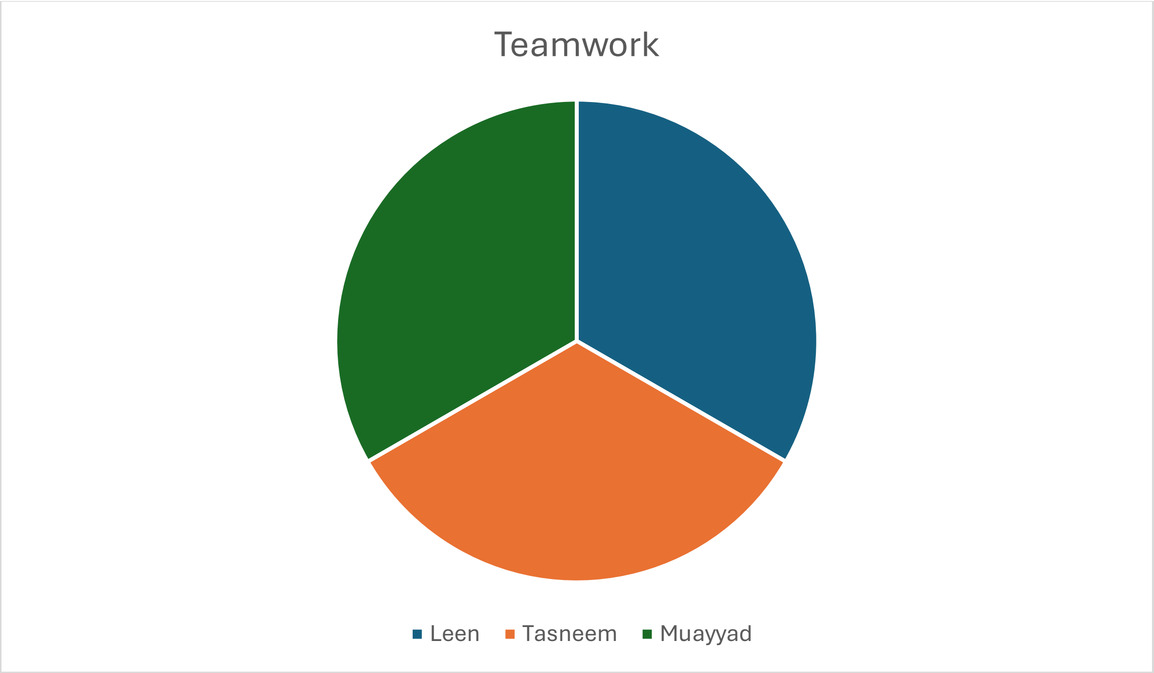
Alternative solutions:

- We could have placed the objects on different web servers; however, it would have resulted in extra RTTs

Issues:

- Servers usually have fixed IPs, but in our case, it is dynamic since the server is on an ordinary laptop. As a result, the code is not general and should be updated whenever the IP or device changes. This problem can be fixed by requesting a fixed IP.
- While making the server we experienced an issue with using the same port multiple times while hosting the server. The solution to this issue was to make sure to close the connection when the process was done.

8 Teamwork



Leen:

- Task 1.
- Parts of task 2 A.
- Task 2 B.
- Major parts of the report.

Muayyad:

- Parts of Task 2 A.
- Parts of task 2 B.
- Frontend of task 3.
- Parts of the report.

Tasneem:

- Parts of Task 2 A.
- Majority of the Python programming for task 3.
- Parts of the report.

9 References

[1] colocation America: https://www.colocationamerica.com/blog/tcp-ip-vs-udp

[2] bgp tool: https://bgp.tools/