BIRZEIT UNIVERSITY

Faculty of Engineering & Technology

Department of Electrical and Computer Engineering

Computer Network

ENCS3320

# Project Report

---

Prepared by:

| | |
|---|---|
| Riwaa Assi | 1201419 |
| Nidal Zabade | 1200153 |
| Dana Imam | 1200121 |

Section: 1

Instructor: Dr. Mohammed Helal

Date: 26/8/2022

## Abstract

The purpose of this Project was to developing a UDP-based chatting application based on a peer-to-peer architecture, we needed a server in order to provide connectivity between the Clients.

# Table of Contents

## Theory

We want to make communication between the server and sender using the protocol UDP and the peer to peer method.

### Server

A server is a computer program or device that provides a service to another computer program and its user, who is known as the client. In a data center, the physical computer that a server program runs on is also frequently referred to as a server.

Here are some properties of the server:

▪ always-on host

▪ permanent IP address

▪ often in data centers, for scaling

### Client

A client is any computer hardware or software device that requests access to a service provided by a server. Clients are typically seen as the requesting program or user in a client-server architecture. There are many properties of the clients, here are the most important ones:

▪ contact, communicate with server

▪ may be intermittently connected

▪ may have dynamic IP addresses

▪ do not communicate directly with each other

▪ examples: HTTP, IMAP, FTP

### UDP Protocol

It is the abbreviation of User Datagram Protocol. It is used in the following:

- streaming multimedia apps (loss tolerant, rate sensitive)
- DNS
- SNMP
- HTTP/3

It is also known as "no frills protocol" for the following reasons:

- segments may be lost, delivered out of order

- best effort service: “send and hope for the best”

Also, UDP has its plusses:

- no setup/handshaking needed (no RTT incurred)
- can function when network service is compromised
- helps with reliability (checksum)
- build additional functionality on top of UDP in application layer (e.g., HTTP/3).

## UDP Format

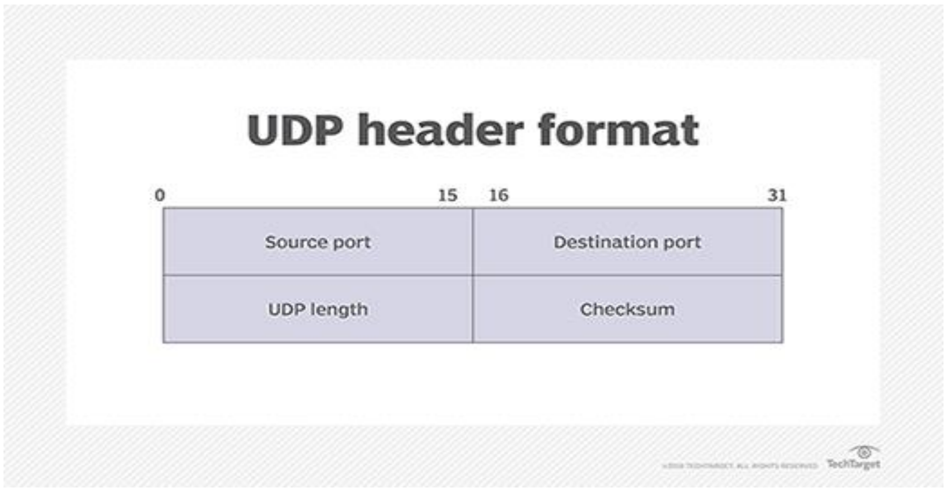


*Figure 1: UDP Format*

## UDP Protocol

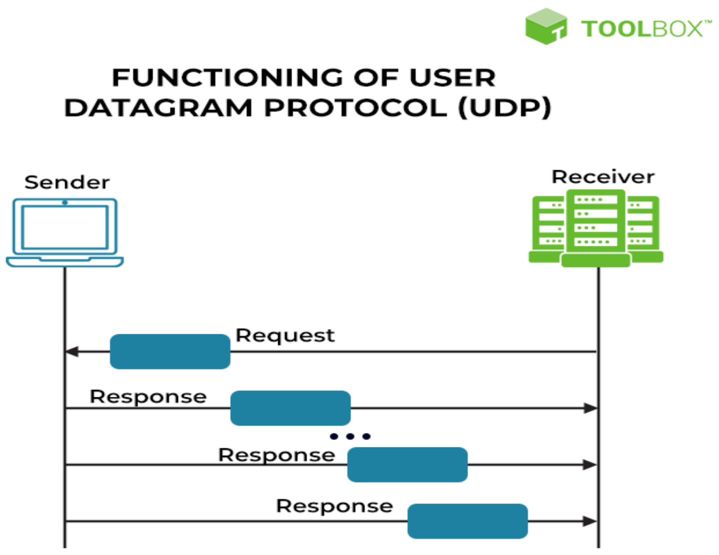


*Figure 2: UDP Protocol*

## Peer to peer architecture

It can be simply defined as nodes that are both clients and servers. P2P networks distribute the workload between peers, and all peers contribute and consume resources within the network without the need for a centralized server.

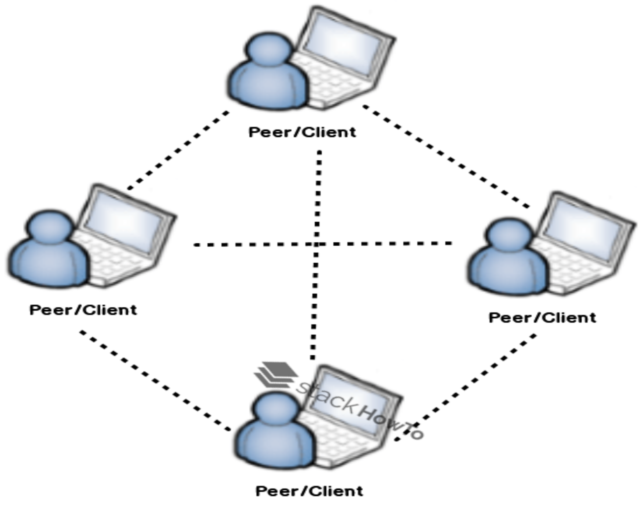


*Figure 3:peer to peer architecture*

## What is required from the project?

Briefly, the project requires creating a chatting application based on peer to peer architecture. First of all, a server is basically created, and has many things to do: it has to save a list of renewing clients, in other words, this list will be updated whenever there is a new online client. Furthermore, this list should also include the required information for connection about each client, which are the user ID, the IP address and port number.  This process will be done whenever the client is active, by sending its ID to the server in UDP datagram, and automatically the server saves the previously mentioned information about the client. In addition, each client will receive the ID of each another new client in a renewing list, which contains only IDs of another clients, and this step fundamentally aims to give the ability for the clients to contact with each other, by sending the ID of the target client to the server, and in return, the server responds to the sender, using UDP message that containing the IP address and the port number of the target client. Finally, since the sender client has information about the target client, he/she can send a UDP message to the latter.

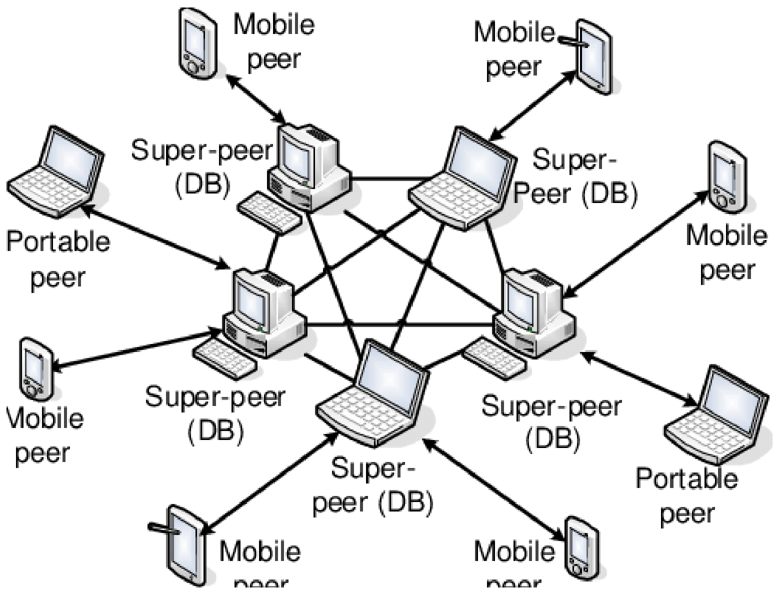
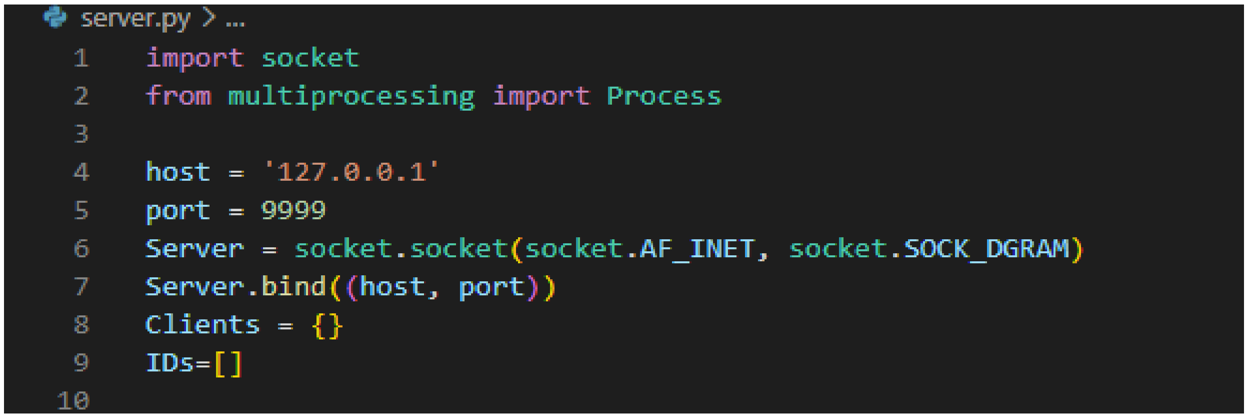


*Figure 4:A simple explanation of the project*

## Procedure

### Server

We defined the server through the port and the host number and use a dictionary of clients that they want to communicate with each other.



server.py > ...

```python
import socket
from multiprocessing import Process

host = '127.0.0.1'
port = 9999
Server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
Server.bind((host, port))
Clients = {}
IDs=[]

```

*Figure 5: server1*

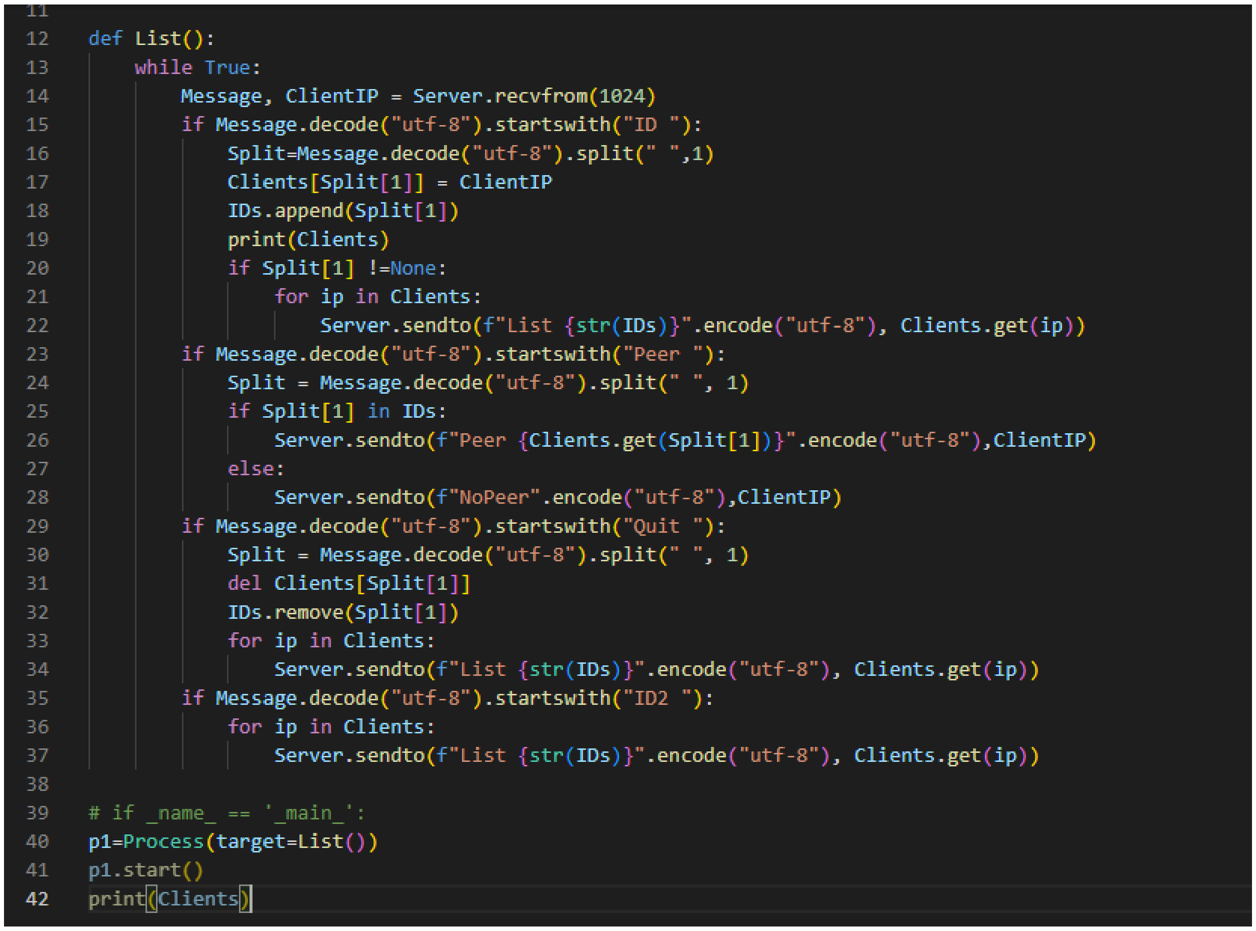
Notice that we used the local host for the server, and the client. Both of the host and port number represent the IP for the server.

Furthermore, (SOCK_DGRAM) indicates to the UDP protocol, which is the required protocol in the project.

In addition, server IP is stable and cannot be changed, if more than server is needed, we only change the port number, but in this project, only one server is needed.

When the server receives a message from the client, which contains a header, content and IP of the client, it can detect what is the required process, by decoding the header of the message:

- If the header was ID, it means that a new client is active, so the server adds this client to the Clients dictionary.
- If the header was Quit, the server deletes the client from the Clients dictionary, and delete his/her ID from the IDs array.
- If the header was Peer, the server check if the content of this message, which is supposed to be an ID number, exists in the IDs array. If it is, then the target client exists, so the server sends the IP of the target client to the sender client.

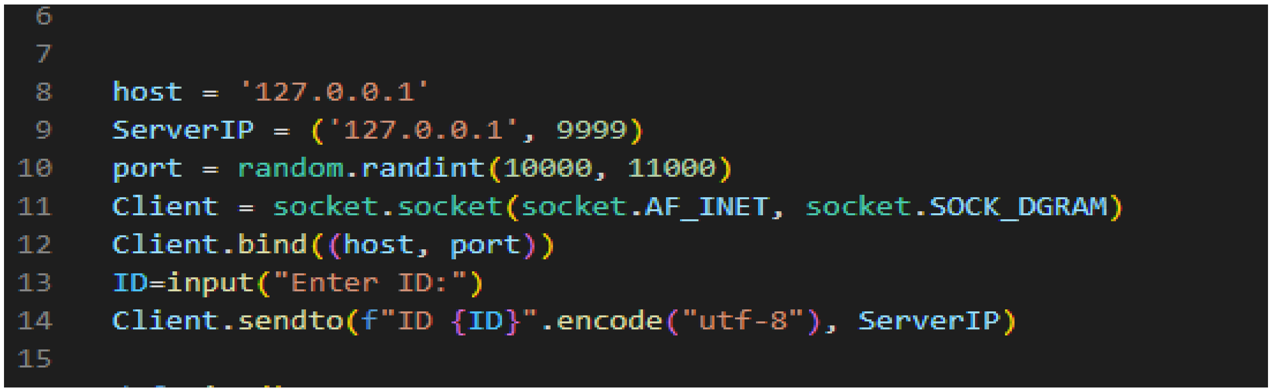


```python
def List():
    while True:
        Message, ClientIP = Server.recvfrom(1024)
        if Message.decode("utf-8").startswith("ID "):
            Split=Message.decode("utf-8").split(" ",1)
            Clients[Split[1]] = ClientIP
            IDs.append(Split[1])
            print(Clients)
            if Split[1] !=None:
                for ip in Clients:
                    Server.sendto(f"List {str(IDs)}".encode("utf-8"), Clients.get(ip))
        if Message.decode("utf-8").startswith("Peer "):
            Split = Message.decode("utf-8").split(" ", 1)
            if Split[1] in IDs:
                Server.sendto(f"Peer {Clients.get(Split[1])}".encode("utf-8"),ClientIP)
            else:
                Server.sendto(f"NoPeer".encode("utf-8"),ClientIP)
        if Message.decode("utf-8").startswith("Quit "):
            Split = Message.decode("utf-8").split(" ", 1)
            del Clients[Split[1]]
            IDs.remove(Split[1])
            for ip in Clients:
                Server.sendto(f"List {str(IDs)}".encode("utf-8"), Clients.get(ip))
        if Message.decode("utf-8").startswith("ID2 "):
            for ip in Clients:
                Server.sendto(f"List {str(IDs)}".encode("utf-8"), Clients.get(ip))

# if _name_ == '_main_':
p1=Process(target=List())
p1.start()
print(Clients)
```

*Figure 6: server2*

## Client

Here, each client will be identified by a randomly selected port number and use the IP server so that there can be communication between the client and the server.



```python
host = '127.0.0.1'
ServerIP = ('127.0.0.1', 9999)
port = random.randint(10000, 11000)
Client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
Client.bind((host, port))
ID=input("Enter ID:")
Client.sendto(f"ID {ID}".encode("utf-8"), ServerIP)
```

*Figure 7: client*

As we mentioned before, the host and port here represent the IP of the client.

In line 14, a new client is active, and sends his/her ID to the server, and implicitly his/her IP.

When the client receives a message from the server, the header of this message also indicates to special process, as it was mentioned in the server process, so:

- If the header was List, this means that the message contains a list of all active clients' IDs.

  Here, the client will be offered if he/she wants to send a message to another client - for sure by peer to peer protocol -:

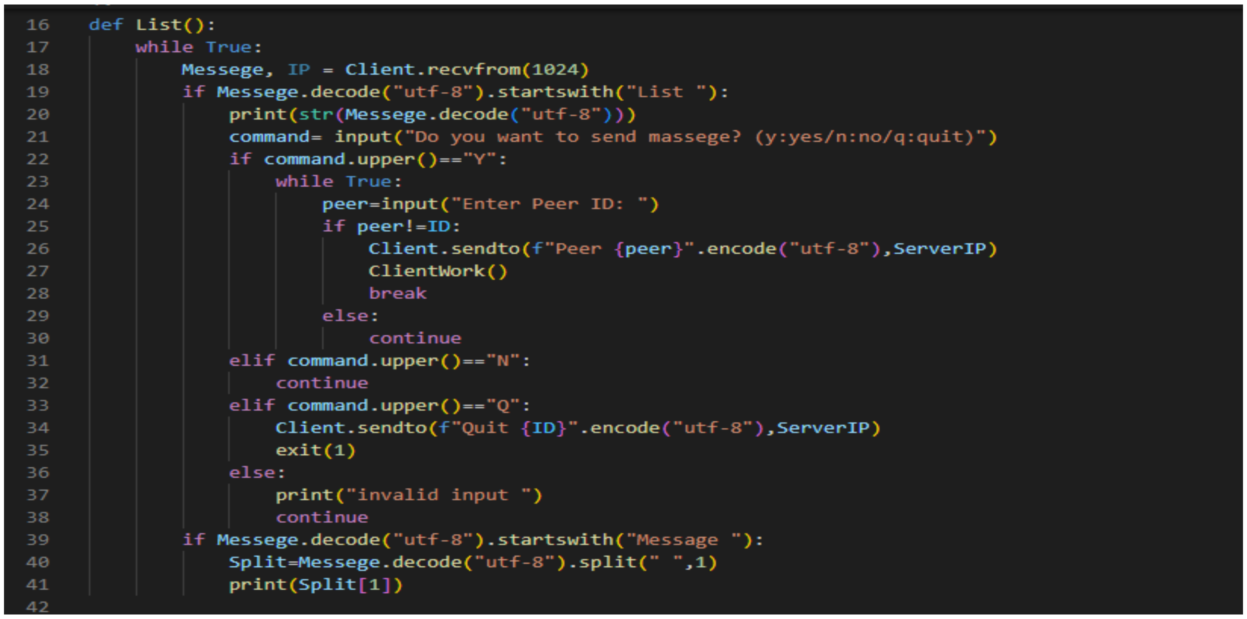
  - If the client accepts, then he/she enters the ID of the target client, and send a message with (Peer) header to the server, to check if this target client is active and exists.

    Notice that our program makes sure that the sender does not enter his/her ID when he/she wants to send a message.

    So, when the server responds back with the IP of the target client – if exist-, the sender could write a message to the target client by peer to peer protocol.

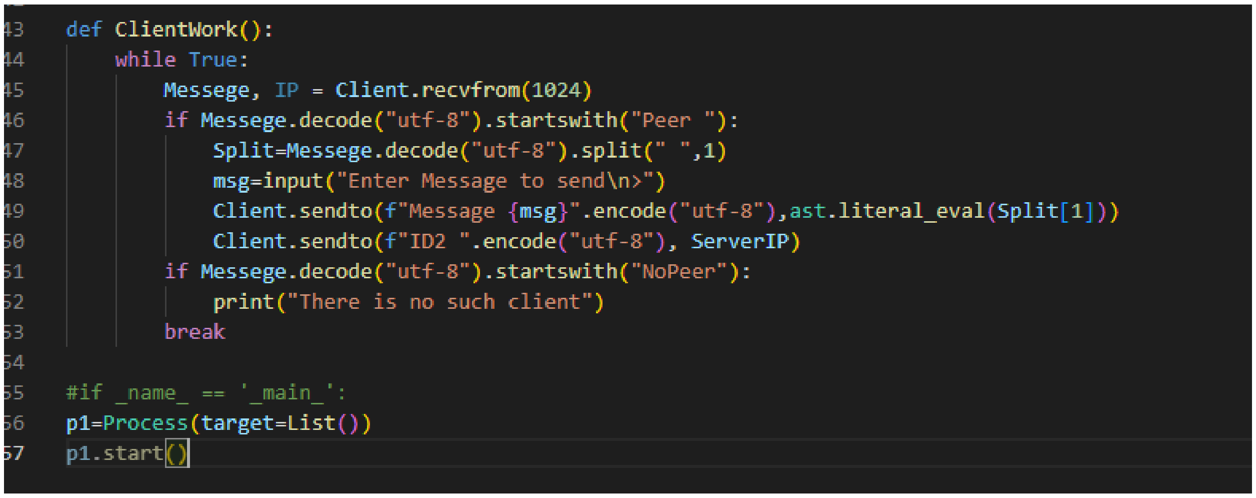
  - If the client denied to send ant message, the program will continue.
  - If the client wants to quit the chatting, the client will be deleted, and an exit command will be executed for this client.

- If the header was Message, this means that it is a message from another client (sender), and it will be shown to the target client.



```python
def List():
    while True:
        Messege, IP = Client.recvfrom(1024)
        if Messege.decode("utf-8").startswith("List "):
            print(str(Messege.decode("utf-8")))
            command= input("Do you want to send massege? (y:yes/n:no/q:quit)")
            if command.upper()=="Y":
                while True:
                    peer=input("Enter Peer ID: ")
                    if peer!=ID:
                        Client.sendto(f"Peer {peer}".encode("utf-8"),ServerIP)
                        ClientWork()
                        break
                    else:
                        continue
            elif command.upper()=="N":
                continue
            elif command.upper()=="Q":
                Client.sendto(f"Quit {ID}".encode("utf-8"),ServerIP)
                exit(1)
            else:
                print("invalid input ")
                continue
        if Messege.decode("utf-8").startswith("Message "):
            Split=Messege.decode("utf-8").split(" ",1)
            print(Split[1])

```

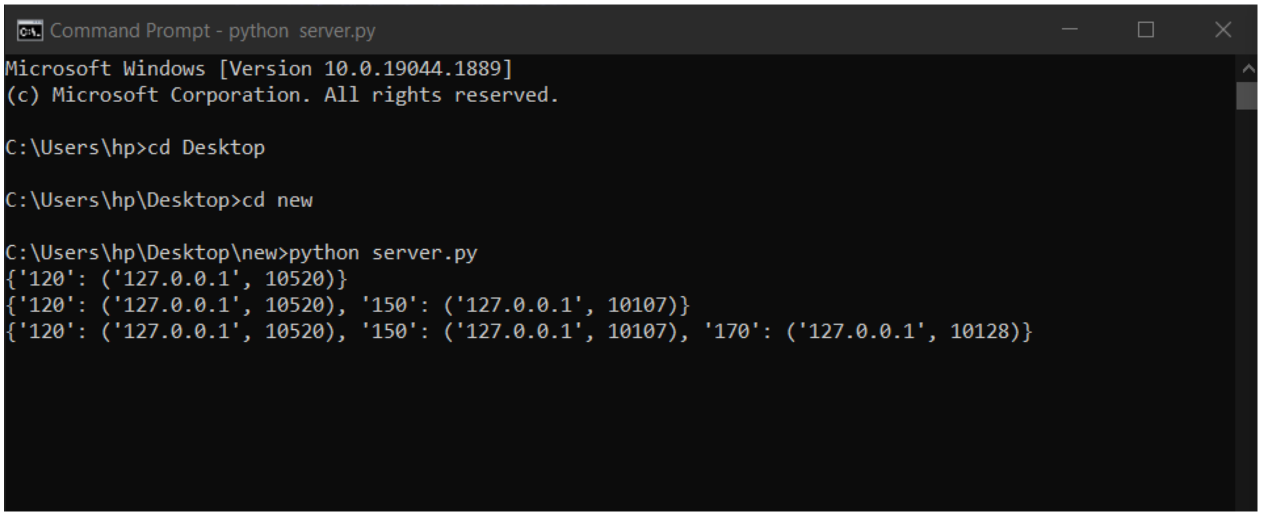
*Figure 8: client2*



```
def ClientWork():
    while True:
        Messege, IP = Client.recvfrom(1024)
        if Messege.decode("utf-8").startswith("Peer "):
            Split=Messege.decode("utf-8").split(" ",1)
            msg=input("Enter Message to send\n>")
            Client.sendto(f"Message {msg}".encode("utf-8"),ast.literal_eval(Split[1]))
            Client.sendto(f"ID2 ".encode("utf-8"), ServerIP)
        if Messege.decode("utf-8").startswith("NoPeer"):
            print("There is no such client")
        break

#if _name_ == '_main_':
p1=Process(target=List())
p1.start()
```

*Figure 9: client3*

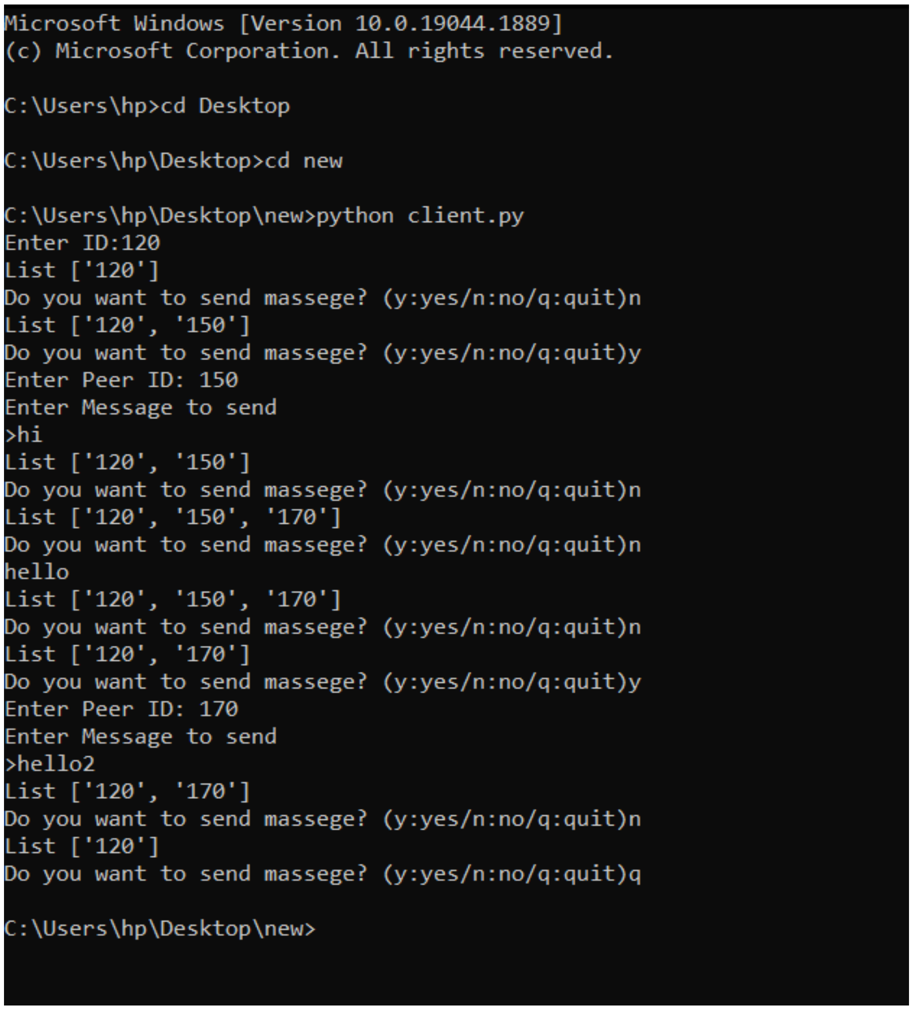
## The output results



Command Prompt - python server.py

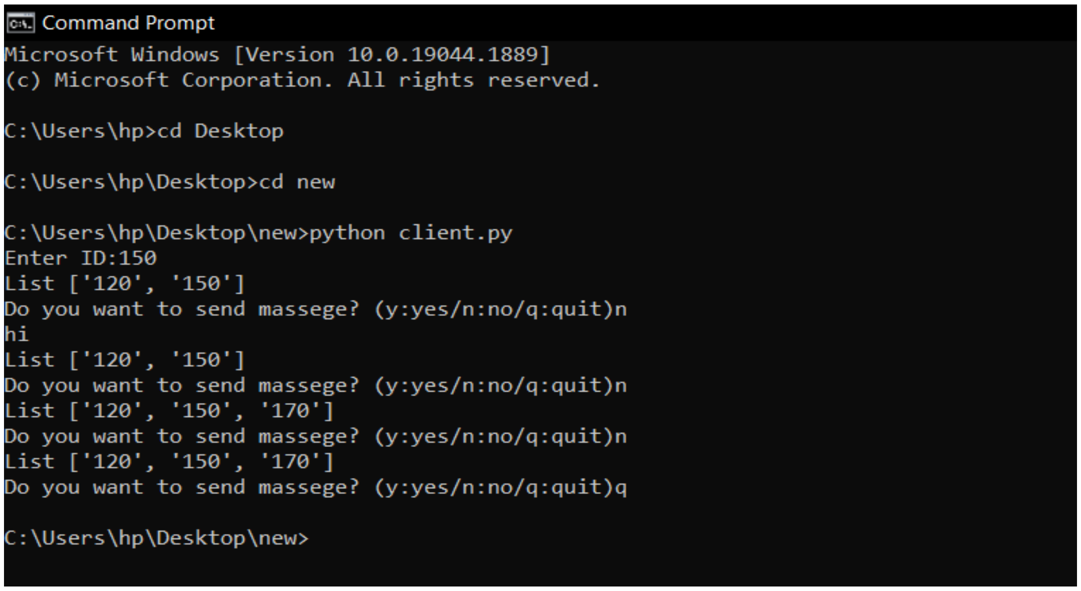
```
Microsoft Windows [Version 10.0.19044.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>cd Desktop

C:\Users\hp\Desktop>cd new

C:\Users\hp\Desktop\new>python server.py
{'120': ('127.0.0.1', 10520)}
{'120': ('127.0.0.1', 10520), '150': ('127.0.0.1', 10107)}
{'120': ('127.0.0.1', 10520), '150': ('127.0.0.1', 10107), '170': ('127.0.0.1', 10128)}
```

*Figure 10: Server*



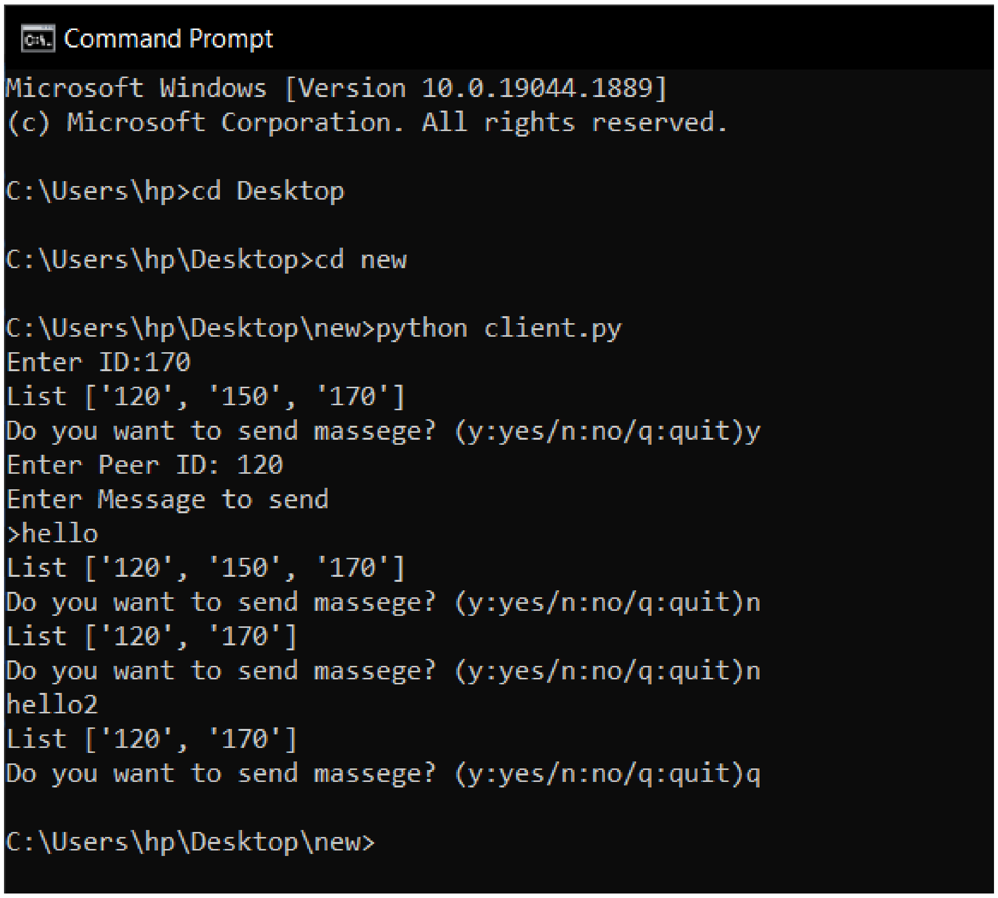
```
Microsoft Windows [Version 10.0.19044.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>cd Desktop

C:\Users\hp\Desktop>cd new

C:\Users\hp\Desktop\new>python client.py
Enter ID:120
List ['120']
Do you want to send massege? (y:yes/n:no/q:quit)n
List ['120', '150']
Do you want to send massege? (y:yes/n:no/q:quit)y
Enter Peer ID: 150
Enter Message to send
>hi
List ['120', '150']
Do you want to send massege? (y:yes/n:no/q:quit)n
List ['120', '150', '170']
Do you want to send massege? (y:yes/n:no/q:quit)n
hello
List ['120', '150', '170']
Do you want to send massege? (y:yes/n:no/q:quit)n
List ['120', '170']
Do you want to send massege? (y:yes/n:no/q:quit)y
Enter Peer ID: 170
Enter Message to send
>hello2
List ['120', '170']
Do you want to send massege? (y:yes/n:no/q:quit)n
List ['120']
Do you want to send massege? (y:yes/n:no/q:quit)q

C:\Users\hp\Desktop\new>
```

*Figure 11: Client 1*



```
Command Prompt
Microsoft Windows [Version 10.0.19044.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>cd Desktop

C:\Users\hp\Desktop>cd new

C:\Users\hp\Desktop\new>python client.py
Enter ID:150
List ['120', '150']
Do you want to send massege? (y:yes/n:no/q:quit)n
hi
List ['120', '150']
Do you want to send massege? (y:yes/n:no/q:quit)n
List ['120', '150', '170']
Do you want to send massege? (y:yes/n:no/q:quit)n
List ['120', '150', '170']
Do you want to send massege? (y:yes/n:no/q:quit)q

C:\Users\hp\Desktop\new>
```

*Figure 12: Client 2*



```
Command Prompt
Microsoft Windows [Version 10.0.19044.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>cd Desktop

C:\Users\hp\Desktop>cd new

C:\Users\hp\Desktop\new>python client.py
Enter ID:170
List ['120', '150', '170']
Do you want to send massege? (y:yes/n:no/q:quit)y
Enter Peer ID: 120
Enter Message to send
>hello
List ['120', '150', '170']
Do you want to send massege? (y:yes/n:no/q:quit)n
List ['120', '170']
Do you want to send massege? (y:yes/n:no/q:quit)n
hello2
List ['120', '170']
Do you want to send massege? (y:yes/n:no/q:quit)q

C:\Users\hp\Desktop\new>
```

*Figure 13: Client 3*

## Conclusion

From this project, we learned how to make an UDP chatting application based on peer to peer architecture. We also understand more about how the connection between them occurs by using these protocols.

## References

https://www.researchgate.net/profile/ErkkiHarjula/publication/224305552/figure/fig2/AS:667774380679177@1536221227291/Peer-to-peer-data-management-architecture.png

https://stackhowto.com/wp-content/uploads/2022/02/peer-to-peer-architecture-advantages-and-disadvantages-with-example-1.png

https://pimages.toolbox.com/wp-content/uploads/2022/04/14111046/105.png

https://www.techtarget.com/whatis/definition/server

https://www.techtarget.com/whatis/definition/client