Final Task- Communication between client and server

*** Written by Guy Azoulay and Batel Cohen**

main goal of the project-

**In this assignment we asked to build a communication app between multiply clients and one server,
we tried our best to achieve our main goal: an application with basic GUI which with it we could communicate each other with messages, private messages file download and upload files.
While exploring this field and programming it we achieve a good and deep understanding about all the communication world, how binding between different clients and server work,
what is happening in the background of it etc.
We won't lie, at first it was a little bit hard to understand how to make all of this work as we want, but giving up was not an option, so we sat and learned every part in this assignment not just for an A in our final grade, we also improved our knowledge in this field.**

In this assignment we implemented two main classes:

*** Server**

*** Client including a GUI**

Explanation about the Server class:

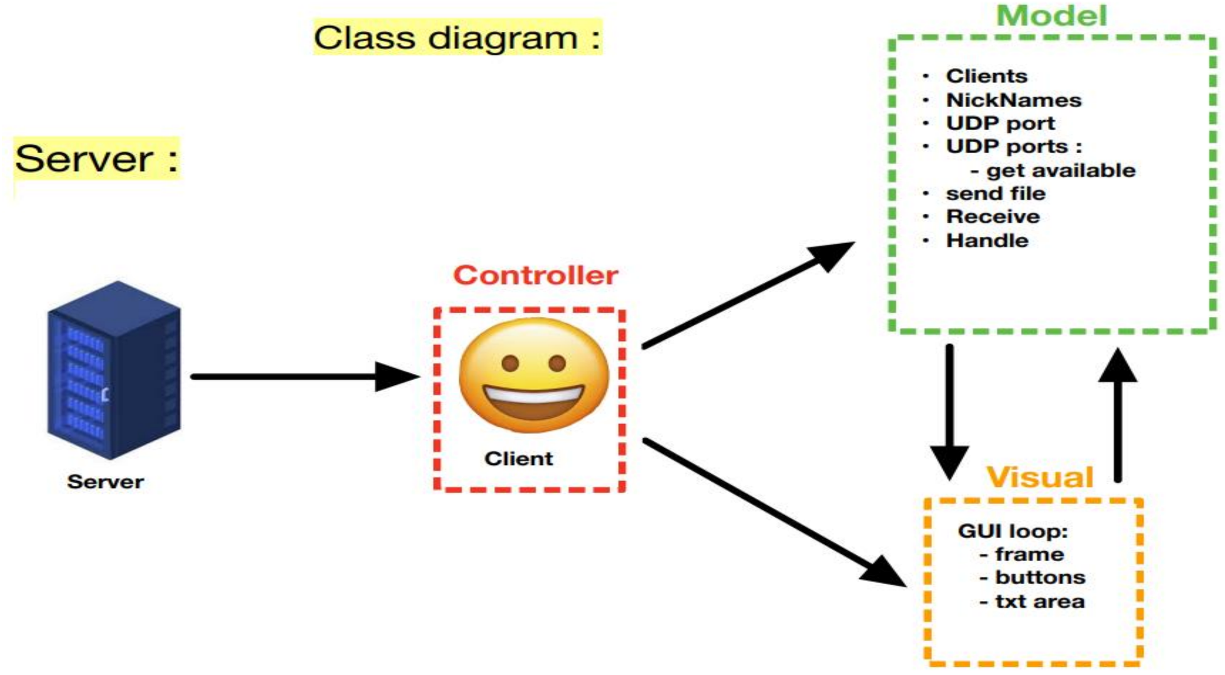
This class is the main class in our project, everything which send from any client is going through the server, and the server provide any action we would like to do.

Our main host IP is 127.0.0.1 and the port we work on is 50,000 as we asked for.

All the communication work via TCP connection, messages, private messages, online members etc.

In aim to transfer files we created another socket which work via UDP connection.

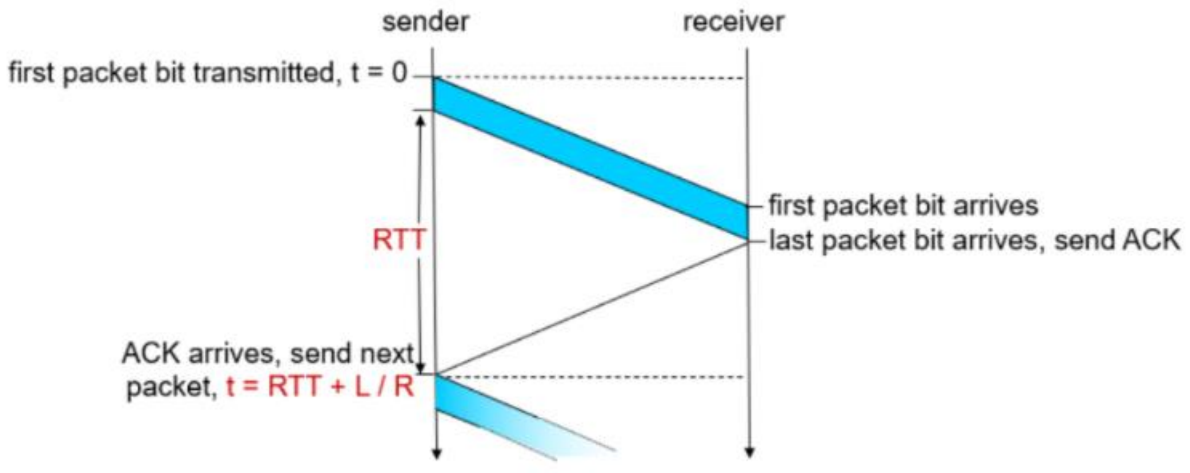
MVC Server diagram:



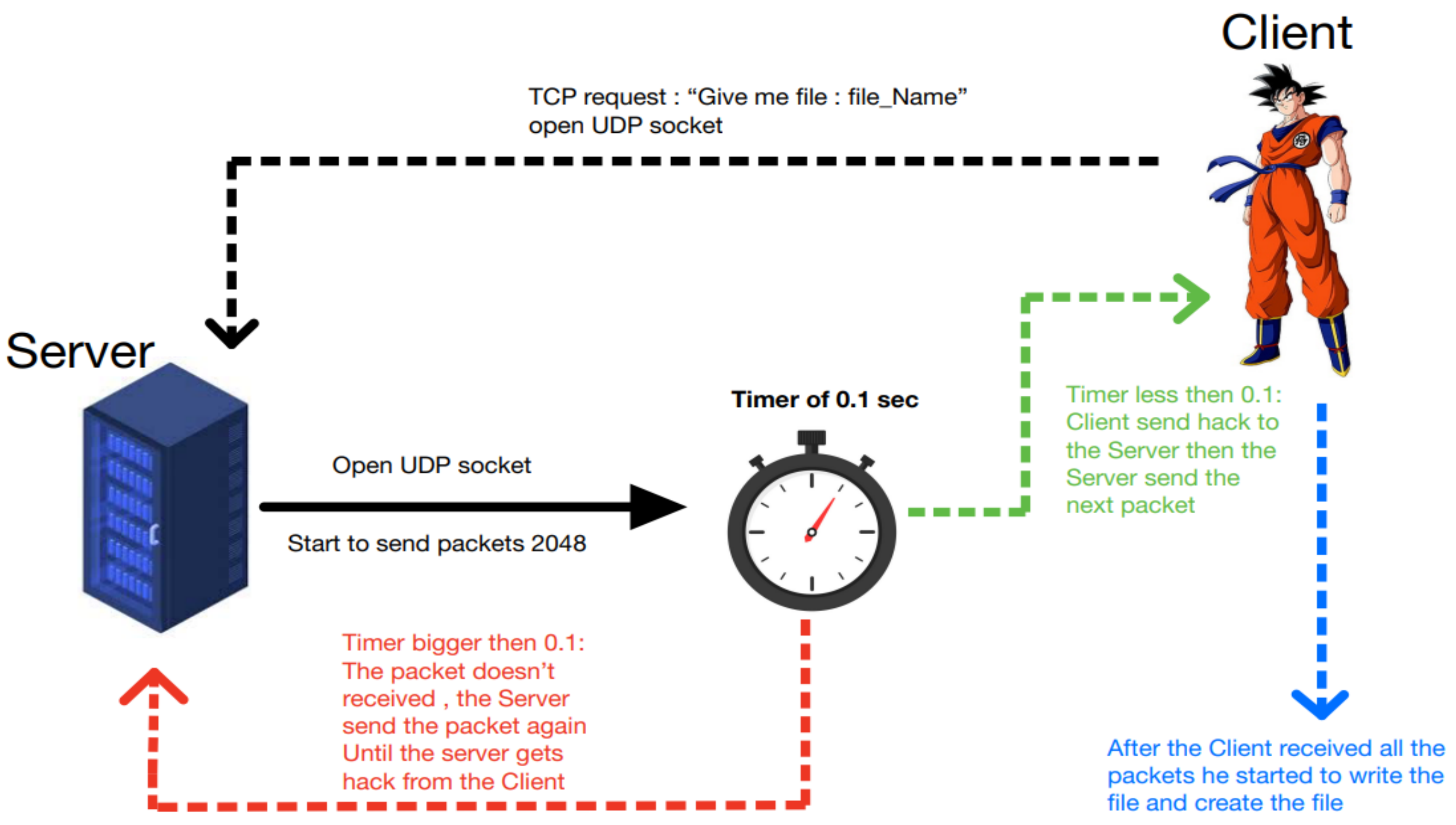
How the File transfer works:

When we got a request from the client to download file from our file list, the server started to split the file into 2048 bytes which represent one packet, in aim to solve this problem we use timer, if in the given time the client didn't send me an "ack" the server will send the packet over and over again, the client always check if the specific packet is in it data structure, if it isn't we append it to our data structure and send ack to the server, the methos we use in is "Stop and Wait", for every packet we send we wait to ack response, didn't got it? Send again and again, in this way we ensure that all the packet will arrive to the wanted destination.

Stop and wait method:


sender
receiver
first packet bit transmitted, t = 0
first packet bit arrives
last packet bit arrives, send ACK
RTT
ACK arrives, send next packet, t = RTT + L / R

RDP Diagram:


Client
TCP request : "Give me file : file_Name"
open UDP socket
Server
Timer of 0.1 sec
Open UDP socket
Start to send packets 2048
Timer less then 0.1: Client send hack to the Server then the Server send the next packet
Timer bigger then 0.1: The packet doesn't received , the Server send the packet again Until the server gets hack from the Client
After the Client received all the packets he started to write the file and create the file

Every new client who entered to our chat is adding to two lists:

*** The first one is nickname fields.**

*** The second one is the socket which the client connected to our socket.**

At first, we created a class which hold the UDP port, which work with 3 main variables:

*** Available: this parameter is Boolean and will help us to know if some port is free to use.**

*** Client Address: what is the client address we would to connect to.**

*** Sock: the UDP socket we connected to.**

There is another function which related to the UDP class, this function called "get available port", this function main aim is to check if there is an available port to send the file from.

Note! - all the previous is related to the file transfer!

The broadcast function

One of the most important function in our project is the broadcast, it might seem a little bit simple but most of the communication is using this function.

This function is going through all the clients' socket and send the messages to all the members which are online via the send function.

The receive function

This function working in an endless loop.

First, it is using the accept method, which accept an incoming connection request from TCP server.

It is asking our client for Nickname and adding it nickname to our nickname list and with the accept function we are getting the specific client socket and add it to the client socket list.

Then we add a thread to the function which will work on the handle class.

The handle function

The main goal of this function is to handle with all the requests which we get from the client class.

We are handling with commend such as:

*** Send message**

*** Send private message**

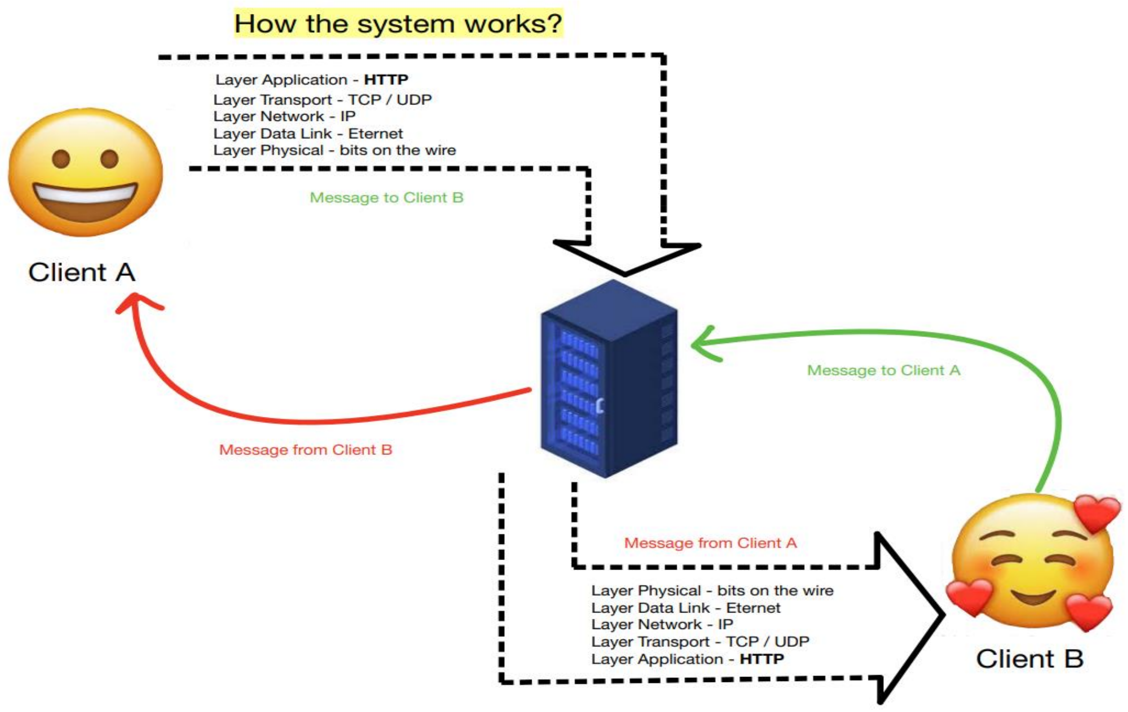
*** Show online members**

*** Show files**

*** Download files**

For every commend from the client we implemented a different implementation in aim to answer on every client's request.

We use key words, with those words the server can handle with the clients requests and provide him the right treatment.



In the diagram above we can see how a simple communication between 2 clients work.

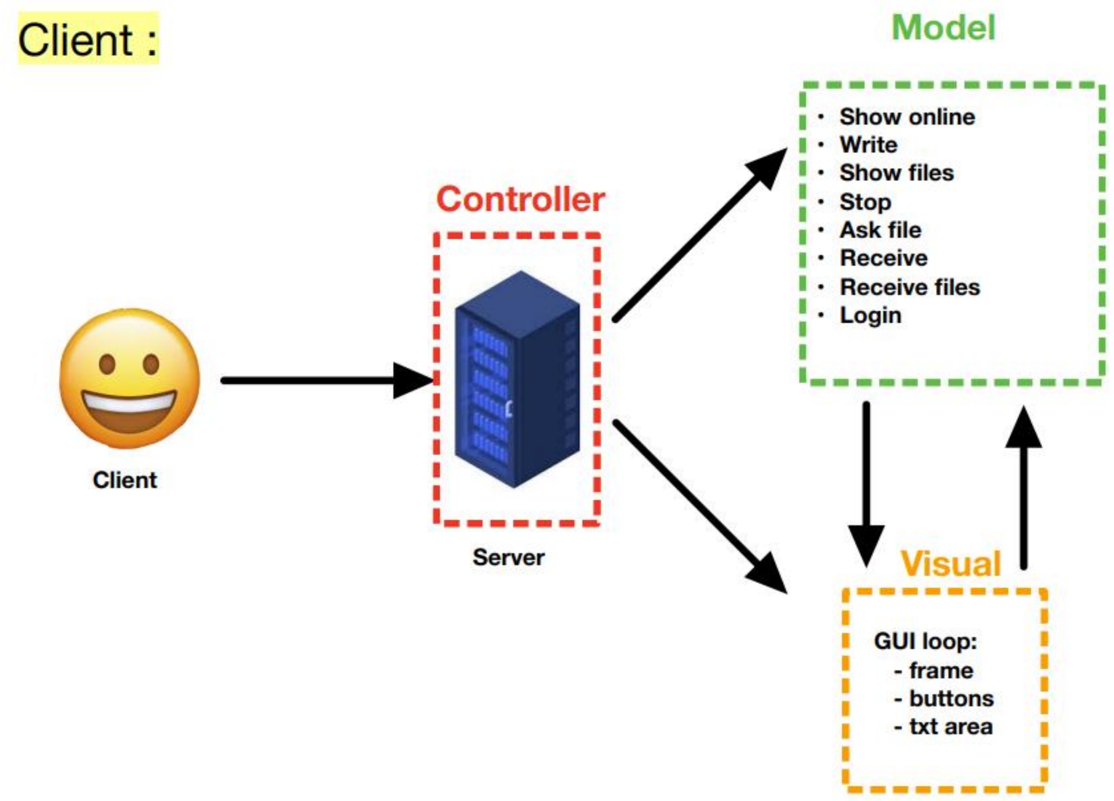
When 1 of the clients send a new message in the chat the message going through all the TCP/IP layers, start from the Application layer which is the higher layer from them all and at the end it become to bits on the wire.
Than the server send to other client the message and the other client is building the data from the lower level into the to Application layer and represent the message to the other client.

Explanation on the Client class

This class represents our clients, every new client which want to communicate via our server is going through this class.

It was a little bit difficult to deeply understand how different classes communicate with each other via commends in the GUI window, the solution was to decide about key words between the client and the server, and with those key word the server knows how to handle with the clients requests.

MVC Client diagram :



Main Client functions:

The show online function:

**This function is showing all the inline members in our chat,
the client ask send to the server request to see who is the online members on the chat with the key word “get_users”, when the server get this key word it sending to this specific client the list of the members that online.**

The write function:

This function is handling with everything we write in our text bar and sending it to the server which using the broadcast function to send all the information to the other clients.

The stop function:

**This function send to the server a message when a client is disconnect from our server.
The disconnection can be push on the “disconnect” button or the exit above, than the server get the key words “(nickname) has disconnected”, and using the broadcast method to tell all the other client the this user has been disconnected and remove it from the nickname and the client socket lists.**

The AskFile function:

This function aim is the send a request to the server for a specific file, here we are using the key word “download”.
In this part we needed to “switch” our socket from TCP one to UDP, in aim to get the RDP protocol work.

The received function:

This function is one of the most important function in the client class, in it we get all the answers from the server.
In it we are getting messages from the server such as download file or what is my nick name.

The received File function:

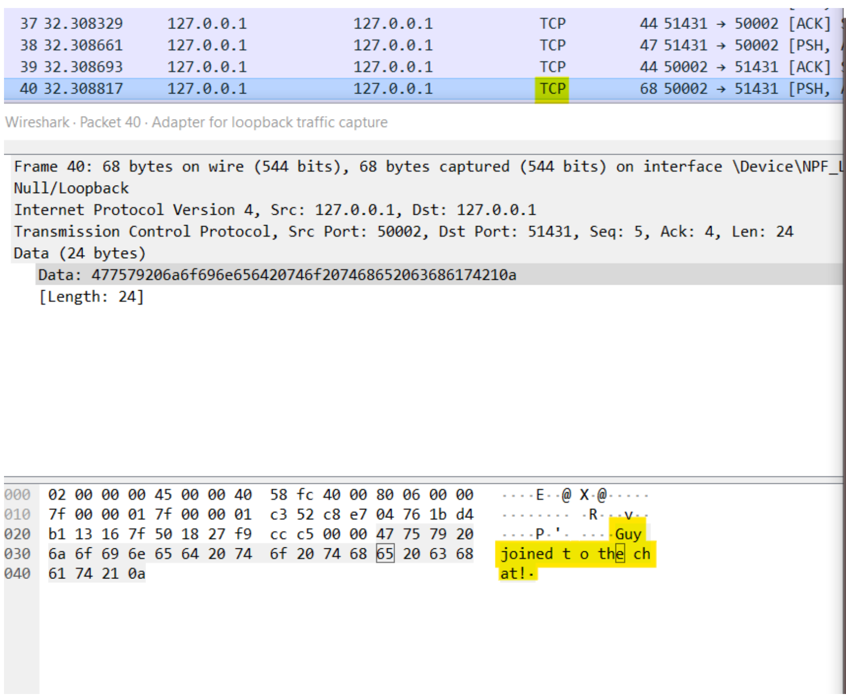
This function main aim is to handle with a file which we asked for from the sever.
Here we need to handle with packet lost, and implement the RDP protocol in the best way, for every packet we received from the server we send “ack” message.

The login function:

When a new client is connect to our server via the login button we want him to connect to our server.

WireShark capture TCP:

In the image below we can see how the wireshark sniff the TCP packets of the connection, in it we see the name of the new client.

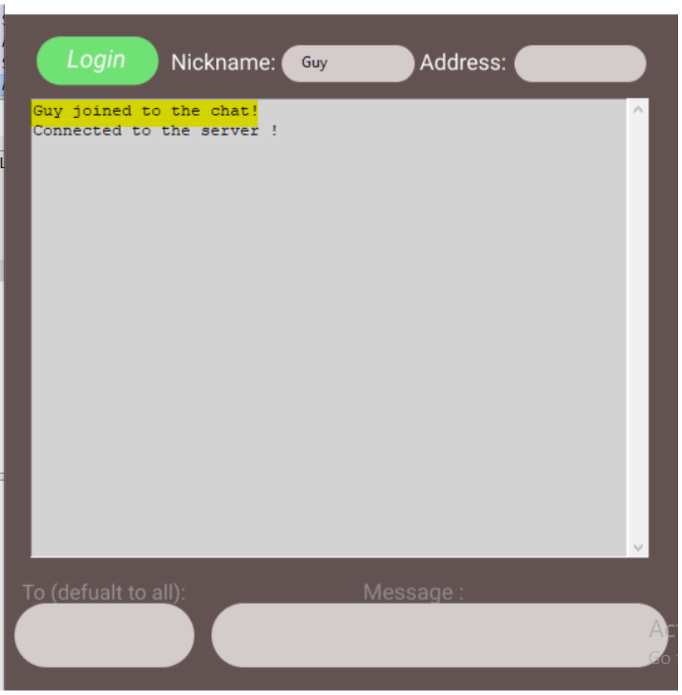


```
37 32.308329    127.0.0.1    127.0.0.1    TCP    44 51431 → 50002 [ACK]
38 32.308661    127.0.0.1    127.0.0.1    TCP    47 51431 → 50002 [PSH,
39 32.308693    127.0.0.1    127.0.0.1    TCP    44 50002 → 51431 [ACK]
40 32.308817    127.0.0.1    127.0.0.1    TCP    68 50002 → 51431 [PSH,
```

Wireshark · Packet 40 · Adapter for loopback traffic capture

```
Frame 40: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface \Device\NPF_L
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 50002, Dst Port: 51431, Seq: 5, Ack: 4, Len: 24
Data (24 bytes)
   Data: 477579206a6f696e656420746f20746865206368617421 0a
   [Length: 24]
```

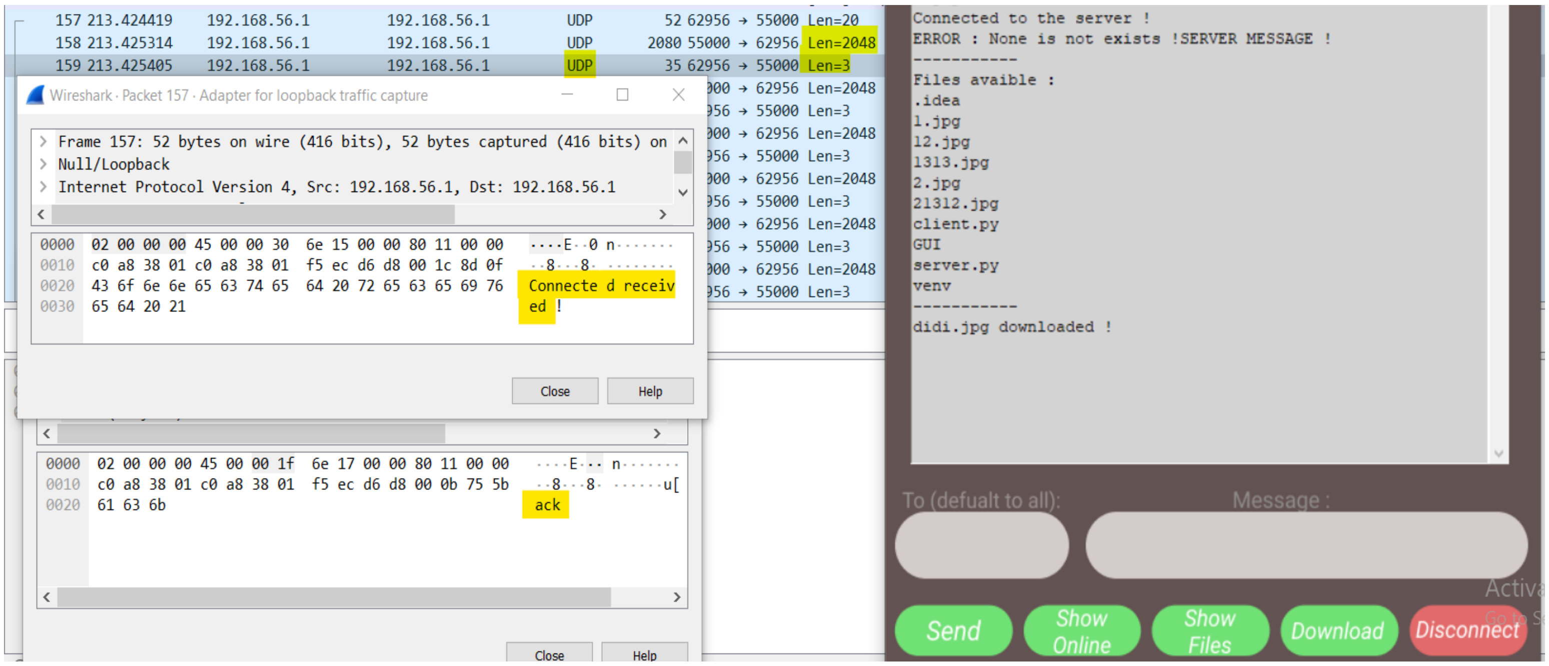
```
000  02 00 00 00 45 00 00 40  58 fc 40 00 80 06 00 00   ····E··@ X·@·····
010  7f 00 00 01 7f 00 00 01  c3 52 c8 e7 04 76 1b d4   ········ ·R···v··
020  b1 13 16 7f 50 18 27 f9  cc c5 00 00 47 75 79 20   ····P·'· ····Guy 
030  6a 6f 69 6e 65 64 20 74  6f 20 74 68 65 20 63 68   joined t o the ch
040  61 74 21 0a                                        at!·
```



```
Guy joined to the chat!
Connected to the server !
```

WireShark capture UDP:

In the image below I tried to capture file transfer and see if it capture the ack which the client send, and if the buffer size is 2048 as we wanted to:



```
157 213.424419   192.168.56.1   192.168.56.1   UDP     52 62956 → 55000 Len=20
158 213.425314   192.168.56.1   192.168.56.1   UDP   2080 55000 → 62956 Len=2048
159 213.425405   192.168.56.1   192.168.56.1   UDP     35 62956 → 55000 Len=3
                                                        000 → 62956 Len=2048
                                                        956 → 55000 Len=3
                                                        000 → 62956 Len=2048
                                                        956 → 55000 Len=3
                                                        000 → 62956 Len=2048
                                                        956 → 55000 Len=3
                                                        000 → 62956 Len=2048
                                                        956 → 55000 Len=3
                                                        000 → 62956 Len=2048
                                                        956 → 55000 Len=3
```

Wireshark · Packet 157 · Adapter for loopback traffic capture

```
> Frame 157: 52 bytes on wire (416 bits), 52 bytes captured (416 bits) on
> Null/Loopback
> Internet Protocol Version 4, Src: 192.168.56.1, Dst: 192.168.56.1

0000  02 00 00 00 45 00 00 30  6e 15 00 00 80 11 00 00   ····E··0 n·······
0010  c0 a8 38 01 c0 a8 38 01  f5 ec d6 d8 00 1c 8d 0f   ··8···8· ········
0020  43 6f 6e 6e 65 63 74 65  64 20 72 65 63 65 69 76   Connecte d receiv
0030  65 64 20 21                                        ed !
```

```
0000  02 00 00 00 45 00 00 1f  6e 17 00 00 80 11 00 00   ····E··· n·······
0010  c0 a8 38 01 c0 a8 38 01  f5 ec d6 d8 00 0b 75 5b   ··8···8· ······u[
0020  61 63 6b                                           ack
```

```
Connected to the server !
ERROR : None is not exists !SERVER MESSAGE !
-----------
Files avaible :
.idea
1.jpg
12.jpg
1313.jpg
2.jpg
21312.jpg
client.py
GUI
server.py
venv
-----------
didi.jpg downloaded !
```

Conclusions:

In this project both of us learned a lot about all the communication world using different protocols.
At the begging we tried to truly understand what is happening behind the scene, how a simple method such as bind and accept are working, when we truly achieved the knowledge how everything works, and after long designing of the code and the needed function, both of us really glad with the result, I think that in this assignment we really enjoyed the process, and this is all it matters.