

Our System TCP traffic protocol

All the messages send to the server and then the server make decision what to do with the message appending to what the client wrote in the message, when the server receive message to the client, the client make decision what to do with the message appending to what the server wrote in the message the Strings inside '< >' will help to make activities with the server.

To this end, we have prepared a protocol that will translate the messages from the server and clients into the appropriate actions.

TCP Clients Protocol: (TCP messages from a client to the server)

-message: <~><name> <msg> (name- the name of the received client, msg- the message)

Will send a private message to client.

For example:

~Daniel Hello Daniel, it's me dad!

Will send a private message to Daniel: "Hello Daniel, it's me dad!"

-message: <show>

Will receive a message to the client with the names of the other connected clients. (every list starts with word 'ALL' (it's not a client).

For example:

show

Will send message to client: ALL Daniel Dad Mom while the names of the connected clients are "Daniel", "Dad", "Mom" and the sender name .

-message: <get_list_file>

Will receive a message to the client with the names of the files that you can download from the server.

(every list starts with CHOOSE_FILE, it's not a file name).

For example:

get_list_file

Will receive a message to client: CHOOSE_FILE file1.txt file2.png file3.mp3

-message: <download_file> <filename> (<filename> - the name of the file to download)

Will send a request to download a file from the server, when the server receive this message he opens a UDP socket, sends a TCP message to the client to open a UDP socket and wait for connection. the file will be saved in a directory named "Downloads" in the client side, if the directory not exist, it will be created.

for example:

download_file file3.mp3

will send file to client

-message: <{quit}> to disconnect and exit. -

Any other message will send message to all the connected clients as a public message

TCP Server Protocol: (TCP messages from the server to the clients)

-message: **<UPD > <msg>** (msg- the message, a part of the file)

When the client requests to download a file from the server, the server divides the file into parts and sends each part of the message in the UDP protocol, the way the client recognizes that it has received a part of a file is when the message starts with the "UDP" string.

-message: **<ALL > <msg>** (msg- the message, connected clients names separated with spaces)

The server sends the names of the clients that connected to the server, and like that the client knows that if the message starts with "ALL", the following will describe what are the names of the clients that connected to the server.

For example:

ALL Daniel Dad Mom

The names of the connected clients are: "Daniel", "Dad", "Mom" and the sender name .

-message: **<<CHOOSE_FILE>> <msg>** (msg- the message, files names separated with spaces)

The server sends the names of the files that prepared to download in "FILES" directory in the server side, and like that the client knows that if the message starts with "<COOSE_FILE>", the following will describe which files the client is able to download.

For example:

ALL Daniel Dad Mom

The files are: file1.txt file2.png file3.mp3

Our System UDP traffic protocol

When a request is received from a particular client to acknowledge a file, the server searches for a free port between 55000 and 55015, opens a UDP socket and sends a message to the server that will also open a UDP socket at the same port.

These messages will be as follows:

UDP Client Protocol: (UDP messages from a client to the server)

-message: **<part> <i>** (i- index of the next part of the file)

A file is downloaded from the server via UDP messages.

Before starting the download, the server receives the file size, it divides the file into pieces of 1000 bits at most and thus knows how many parts it needs to request from the server.

-message: **<finished>**

The client sends this message when he has finished receiving all the packages he requested in order to download the file in their entirety, after making sure that each package has no information leakage, after sending the message the client closes the UDP socket and when the server receives this message he also closes the UDP socket.

UDP Server Protocol: (UDP messages from the server to the client)

-message: **<(<data>,<data_size>)>** (data- the data of the part of the requested file
data_size- size of the data of the part of the requested file.)

After a request from the client for a certain part of the file, the server seeks the file for the beginning of the requested part, saves a maximum of 1000 bits and sends the client the requested part together with the part size, this figure is useful for checking the UDP reliability.

Fast Reliable UDP

When downloading a file from the server, we want to make sure that there is no loss of packages because every slightest mistake in the information will result in the file not being able to be opened.

For this reason we must be sure that if the client writes a piece of data to a file he will write the whole piece in its entirety as sent from the server.

To this end, we implemented a method for RDTP over a UDP connection plus Congestion control,

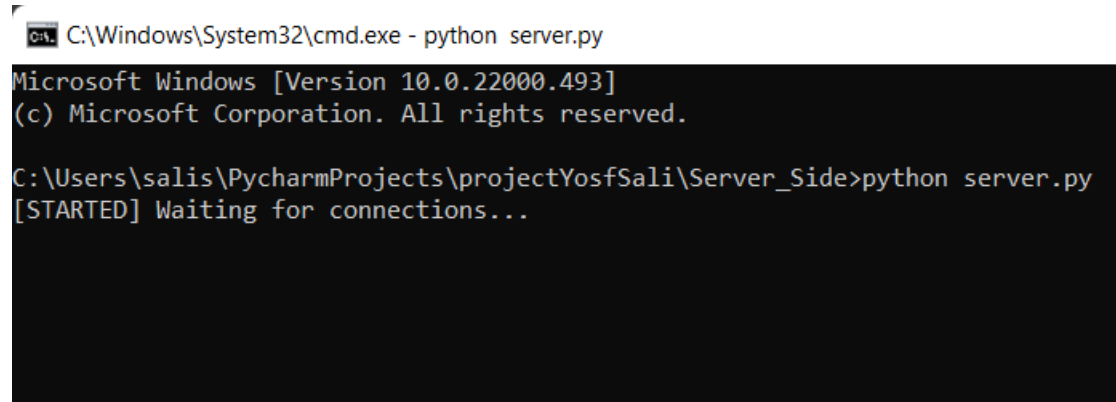
The method we wrote is, in every UDP packet sent from the server, the size of the information attached to the packet is also sent, each time the client receives such a message, he enters a loop in which he compares the size of the information sent from the server with the size of the information sent from the client, repeat the loop again until the sizes are equal, when they are equal, we will exit the loop and write the information received to the file.

```
def download(self, port, size, path):
    try:
        with open(path, "wb") as file:
            size = (size // 1000) + 1
            ADDRESS_SERVER = (self.HOST, port)
            CLIENTUDP = socket(AF_INET, SOCK_DGRAM)
            for i in range(size):
                done = False
                while not done:
                    CLIENTUDP.sendto(bytes(f"part {i}", "utf-8"), ADDRESS_SERVER)
                    msg, addr = CLIENTUDP.recvfrom(self.BUFSIZ)
                    # filter messages that are not from our server
                    while (addr != ADDRESS_SERVER):
                        msg, addr = CLIENTUDP.recvfrom(self.BUFSIZ)
                    msg_size = pickle.loads(msg)
                    if type(msg_size) == int and len(msg) == msg_size:
                        done = True
                    else:
                        print("lost a packet, send again.")
                file.write(msg)
                percent = round(i / size * 100, 2)
                print(f"{percent}% downloaded")
                if self.isgui:
                    self.percent_text.set(f"{percent}%")
                    self.progress['value'] = percent
            print("File was downloaded 100%\n")
            if self.isgui:
                self.percent_text.set("100%")
                self.pushMSG(f"{path.split('/')[-1]} downloaded successfully.\n")
                self.progress['value'] = 100
                time.sleep(2)
                self.percent_text.set("")
                self.progress['value'] = 0
    finally:
        CLIENTUDP.sendto(bytes("finished", "utf-8"), ADDRESS_SERVER)
        CLIENTUDP.close()
```

Server.py run

*Open cmd from “Server_side” folder and write the next command “python server.py”.

The Server’s output after running server.py:

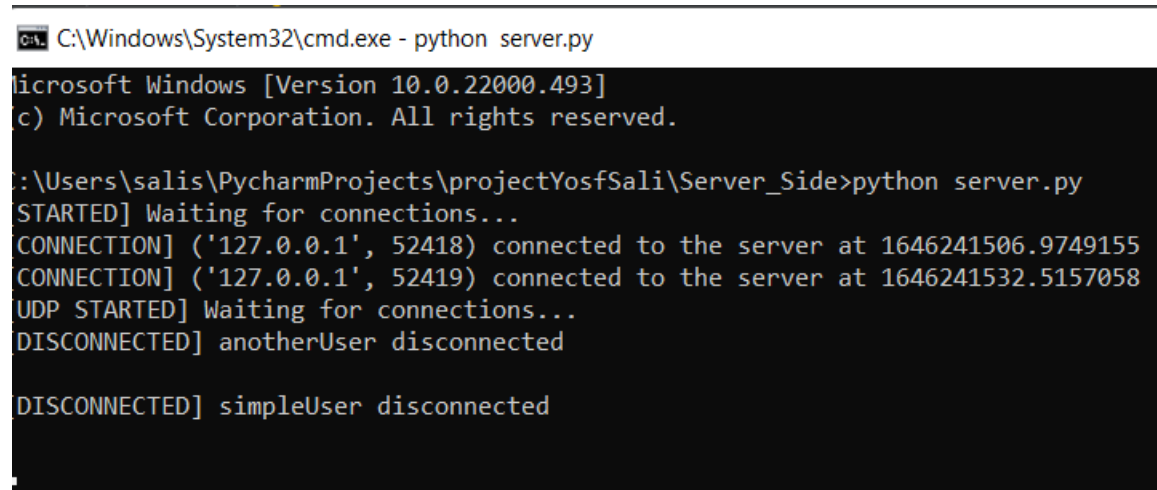


```
C:\Windows\System32\cmd.exe - python server.py
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

C:\Users\salis\PycharmProjects\projectYosfSali\Server_Side>python server.py
[STARTED] Waiting for connections...
```

Now the server is up and wait for clients connections.

The Server’s output after chat between 2 clients a download a file:



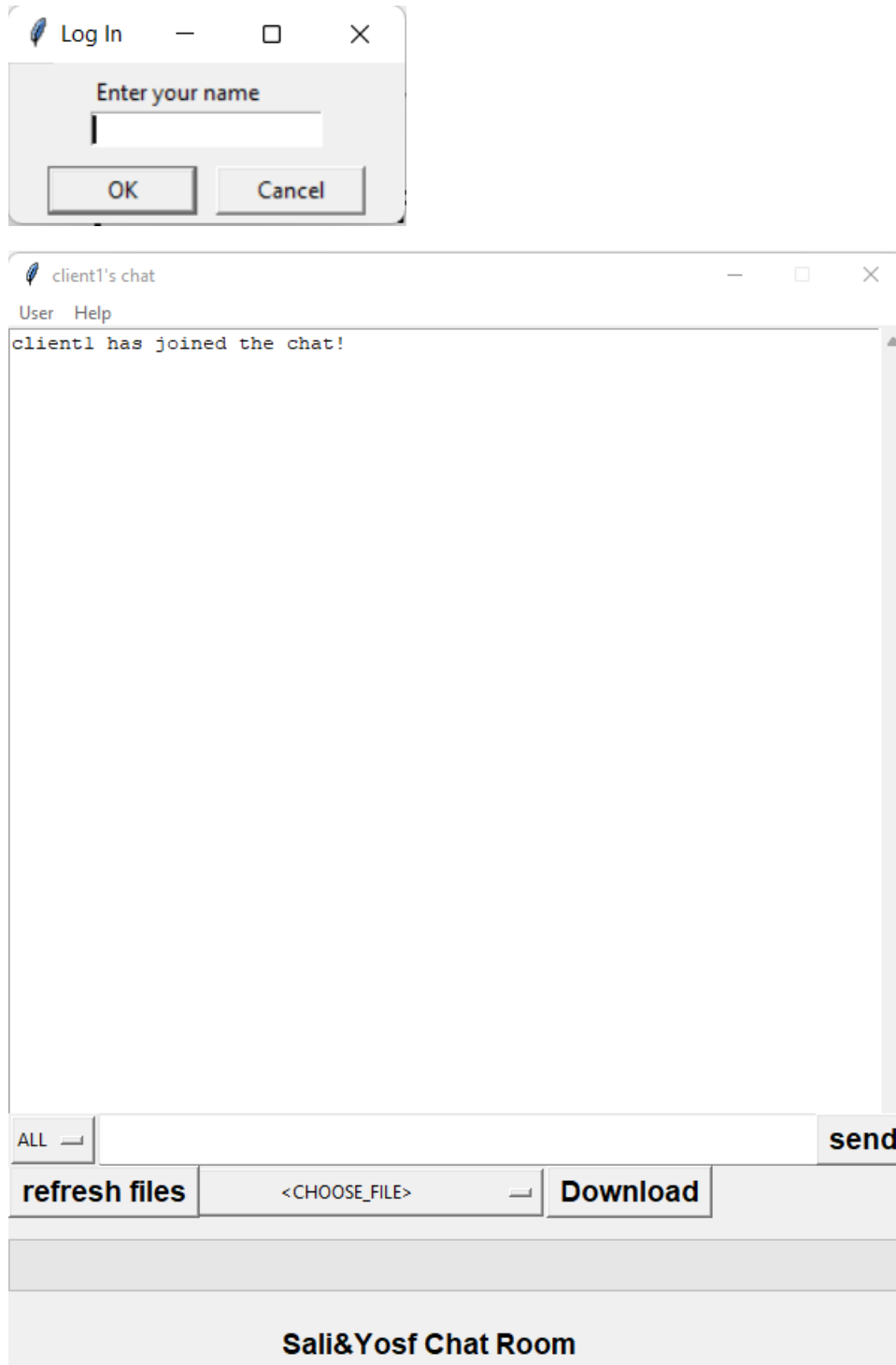
```
C:\Windows\System32\cmd.exe - python server.py
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

C:\Users\salis\PycharmProjects\projectYosfSali\Server_Side>python server.py
[STARTED] Waiting for connections...
[CONNECTION] ('127.0.0.1', 52418) connected to the server at 1646241506.9749155
[CONNECTION] ('127.0.0.1', 52419) connected to the server at 1646241532.5157058
[UDP STARTED] Waiting for connections...
[DISCONNECTED] anotherUser disconnected

[DISCONNECTED] simpleUser disconnected
```

client.py- Graphic User Interface

client.py



simple_client.py

This is an executable file that allows you to connect to the server through the command line. First the user asked to enter a name as input, then the system protocol is printed on the screen so that the user knows how to communicate with the server, then, a client object is created without a graphical interface with the name entered and then the loop begins, every message the client receives is printed on the screen and every input is sent as a message Follow the

```
Thread(target=update_messages).start()
print(f"-----welcome{name}!-----\n")
print("The input you will type will send message to the server\n")
print("The next inputs (inside the <>) will help you to make some activities with the server:\n")
print("input:<><name> <msg> to send a private message to client. (name- the name of the received client, msg- your message)\n")
print("input: <show> to get the names of the connected clients. (every list starts with ALL, its not a client)\n")
print("input: <get_list_file> to get the names of the files that you can download from the server.(every list starts with CHOOSE_FILE, its not a file name)\n")
print("input: <download_file> <filename> to download a file from the server. (<filename>- the name of the file that you want to download)\n")
print("input: <{quit}> to disconnect and exit.")
print("any other input will be sent from you to all the connected clients as a public message.\n")
print("-----sali&yosef chatroom-----\n")

while True:
    msg=input()
    if msg == '{quit}':
        cl.disconnect()
        sys.exit()
    else:
        cl.send_message(msg)
```

array protocol to communicate with the server optimally.

- * Open cmd from “Client_side” folder and write the next command “python simple_client.py”.
- * Input the name.
- * Start messaging with the server.

```
C:\Windows\System32\cmd.exe - python simple_client.py 127.0.0.1
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sali\PycharmProjects\projectYosfSali\Client_Side>python simple_client.py 127.0.0.1
Enter your name:
simpleUser
client simpleUser received message:simpleUser has joined the chat!
client simpleUser received message:ALL
-----welcome simpleUser!-----

The input you will type will send message to the server

The next inputs (inside the <>) will help you to make some activities with the server:

input:<><name> <msg> to send a private message to client. (name- the name of the received client, msg- your message)

input: <show> to get the names of the connected clients. (every list starts with ALL, its not a client)

input: <get_list_file> to get the names of the files that you can download from the server.(every list starts with CHOOSE_FILE, its not a file name)

input: <download_file> <filename> to download a file from the server. (<filename>- the name of the file that you want to download)

input: <{quit}> to disconnect and exit.
any other input will be sent from you to all the connected clients as a public message.

-----sali&yosef chatroom-----
```

An example for chatting from simple_client.py:

```
C:\Windows\System32\cmd.exe - python simple_client.py
-----sali&yosef chatroom-----

hey alll
client simpleUser received message:simpleUser: hey alll
show
client simpleUser received message:ALL
client simpleUser received message:anotherUser has joined the chat!
client simpleUser received message:ALL anotherUser
~anotherUser hey another user how are you?
client simpleUser received message:from you to anotherUser: (PRIVATE) hey another user how are you?
client simpleUser received message:anotherUser: (PRIVATE) im good how are you?
~anotherUser im good thanks
client simpleUser received message:from you to anotherUser: (PRIVATE) im good thanks
client simpleUser received message:anotherUser: (PRIVATE) have you already downloaded a file from the server?
~anotherUser no, how to do it?
client simpleUser received message:from you to anotherUser: (PRIVATE) no, how to do it?
client simpleUser received message:anotherUser: (PRIVATE) asked the server!!
~anotherUser ok
client simpleUser received message:from you to anotherUser: (PRIVATE) ok
get_list_file
client simpleUser received message:CHOOSE_FILE alice.jpg bob&alice_Wireshark.pcapng bob.jpg Ex2.jar hello.c Pokemon1
download_file hello.c
client simpleUser received message:UDP 55000 0 hello.c
0.0% downloaded
File was downloaded 100%

~anotherUser wow!! I have downloaded a C file right now
client simpleUser received message:from you to anotherUser: (PRIVATE) wow!! I have downloaded a C file right now
~anotherUser thank you
client simpleUser received message:from you to anotherUser: (PRIVATE) thank you
client simpleUser received message:anotherUser: (PRIVATE) my plesure, bye bye
~anotherUser bye
client simpleUser received message:from you to anotherUser: (PRIVATE) bye
client simpleUser received message:anotherUser left the chat...
client simpleUser received message:ALL
{quit}
```


bob_alice_test.py

In addition to the regular executable files we added a unittest file named "bob_alice_test.py".

This file shows the use of chat by two users, Bob and Alice, while running the file, two client-type objects are opened, and they activate all communication functions with the server according to the system protocol, every received message append to one of 2 lists, list for bob's received messages and list for Alice's, after each Bob and Alice operation the test system will compare the expected messages with the messages actually entering the lists..

```
# create client for bob without gui
bob = Client("Bob", False)
# create client for alice without gui
alice = Client("Alice", False)

bobmsg=[]
alicemsg=[]

def update_messages():...

class MyTestCase(unittest.TestCase):
    def setUp(self) -> None:
        # start thread for receive messages
        Thread(target=update_messages).start()

    def test_bob_alice(self):
        self.assertEqual('Bob has joined the chat!', bobmsg[0])
        self.assertEqual('Alice has joined the chat!', alicemsg[0])

        # Bob sends message to all
        bob.send_message("hello")
        time.sleep(2)
        self.assertEqual("Bob: hello", bobmsg[-1])
        self.assertEqual("Bob: hello", alicemsg[-1])

        # Alice sends message to all
        alice.send_message("hello")
        time.sleep(2)
        self.assertEqual("Alice: hello", alicemsg[-1])
        self.assertEqual("Alice: hello", bobmsg[-1])

        # Bob sends message to all
        bob.send_message("whats up")
        time.sleep(2)
        self.assertEqual("Bob: whats up", bobmsg[-1])
        self.assertEqual("Bob: whats up", alicemsg[-1])

        # Alice sends message to all
        alice.send_message("Nothing much")
        time.sleep(2)
        self.assertEqual("Alice: Nothing much", alicemsg[-1])
        self.assertEqual("Alice: Nothing much", bobmsg[-1])

        # Bob sends message privately to Alice
        bob.send_message("~Alice Hello Alice")
        time.sleep(2)
        self.assertEqual("from you to Alice: (PRIVATE) Hello Alice", bobmsg[-1])
        self.assertEqual("Bob: (PRIVATE) Hello Alice", alicemsg[-1])

        # Bob wants to know who is connected to the server
        bob.send_message("show")
        time.sleep(2)
        self.assertEqual("ALL Alice ", bobmsg[-1])

        # Alice wants to know who is connected to the server
        alice.send_message("show")
        time.sleep(2)
        self.assertEqual("ALL Bob ", alicemsg[-1])

        # Alice wants to know which files the server has for download
        alice.send_message("get_list_file")
        time.sleep(2)
        self.assertEqual('CHOOSE_FILE', alicemsg[-1].split()[0])

        # Alice download a file
        alice.send_message("download_file alice.jpg")
        time.sleep(3)
        self.assertEqual('UDP', alicemsg[-1].split()[0])

        # Bob wants to know which files the server has for download
        bob.send_message("get_list_file")
        time.sleep(2)
        self.assertEqual('CHOOSE_FILE', bobmsg[-1].split()[0])

        # Bob download a file
        bob.send_message("download_file bob.jpg")
        time.sleep(3)
        self.assertEqual('UDP', bobmsg[-1].split()[0])

        # Bob disconnect
        bob.disconnect()
        time.sleep(2)
        self.assertEqual('Bob left the chat...', alicemsg[-2])
        self.assertEqual('ALL ', alicemsg[-1])

        # Alice disconnect
        alice.disconnect()
        print('Alice left the chat...')
```

As you can see in the code comments, first Bob and Alice connect to the server, then Bob sends a public message "hello", Alice sends a public message "hello".

Bob sends a public message "whats up", Alice sends a public message "Nothing much", Bob Sends private message to Alice "Hello Alice", Bob sends "show" to get the list of connected users, Alice sends "show" to get the list of connected users, Alice sends "get_list_file" to get the list of files from the server, Alice sends "download_file alice.jpg "To download her image from the server, Bob sends" get_list_file "to get the list of files from the server, Bob sends" download_file bob.jpg "to download her image from the server, Bob sends" {quit} "to disconnect from the server and finally Alice sends {"quit"} to disconnect from the server.

test_bob_alice.py run

after running the server,

*Open cmd from “Client_side” folder.

*Write the next command “python test_bob_alice.py <IPv4>”

<IPv4>- the IPv4 address of the server that already run.

for example: python test_bob_alice.py 127.0.0.1 (this IP address was set default you are not have to write it)

```

Select C:\Windows\System32\cmd.exe - python bob_alice_test.py 127.0.0.1
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sali\PycharmProjects\projectYosfSali\Client_Side>python bob_alice_test.py 127.0.0.1
client Bob received message:Bob has joined the chat!
client Bob received message:All
client Bob received message:Alice has joined the chat!
client Bob received message:All Alice
client Alice received message:Alice has joined the chat!
client Alice received message:All Bob
client Bob received message:Bob: hello
client Alice received message:Bob: hello
client Bob received message:Alice: hello
client Alice received message:Alice: hello
client Bob received message:Bob: whats up
client Alice received message:Bob: whats up
client Bob received message:Alice: Nothing much
client Alice received message:Alice: Nothing much
client Bob received message:from you to Alice: (PRIVATE) Hello Alice
client Alice received message:Bob: (PRIVATE) Hello Alice
client Bob received message:All Alice
client Alice received message:All Bob
client Alice received message:CHOOSE FILE alice.jpg bob&alice_Wireshark.pcapng bob.jpg Ex2.jar hello.c Pok
client Alice received message:UDP 55001 10341 bob.jpg
client Bob received message:CHOOSE FILE alice.jpg bob&alice_Wireshark.pcapng bob.jpg Ex2.jar hello.c Pok
client Bob received message:UDP 55001 10341 bob.jpg
0.0% downloaded
0.62% downloaded 23.6% downloaded 45.96% downloaded 68.94% downloaded 91.93% downloaded
1.24% downloaded 24.22% downloaded 46.58% downloaded 69.57% downloaded 92.55% downloaded
1.86% downloaded 24.84% downloaded 47.2% downloaded 70.19% downloaded 93.17% downloaded
2.48% downloaded 25.47% downloaded 47.81% downloaded 70.81% downloaded 93.79% downloaded
3.11% downloaded 26.09% downloaded 48.43% downloaded 71.43% downloaded 94.41% downloaded
3.73% downloaded 26.71% downloaded 49.0% downloaded 72.05% downloaded 95.03% downloaded
4.35% downloaded 27.33% downloaded 49.6% downloaded 72.67% downloaded 95.65% downloaded
4.97% downloaded 27.95% downloaded 50.1% downloaded 73.29% downloaded 96.27% downloaded
5.59% downloaded 28.57% downloaded 50.7% downloaded 73.91% downloaded 96.89% downloaded
6.21% downloaded 29.19% downloaded 51.3% downloaded 74.53% downloaded 97.52% downloaded
6.83% downloaded 29.81% downloaded 51.9% downloaded 75.16% downloaded 98.14% downloaded
7.45% downloaded 30.43% downloaded 52.5% downloaded 75.78% downloaded 98.76% downloaded
8.07% downloaded 31.05% downloaded 53.12% downloaded 76.4% downloaded 99.38% downloaded
8.7% downloaded 31.68% downloaded 53.74% downloaded 77.02% downloaded
9.32% downloaded 32.3% downloaded 54.36% downloaded 77.64% downloaded
9.94% downloaded 32.92% downloaded 54.98% downloaded 78.26% downloaded
10.56% downloaded 33.54% downloaded 55.6% downloaded 78.88% downloaded
11.18% downloaded 34.16% downloaded 56.22% downloaded 79.5% downloaded
11.8% downloaded 34.78% downloaded 56.84% downloaded 80.12% downloaded
12.42% downloaded 35.4% downloaded 57.46% downloaded 80.74% downloaded
13.04% downloaded 36.02% downloaded 58.08% downloaded 81.36% downloaded
13.66% downloaded 36.64% downloaded 58.7% downloaded 81.98% downloaded
14.28% downloaded 37.26% downloaded 59.32% downloaded 82.6% downloaded
14.9% downloaded 37.88% downloaded 59.94% downloaded 83.22% downloaded
15.52% downloaded 38.5% downloaded 60.56% downloaded 83.84% downloaded
16.14% downloaded 39.12% downloaded 61.18% downloaded 84.46% downloaded
16.76% downloaded 39.74% downloaded 61.8% downloaded 85.08% downloaded
17.38% downloaded 40.36% downloaded 62.42% downloaded 85.7% downloaded
18.0% downloaded 40.98% downloaded 63.04% downloaded 86.32% downloaded
18.62% downloaded 41.6% downloaded 63.66% downloaded 86.94% downloaded
19.24% downloaded 42.22% downloaded 64.28% downloaded 87.56% downloaded
20.86% downloaded 42.84% downloaded 64.9% downloaded 88.18% downloaded
21.48% downloaded 43.46% downloaded 65.52% downloaded 88.8% downloaded
22.1% downloaded 44.08% downloaded 66.14% downloaded 89.42% downloaded
22.72% downloaded 44.7% downloaded 66.76% downloaded 90.04% downloaded
23.34% downloaded 45.32% downloaded 67.38% downloaded 90.66% downloaded
23.96% downloaded 45.94% downloaded 68.0% downloaded 91.28% downloaded

```

The Server's output after running test bob alice.py:

```

C:\Windows\System32\cmd.exe - python server.py
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sali\PycharmProjects\projectYosfSali\Server_Side>python server.py
[STARTED] Waiting for connections...
[CONNECTION] ('127.0.0.1', 51715) connected to the server at 1646231821.845275
[CONNECTION] ('127.0.0.1', 51716) connected to the server at 1646231821.845275
[UDP STARTED] Waiting for connections...
[DISCONNECTED] Bob disconnected
[DISCONNECTED] Alice disconnected

```

test_bob_alice.pcapng- Wireshark Record

While running the " test_bob_alice.py" program we will run Wireshark to capture the facts that are sent and received between the clients and the server.

You can find the full recording file named " test_bob_alice.pcapng" in the project files.

Representative screenshots of selected parts of the recording file:

TCP traffic:

No. 1- Bob (port 51017) send a connection request to the server (port 55000).

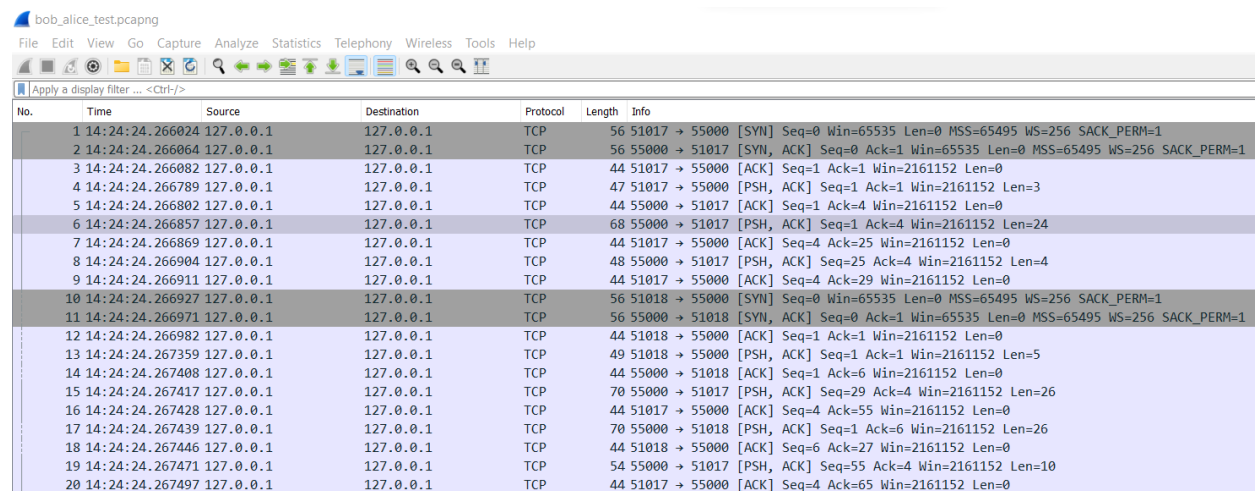
No. 2- server (port 55000) send ACK message to Bob (port 51017).

No. 3-9- TCP messages and ACK between server and BOB.

No. 10- Alice (port 51018) send a connection request to the server (port 55000).

No. 11- server (port 55000) send ACK message to Alice (port 51018).

No. 12-68- messages and ACK between server and Alice or server and BOB.



The screenshot shows the Wireshark interface with a packet capture of 'bob_alice_test.pcapng'. The packet list pane displays 20 packets. The first 10 packets (1-10) represent the initial connection and data exchange between Bob (127.0.0.1) and the server (127.0.0.1). The remaining 10 packets (11-20) represent the connection and data exchange between Alice (127.0.0.1) and the server (127.0.0.1). The packet details pane shows the selected packet (No. 1) as a SYN packet from Bob to the server.

No.	Time	Source	Destination	Protocol	Length	Info
1	14:24:24.266024	127.0.0.1	127.0.0.1	TCP	56	51017 → 55000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	14:24:24.266064	127.0.0.1	127.0.0.1	TCP	56	55000 → 51017 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	14:24:24.266082	127.0.0.1	127.0.0.1	TCP	44	51017 → 55000 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
4	14:24:24.266789	127.0.0.1	127.0.0.1	TCP	47	51017 → 55000 [PSH, ACK] Seq=1 Ack=1 Win=2161152 Len=3
5	14:24:24.266802	127.0.0.1	127.0.0.1	TCP	44	55000 → 51017 [ACK] Seq=1 Ack=4 Win=2161152 Len=0
6	14:24:24.266857	127.0.0.1	127.0.0.1	TCP	68	55000 → 51017 [PSH, ACK] Seq=1 Ack=4 Win=2161152 Len=24
7	14:24:24.266869	127.0.0.1	127.0.0.1	TCP	44	51017 → 55000 [ACK] Seq=4 Ack=25 Win=2161152 Len=0
8	14:24:24.266904	127.0.0.1	127.0.0.1	TCP	48	55000 → 51017 [PSH, ACK] Seq=25 Ack=4 Win=2161152 Len=4
9	14:24:24.266911	127.0.0.1	127.0.0.1	TCP	44	51017 → 55000 [ACK] Seq=4 Ack=29 Win=2161152 Len=0
10	14:24:24.266927	127.0.0.1	127.0.0.1	TCP	56	51018 → 55000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
11	14:24:24.266971	127.0.0.1	127.0.0.1	TCP	56	55000 → 51018 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
12	14:24:24.266982	127.0.0.1	127.0.0.1	TCP	44	51018 → 55000 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
13	14:24:24.267359	127.0.0.1	127.0.0.1	TCP	49	51018 → 55000 [PSH, ACK] Seq=1 Ack=1 Win=2161152 Len=5
14	14:24:24.267408	127.0.0.1	127.0.0.1	TCP	44	55000 → 51018 [ACK] Seq=1 Ack=6 Win=2161152 Len=0
15	14:24:24.267417	127.0.0.1	127.0.0.1	TCP	70	55000 → 51017 [PSH, ACK] Seq=29 Ack=4 Win=2161152 Len=26
16	14:24:24.267428	127.0.0.1	127.0.0.1	TCP	44	51017 → 55000 [ACK] Seq=4 Ack=55 Win=2161152 Len=0
17	14:24:24.267439	127.0.0.1	127.0.0.1	TCP	70	55000 → 51018 [PSH, ACK] Seq=1 Ack=6 Win=2161152 Len=26
18	14:24:24.267446	127.0.0.1	127.0.0.1	TCP	44	51018 → 55000 [ACK] Seq=6 Ack=27 Win=2161152 Len=0
19	14:24:24.267471	127.0.0.1	127.0.0.1	TCP	54	55000 → 51017 [PSH, ACK] Seq=55 Ack=4 Win=2161152 Len=10
20	14:24:24.267497	127.0.0.1	127.0.0.1	TCP	44	51017 → 55000 [ACK] Seq=4 Ack=65 Win=2161152 Len=0

No. 69-1035- UDP traffic between Alice (port 59120) and server (port 55000) for download an image.

No.	Time	Source	Destination	Protocol	Length	Info
61	14:24:38.347331	127.0.0.1	127.0.0.1	TCP	57	51018 → 55000 [PSH, ACK] Seq=27 Ack=124 Win=2161152 Len=13
62	14:24:38.347405	127.0.0.1	127.0.0.1	TCP	44	55000 → 51018 [ACK] Seq=124 Ack=40 Win=2161152 Len=0
63	14:24:38.347888	127.0.0.1	127.0.0.1	TCP	164	55000 → 51018 [PSH, ACK] Seq=124 Ack=40 Win=2161152 Len=120
64	14:24:38.347945	127.0.0.1	127.0.0.1	TCP	44	51018 → 55000 [ACK] Seq=40 Ack=244 Win=2160896 Len=0
65	14:24:40.352229	127.0.0.1	127.0.0.1	TCP	67	51018 → 55000 [PSH, ACK] Seq=40 Ack=244 Win=2160896 Len=23
66	14:24:40.352295	127.0.0.1	127.0.0.1	TCP	44	55000 → 51018 [ACK] Seq=244 Ack=63 Win=2161152 Len=0
67	14:24:40.352988	127.0.0.1	127.0.0.1	TCP	69	55000 → 51018 [PSH, ACK] Seq=244 Ack=63 Win=2161152 Len=25
68	14:24:40.353047	127.0.0.1	127.0.0.1	TCP	44	51018 → 55000 [ACK] Seq=63 Ack=269 Win=2160896 Len=0
69	14:24:40.354796	127.0.0.1	127.0.0.1	UDP	38	59120 → 55000 Len=6
70	14:24:40.356845	127.0.0.1	127.0.0.1	UDP	1050	55000 → 59120 Len=1018
71	14:24:40.357089	127.0.0.1	127.0.0.1	UDP	38	59120 → 55000 Len=6
72	14:24:40.357334	127.0.0.1	127.0.0.1	UDP	1050	55000 → 59120 Len=1018
73	14:24:40.357539	127.0.0.1	127.0.0.1	UDP	38	59120 → 55000 Len=6
74	14:24:40.357744	127.0.0.1	127.0.0.1	UDP	1050	55000 → 59120 Len=1018
75	14:24:40.358020	127.0.0.1	127.0.0.1	UDP	38	59120 → 55000 Len=6
76	14:24:40.358212	127.0.0.1	127.0.0.1	UDP	1050	55000 → 59120 Len=1018
77	14:24:40.358396	127.0.0.1	127.0.0.1	UDP	38	59120 → 55000 Len=6
78	14:24:40.358589	127.0.0.1	127.0.0.1	UDP	1050	55000 → 59120 Len=1018
79	14:24:40.358770	127.0.0.1	127.0.0.1	UDP	38	59120 → 55000 Len=6
80	14:24:40.358965	127.0.0.1	127.0.0.1	UDP	1050	55000 → 59120 Len=1018
81	14:24:40.359207	127.0.0.1	127.0.0.1	UDP	38	59120 → 55000 Len=6
82	14:24:40.359400	127.0.0.1	127.0.0.1	UDP	1050	55000 → 59120 Len=1018
83	14:24:40.359586	127.0.0.1	127.0.0.1	UDP	38	59120 → 55000 Len=6
84	14:24:40.359779	127.0.0.1	127.0.0.1	UDP	1050	55000 → 59120 Len=1018
85	14:24:40.359962	127.0.0.1	127.0.0.1	UDP	38	59120 → 55000 Len=6
86	14:24:40.360155	127.0.0.1	127.0.0.1	UDP	1050	55000 → 59120 Len=1018
87	14:24:40.360395	127.0.0.1	127.0.0.1	UDP	38	59120 → 55000 Len=6
88	14:24:40.360587	127.0.0.1	127.0.0.1	UDP	1050	55000 → 59120 Len=1018
89	14:24:40.360773	127.0.0.1	127.0.0.1	UDP	38	59120 → 55000 Len=6
90	14:24:40.360965	127.0.0.1	127.0.0.1	UDP	1050	55000 → 59120 Len=1018
91	14:24:40.361150	127.0.0.1	127.0.0.1	UDP	38	59120 → 55000 Len=6
92	14:24:40.361341	127.0.0.1	127.0.0.1	UDP	1050	55000 → 59120 Len=1018
93	14:24:40.361581	127.0.0.1	127.0.0.1	UDP	38	59120 → 55000 Len=6
94	14:24:40.361776	127.0.0.1	127.0.0.1	UDP	1050	55000 → 59120 Len=1018

No. 1036-1043- TCP messages and ACK between server and BOB.

1035	14:24:40.392064	127.0.0.1	127.0.0.1	UDP	40	59120 → 55000 Len=8
1036	14:24:43.367150	127.0.0.1	127.0.0.1	TCP	57	51017 → 55000 [PSH, ACK] Seq=39 Ack=170 Win=2161152 Len=13
1037	14:24:43.367240	127.0.0.1	127.0.0.1	TCP	44	55000 → 51017 [ACK] Seq=170 Ack=52 Win=2161152 Len=0
1038	14:24:43.367721	127.0.0.1	127.0.0.1	TCP	164	55000 → 51017 [PSH, ACK] Seq=170 Ack=52 Win=2161152 Len=120
1039	14:24:43.367768	127.0.0.1	127.0.0.1	TCP	44	51017 → 55000 [ACK] Seq=52 Ack=290 Win=2160896 Len=0
1040	14:24:45.372639	127.0.0.1	127.0.0.1	TCP	65	51017 → 55000 [PSH, ACK] Seq=52 Ack=290 Win=2160896 Len=21
1041	14:24:45.372727	127.0.0.1	127.0.0.1	TCP	44	55000 → 51017 [ACK] Seq=290 Ack=73 Win=2161152 Len=0
1042	14:24:45.373435	127.0.0.1	127.0.0.1	TCP	67	55000 → 51017 [PSH, ACK] Seq=290 Ack=73 Win=2161152 Len=23
1043	14:24:45.373498	127.0.0.1	127.0.0.1	TCP	44	51017 → 55000 [ACK] Seq=73 Ack=313 Win=2160896 Len=0
1044	14:24:45.375031	127.0.0.1	127.0.0.1	UDP	38	59121 → 55001 Len=6

No. 1044-1290- UDP traffic between Bob (port 59121) and server (port 55001) for download an image.

No. 1291-1292- TCP message and ACK from Bob to server

No. 1293-1295- TCP messages between Bob and server, Bob is disconnecting.

No. 1296-1301- TCP message and ACK from Alice to server

No. 1302- TCP message from Alice to server, Alice is disconnecting

1289	14:24:45.392504	127.0.0.1	127.0.0.1	UDP	47	55001 → 59121 Len=15
1290	14:24:45.392651	127.0.0.1	127.0.0.1	UDP	40	59121 → 55001 Len=8
1291	14:24:48.387567	127.0.0.1	127.0.0.1	TCP	50	51017 → 55000 [PSH, ACK] Seq=73 Ack=313 Win=2160896 Len=6
1292	14:24:48.387638	127.0.0.1	127.0.0.1	TCP	44	55000 → 51017 [ACK] Seq=313 Ack=79 Win=2161152 Len=0
1293	14:24:48.387720	127.0.0.1	127.0.0.1	TCP	44	51017 → 55000 [RST, ACK] Seq=79 Ack=313 Win=0 Len=0
1294	14:24:48.387726	127.0.0.1	127.0.0.1	TCP	44	55000 → 51017 [FIN, ACK] Seq=313 Ack=79 Win=2161152 Len=0
1295	14:24:48.387763	127.0.0.1	127.0.0.1	TCP	44	51017 → 55000 [RST] Seq=79 Win=0 Len=0
1296	14:24:48.387854	127.0.0.1	127.0.0.1	TCP	64	55000 → 51018 [PSH, ACK] Seq=269 Ack=63 Win=2161152 Len=20
1297	14:24:48.387896	127.0.0.1	127.0.0.1	TCP	44	51018 → 55000 [ACK] Seq=63 Ack=289 Win=2160896 Len=0
1298	14:24:48.387988	127.0.0.1	127.0.0.1	TCP	48	55000 → 51018 [PSH, ACK] Seq=289 Ack=63 Win=2161152 Len=4
1299	14:24:48.388011	127.0.0.1	127.0.0.1	TCP	44	51018 → 55000 [ACK] Seq=63 Ack=293 Win=2160896 Len=0
1300	14:24:50.392998	127.0.0.1	127.0.0.1	TCP	50	51018 → 55000 [PSH, ACK] Seq=63 Ack=293 Win=2160896 Len=6
1301	14:24:50.393040	127.0.0.1	127.0.0.1	TCP	44	55000 → 51018 [ACK] Seq=293 Ack=69 Win=2161152 Len=0
1302	14:24:50.393082	127.0.0.1	127.0.0.1	TCP	44	51018 → 55000 [RST, ACK] Seq=69 Ack=293 Win=0 Len=0