פרויקט סיום

אילן מאיר סופיר : 342615648

בן כהן : 207029786

חלק א

In this first part, we were asked to build in python : a primitive instant messaging system (similar to MSN) based on network.

רשימת הודעות שנשלחות מהלקוח (לשרת):

1) אני רוצה להתחבר: <connect><name>
2) אני רוצה לקבל את רשימת המחוברים: <get_users>
3) אני רוצה להתנתק: <disconnect>
4) אני רוצה לשלוח הודעה ללקוח: <set_msg><Name>
5) אני רוצה לשלוח הודעה לכולם: <set_msg_all>
6) אני רוצה לקבל את רשימת הקבצים שיש בשרת: <get_list_file>
7) אני רוצה להוריד קובץ <test.txt": <download> < test.txt > <id"
8) המשך בהעברת הקובץ: <proceed>

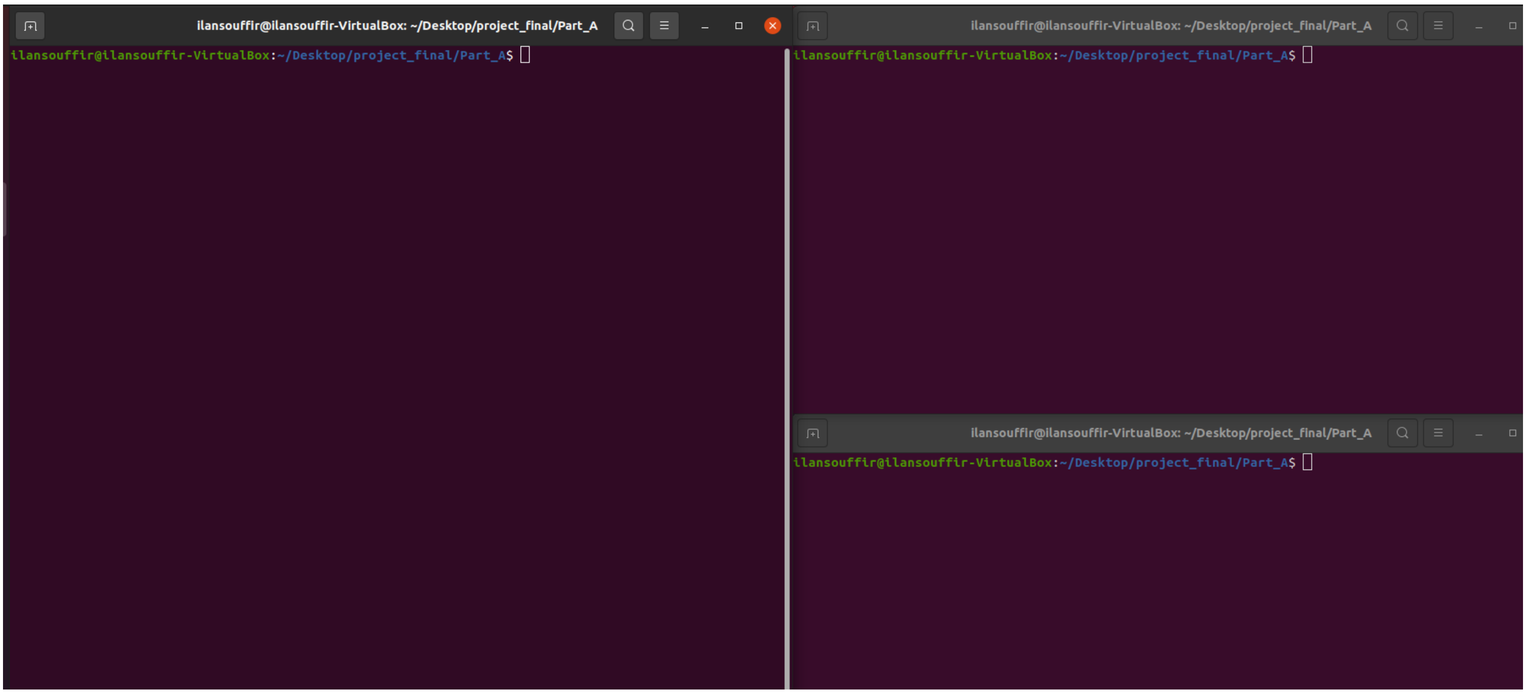
Then we created 2 classes : server1.py and client1.py to solve the requested exercise.

For our exemple we wanted there to be 2 users who communicate with each other on the same server, their names will be Ilan and Ben.

How to Run ?

We run our messaging system on linux

1. Open 3 terminals (one for the server and two for the users Ilan and Ben).



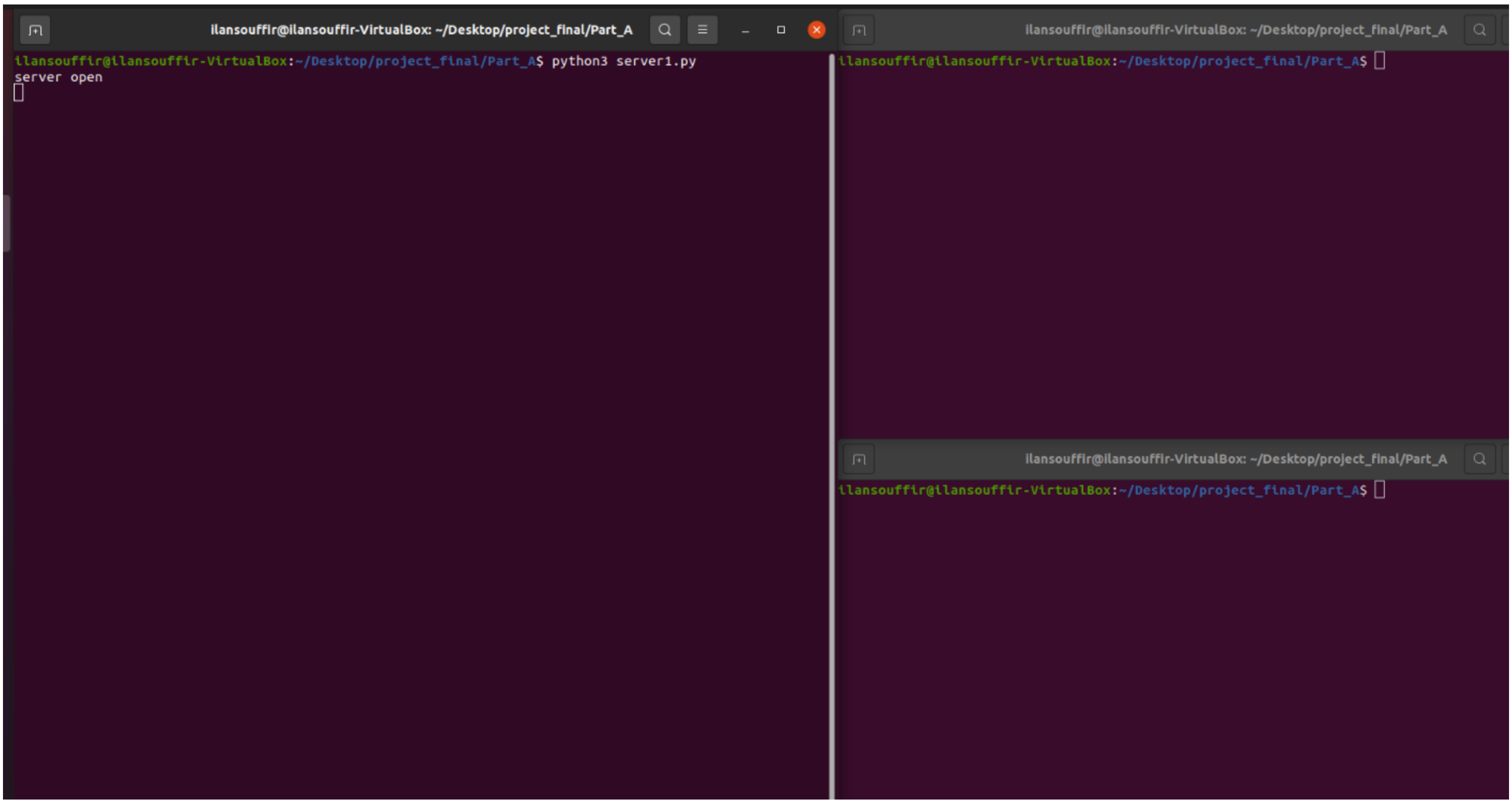
```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$
```

ilansouffir@ilansouffir-VirtualBox: ~/Desktop/project_final/Part_A
ilansouffir@ilansouffir-VirtualBox: ~/Desktop/project_final/Part_A
ilansouffir@ilansouffir-VirtualBox: ~/Desktop/project_final/Part_A

2. Write in the first one « python3 server1.py » to create the server.

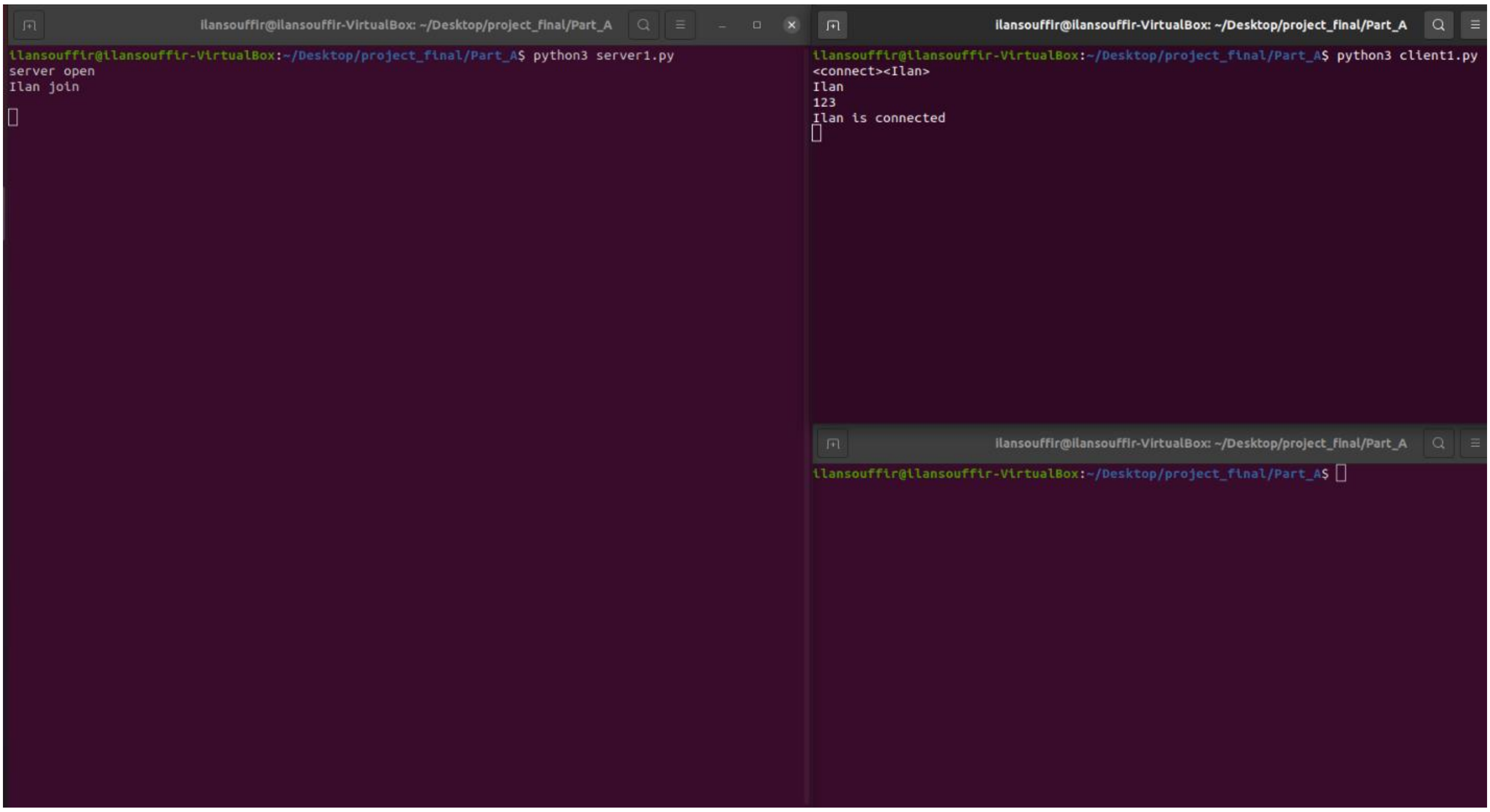


```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 server1.py
server open
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$
```

3. Write in the second one « python3 client1.py » to create the first user with name Ilan. We use the command <connect><Name> such that Name = Ilan.

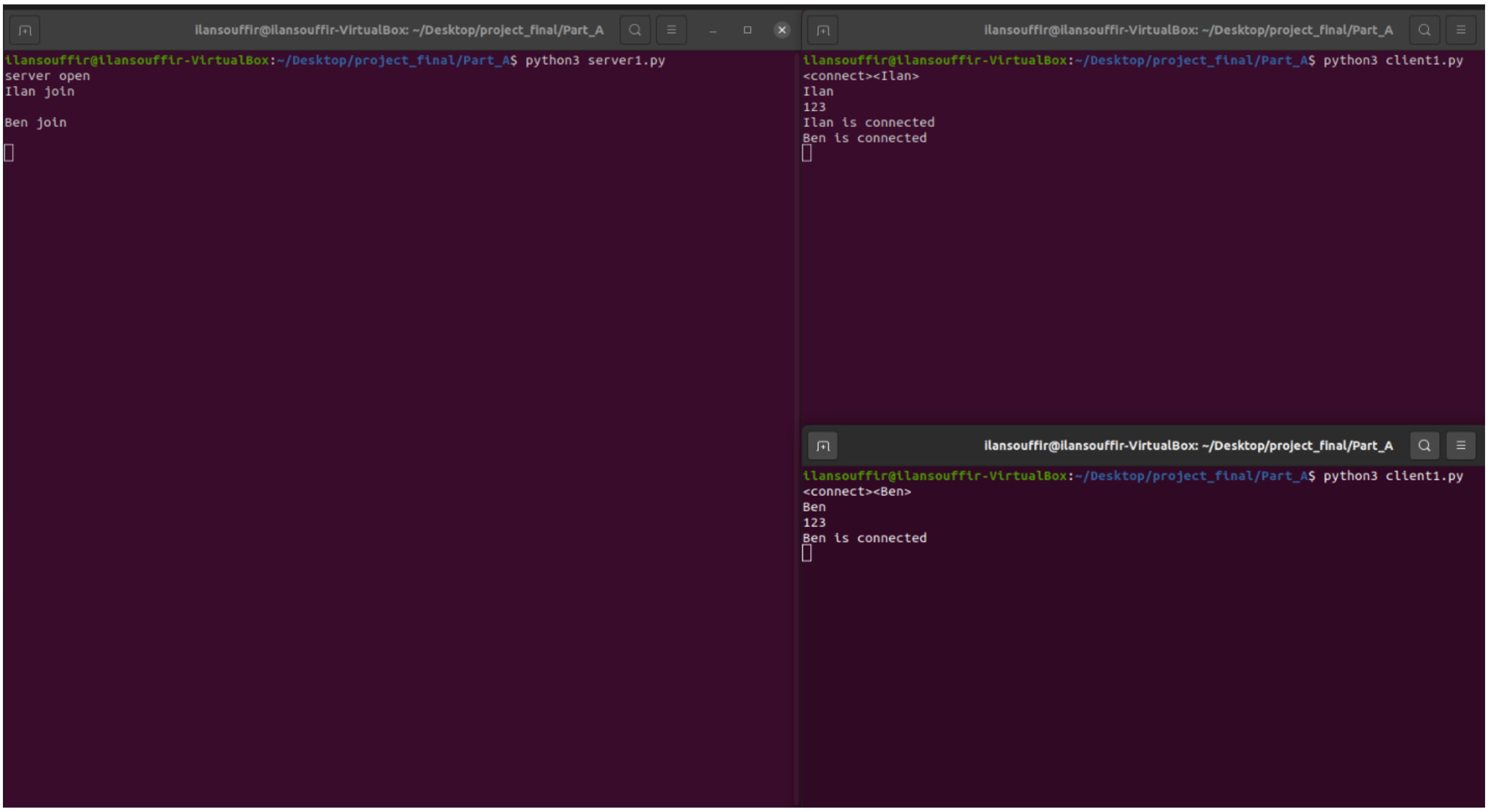


```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 server1.py
server open
Ilan join
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 client1.py
<connect><Ilan>
Ilan
123
Ilan is connected
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$
```

4. Write in the thirdone « python3 client1.py » to create the second user with name Ben. We use the command <connect><Name> such that Name = Ben.


ilansouffir@ilansouffir-VirtualBox: ~/Desktop/project_final/Part_A
ilansouffir@ilansouffir-VirtualBox: ~/Desktop/project_final/Part_A
ilansouffir@ilansouffir-VirtualBox: ~/Desktop/project_final/Part_A

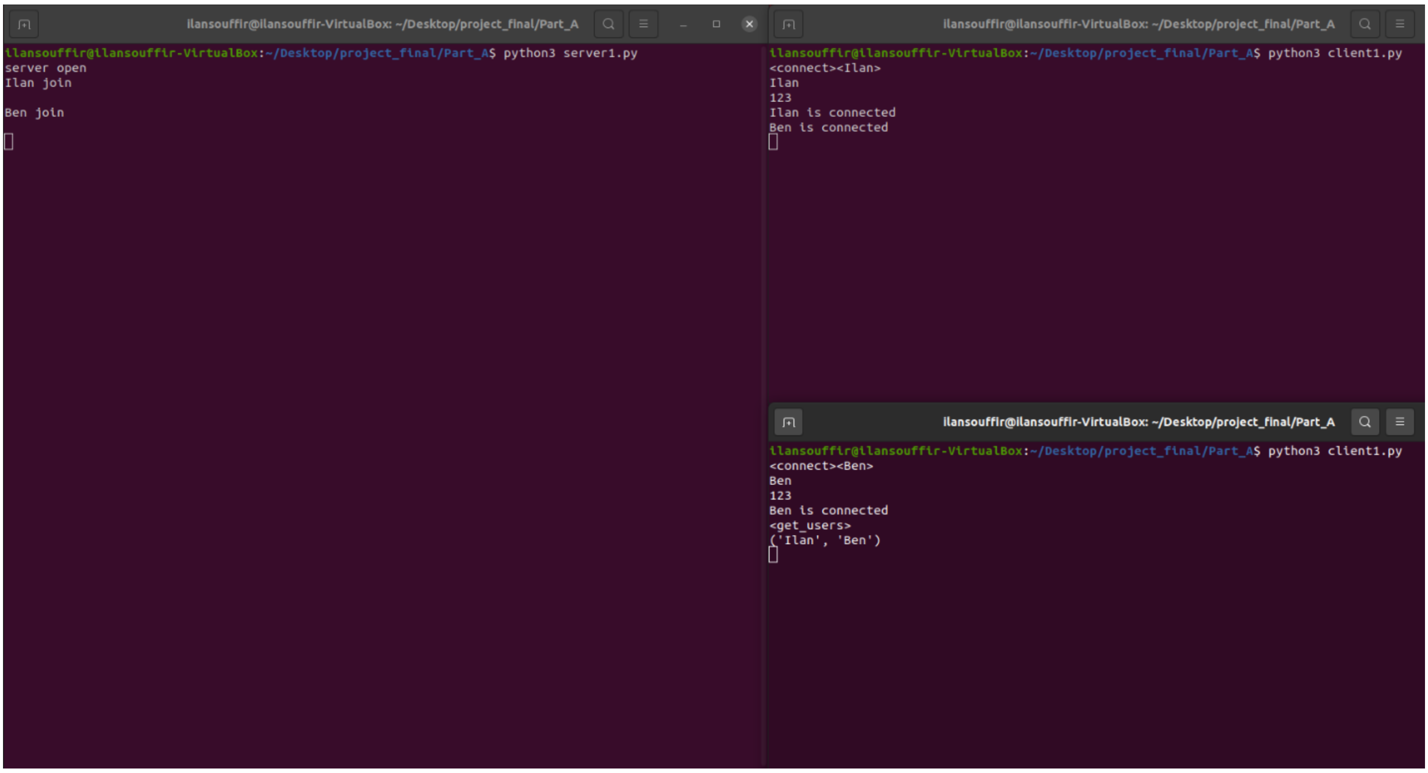
```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 server1.py
server open
Ilan join

Ben join
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 client1.py
<connect><Ilan>
Ilan
123
Ilan is connected
Ben is connected
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 client1.py
<connect><Ben>
Ben
123
Ben is connected
```

We can see on the server that Ilan can see that Ben is online now and now Ilan and Ben are connected to the same server, so they can communicate with each other.

5. Ben wants to know with whom he will be able to communicate so he wants to know the list of people connected on the server (on MSN) using the command <get_users> .


ilansouffir@ilansouffir-VirtualBox: ~/Desktop/project_final/Part_A
ilansouffir@ilansouffir-VirtualBox: ~/Desktop/project_final/Part_A
ilansouffir@ilansouffir-VirtualBox: ~/Desktop/project_final/Part_A

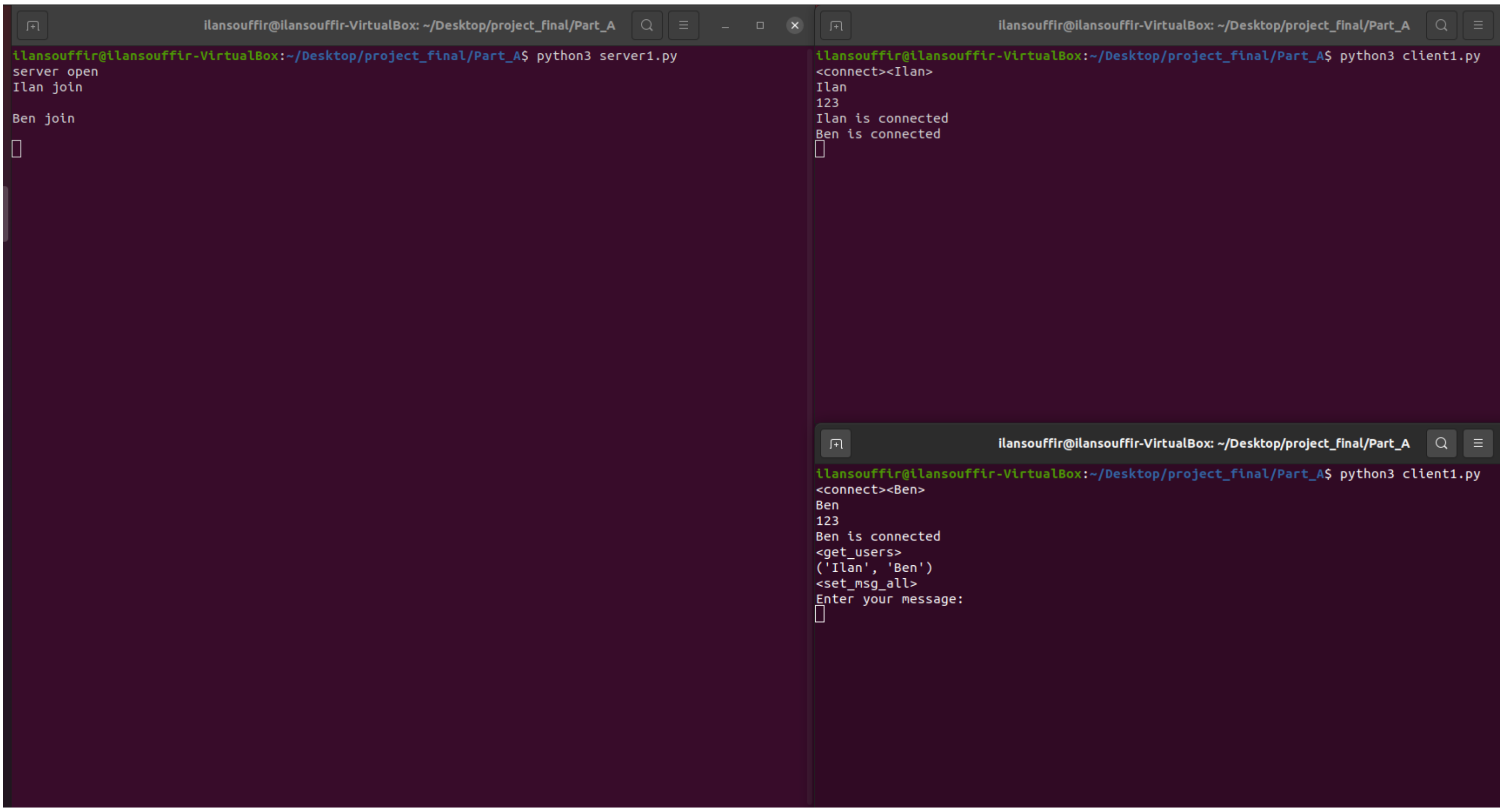
```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 server1.py
server open
Ilan join

Ben join
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 client1.py
<connect><Ilan>
Ilan
123
Ilan is connected
Ben is connected
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 client1.py
<connect><Ben>
Ben
123
Ben is connected
<get_users>
('Ilan', 'Ben')
```

We can see that there is only two users : Ilan and Ben.

6. Ben wants now to send a message to everyone that is in the messenger, so we use the command <set_msg_all> and then we are asked to « Enter your message » .

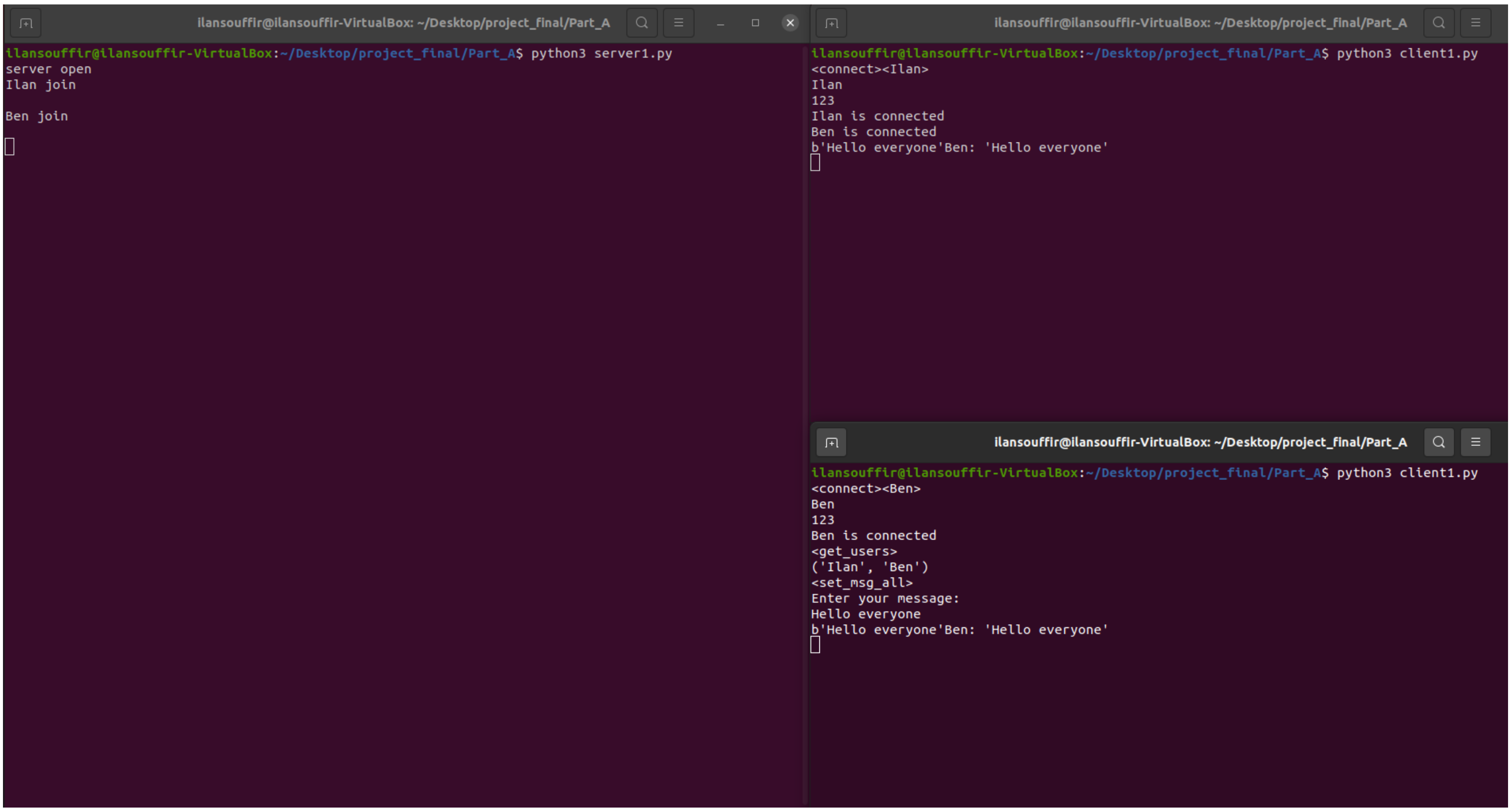


```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 server1.py
server open
Ilan join

Ben join
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 client1.py
<connect><Ilan>
Ilan
123
Ilan is connected
Ben is connected
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 client1.py
<connect><Ben>
Ben
123
Ben is connected
<get_users>
('Ilan', 'Ben')
<set_msg_all>
Enter your message:
```

7. Ben writes the message "Hello everyone" and sends it to everyone by clicking on enter.



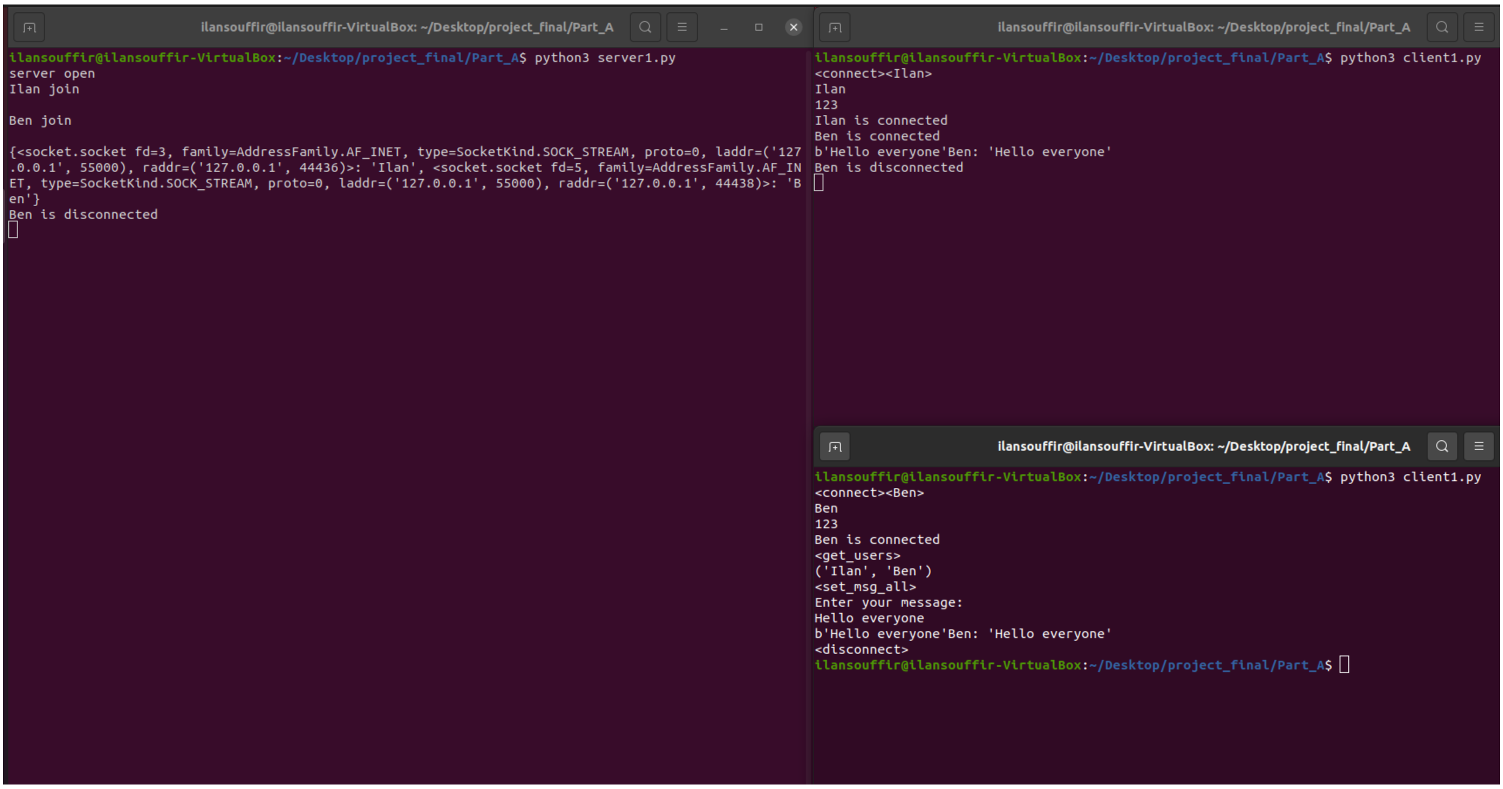
```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 server1.py
server open
Ilan join

Ben join
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 client1.py
<connect><Ilan>
Ilan
123
Ilan is connected
Ben is connected
b'Hello everyone'Ben: 'Hello everyone'
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 client1.py
<connect><Ben>
Ben
123
Ben is connected
<get_users>
('Ilan', 'Ben')
<set_msg_all>
Enter your message:
Hello everyone
b'Hello everyone'Ben: 'Hello everyone'
```

We can see that Ilan got the message from Ben.

8. Ben now wants to be offline so he disconnects from messenger using the command <disconnect>.


ilansouffir@ilansouffir-VirtualBox: ~/Desktop/project_final/Part_A
ilansouffir@ilansouffir-VirtualBox: ~/Desktop/project_final/Part_A
ilansouffir@ilansouffir-VirtualBox: ~/Desktop/project_final/Part_A

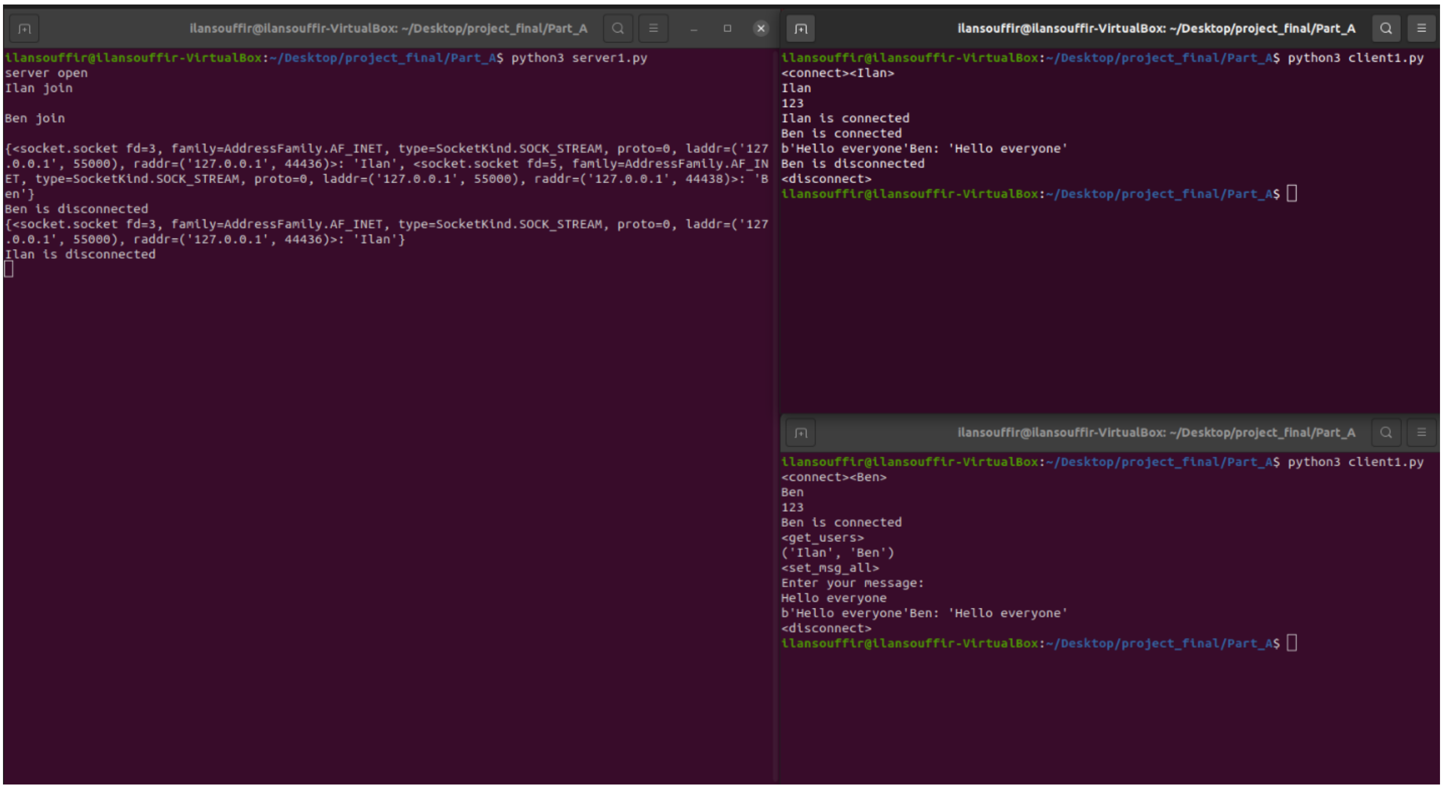
```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 server1.py
server open
Ilan join

Ben join

{<socket.socket fd=3, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127
.0.0.1', 55000), raddr=('127.0.0.1', 44436)>: 'Ilan', <socket.socket fd=5, family=AddressFamily.AF_IN
ET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 55000), raddr=('127.0.0.1', 44438)>: 'B
en'}
Ben is disconnected
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 client1.py
<connect><Ilan>
Ilan
123
Ilan is connected
Ben is connected
b'Hello everyone'Ben: 'Hello everyone'
Ben is disconnected
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 client1.py
<connect><Ben>
Ben
123
Ben is connected
<get_users>
('Ilan', 'Ben')
<set_msg_all>
Enter your message:
Hello everyone
b'Hello everyone'Ben: 'Hello everyone'
<disconnect>
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$
```

Ilan see that Ben is disconnect so he is now alone on messenger.

9. So Ilan wants to be offline too so he disconnects from messenger using the command <disconnect> too.

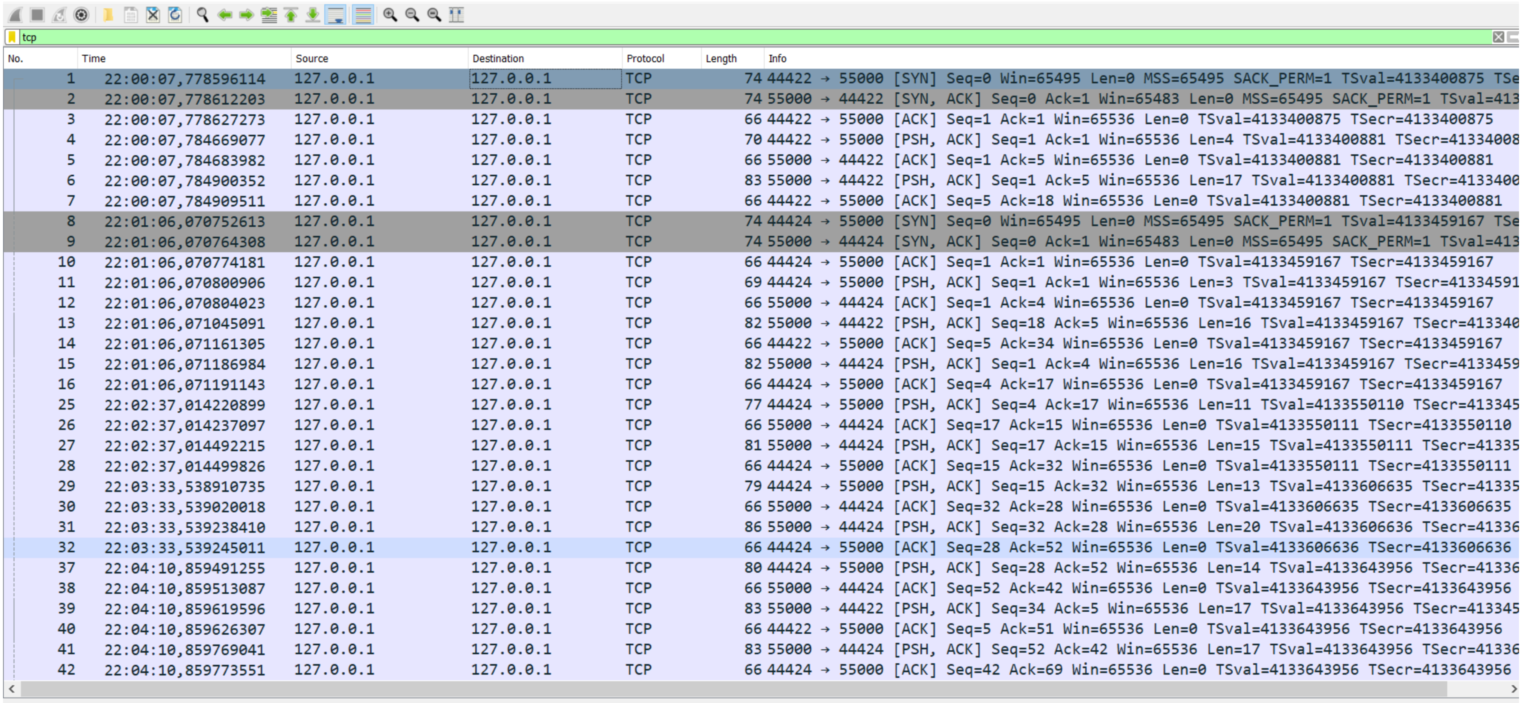

ilansouffir@ilansouffir-VirtualBox: ~/Desktop/project_final/Part_A
ilansouffir@ilansouffir-VirtualBox: ~/Desktop/project_final/Part_A
ilansouffir@ilansouffir-VirtualBox: ~/Desktop/project_final/Part_A

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 server1.py
server open
Ilan join

Ben join

{<socket.socket fd=3, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127
.0.0.1', 55000), raddr=('127.0.0.1', 44436)>: 'Ilan', <socket.socket fd=5, family=AddressFamily.AF_IN
ET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 55000), raddr=('127.0.0.1', 44438)>: 'B
en'}
Ben is disconnected
{<socket.socket fd=3, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127
.0.0.1', 55000), raddr=('127.0.0.1', 44436)>: 'Ilan'}
Ilan is disconnected
```

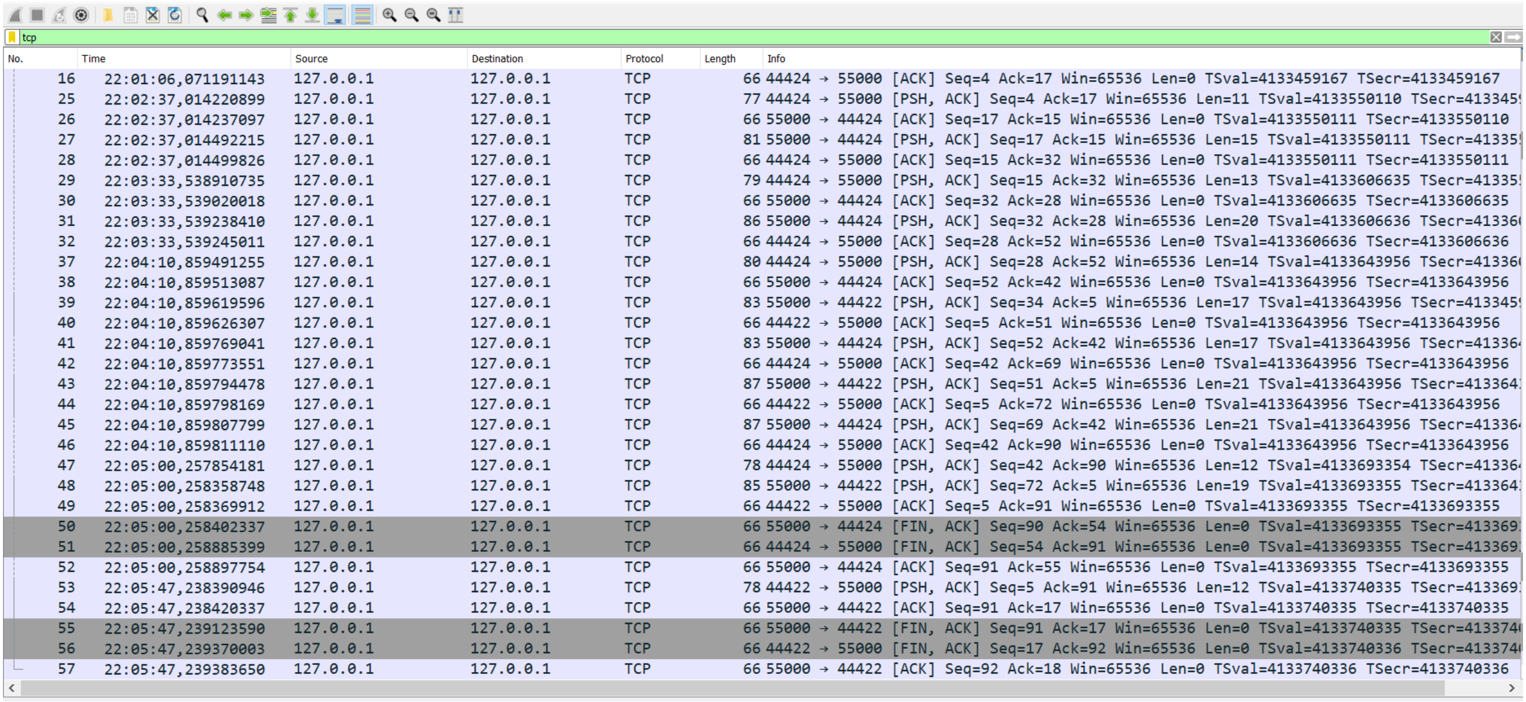
```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 client1.py
<connect><Ilan>
Ilan
123
Ilan is connected
Ben is connected
b'Hello everyone'Ben: 'Hello everyone'
Ben is disconnected
<disconnect>
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$ python3 client1.py
<connect><Ben>
Ben
123
Ben is connected
<get_users>
('Ilan', 'Ben')
<set_msg_all>
Enter your message:
Hello everyone
b'Hello everyone'Ben: 'Hello everyone'
<disconnect>
ilansouffir@ilansouffir-VirtualBox:~/Desktop/project_final/Part_A$
```

Capture on Wireshark of the example seen before (capture1.pcapng) :



```
tcp
No.   Time                Source      Destination  Protocol  Length  Info
  1   22:00:07,778596114  127.0.0.1   127.0.0.1    TCP       74 44422 → 55000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=4133400875 TSe
  2   22:00:07,778612203  127.0.0.1   127.0.0.1    TCP       74 55000 → 44422 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=413
  3   22:00:07,778627273  127.0.0.1   127.0.0.1    TCP       66 44422 → 55000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=4133400875 TSecr=4133400875
  4   22:00:07,784669077  127.0.0.1   127.0.0.1    TCP       70 44422 → 55000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=4 TSval=4133400881 TSecr=41334008
  5   22:00:07,784683982  127.0.0.1   127.0.0.1    TCP       66 55000 → 44422 [ACK] Seq=1 Ack=5 Win=65536 Len=0 TSval=4133400881 TSecr=4133400881
  6   22:00:07,784900352  127.0.0.1   127.0.0.1    TCP       83 55000 → 44422 [PSH, ACK] Seq=1 Ack=5 Win=65536 Len=17 TSval=4133400881 TSecr=4133400
  7   22:00:07,784909511  127.0.0.1   127.0.0.1    TCP       66 44422 → 55000 [ACK] Seq=5 Ack=18 Win=65536 Len=0 TSval=4133400881 TSecr=4133400881
  8   22:01:06,070752613  127.0.0.1   127.0.0.1    TCP       74 44424 → 55000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=4133459167 TSe
  9   22:01:06,070764308  127.0.0.1   127.0.0.1    TCP       74 55000 → 44424 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=413
 10   22:01:06,070774181  127.0.0.1   127.0.0.1    TCP       66 44424 → 55000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=4133459167 TSecr=4133459167
 11   22:01:06,070800906  127.0.0.1   127.0.0.1    TCP       69 44424 → 55000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=3 TSval=4133459167 TSecr=41334591
 12   22:01:06,070804023  127.0.0.1   127.0.0.1    TCP       66 55000 → 44424 [ACK] Seq=1 Ack=4 Win=65536 Len=0 TSval=4133459167 TSecr=4133459167
 13   22:01:06,071045091  127.0.0.1   127.0.0.1    TCP       82 55000 → 44422 [PSH, ACK] Seq=18 Ack=5 Win=65536 Len=16 TSval=4133459167 TSecr=413340
 14   22:01:06,071161305  127.0.0.1   127.0.0.1    TCP       66 44422 → 55000 [ACK] Seq=5 Ack=34 Win=65536 Len=0 TSval=4133459167 TSecr=4133459167
 15   22:01:06,071186984  127.0.0.1   127.0.0.1    TCP       82 55000 → 44424 [PSH, ACK] Seq=1 Ack=4 Win=65536 Len=16 TSval=4133459167 TSecr=4133459
 16   22:01:06,071191143  127.0.0.1   127.0.0.1    TCP       66 44424 → 55000 [ACK] Seq=4 Ack=17 Win=65536 Len=0 TSval=4133459167 TSecr=4133459167
 25   22:02:37,014220899  127.0.0.1   127.0.0.1    TCP       77 44424 → 55000 [PSH, ACK] Seq=4 Ack=17 Win=65536 Len=11 TSval=4133550110 TSecr=413345
 26   22:02:37,014237097  127.0.0.1   127.0.0.1    TCP       66 55000 → 44424 [ACK] Seq=17 Ack=15 Win=65536 Len=0 TSval=4133550111 TSecr=4133550110
 27   22:02:37,014492215  127.0.0.1   127.0.0.1    TCP       81 55000 → 44424 [PSH, ACK] Seq=17 Ack=15 Win=65536 Len=15 TSval=4133550111 TSecr=41335
 28   22:02:37,014499826  127.0.0.1   127.0.0.1    TCP       66 44424 → 55000 [ACK] Seq=15 Ack=32 Win=65536 Len=0 TSval=4133550111 TSecr=4133550111
 29   22:03:33,538910735  127.0.0.1   127.0.0.1    TCP       79 44424 → 55000 [PSH, ACK] Seq=15 Ack=32 Win=65536 Len=13 TSval=4133606635 TSecr=41335
 30   22:03:33,539020018  127.0.0.1   127.0.0.1    TCP       66 55000 → 44424 [ACK] Seq=32 Ack=28 Win=65536 Len=0 TSval=4133606635 TSecr=4133606635
 31   22:03:33,539238410  127.0.0.1   127.0.0.1    TCP       86 55000 → 44424 [PSH, ACK] Seq=32 Ack=28 Win=65536 Len=20 TSval=4133606636 TSecr=41336
 32   22:03:33,539245011  127.0.0.1   127.0.0.1    TCP       66 44424 → 55000 [ACK] Seq=28 Ack=52 Win=65536 Len=0 TSval=4133606636 TSecr=4133606636
 37   22:04:10,859491255  127.0.0.1   127.0.0.1    TCP       80 44424 → 55000 [PSH, ACK] Seq=28 Ack=52 Win=65536 Len=14 TSval=4133643956 TSecr=41336
 38   22:04:10,859513087  127.0.0.1   127.0.0.1    TCP       66 55000 → 44424 [ACK] Seq=52 Ack=42 Win=65536 Len=0 TSval=4133643956 TSecr=4133643956
 39   22:04:10,859619596  127.0.0.1   127.0.0.1    TCP       83 55000 → 44422 [PSH, ACK] Seq=34 Ack=5 Win=65536 Len=17 TSval=4133643956 TSecr=413345
 40   22:04:10,859626307  127.0.0.1   127.0.0.1    TCP       66 44422 → 55000 [ACK] Seq=5 Ack=51 Win=65536 Len=0 TSval=4133643956 TSecr=4133643956
 41   22:04:10,859769041  127.0.0.1   127.0.0.1    TCP       83 55000 → 44424 [PSH, ACK] Seq=52 Ack=42 Win=65536 Len=17 TSval=4133643956 TSecr=41336
 42   22:04:10,859773551  127.0.0.1   127.0.0.1    TCP       66 44424 → 55000 [ACK] Seq=42 Ack=69 Win=65536 Len=0 TSval=4133643956 TSecr=4133643956
```



```
tcp
No.   Time                Source      Destination  Protocol  Length  Info
 16   22:01:06,071191143  127.0.0.1   127.0.0.1    TCP       66 44424 → 55000 [ACK] Seq=4 Ack=17 Win=65536 Len=0 TSval=4133459167 TSecr=4133459167
 25   22:02:37,014220899  127.0.0.1   127.0.0.1    TCP       77 44424 → 55000 [PSH, ACK] Seq=4 Ack=17 Win=65536 Len=11 TSval=4133550110 TSecr=413345
 26   22:02:37,014237097  127.0.0.1   127.0.0.1    TCP       66 55000 → 44424 [ACK] Seq=17 Ack=15 Win=65536 Len=0 TSval=4133550111 TSecr=4133550110
 27   22:02:37,014492215  127.0.0.1   127.0.0.1    TCP       81 55000 → 44424 [PSH, ACK] Seq=17 Ack=15 Win=65536 Len=15 TSval=4133550111 TSecr=41335
 28   22:02:37,014499826  127.0.0.1   127.0.0.1    TCP       66 44424 → 55000 [ACK] Seq=15 Ack=32 Win=65536 Len=0 TSval=4133550111 TSecr=4133550111
 29   22:03:33,538910735  127.0.0.1   127.0.0.1    TCP       79 44424 → 55000 [PSH, ACK] Seq=15 Ack=32 Win=65536 Len=13 TSval=4133606635 TSecr=41335
 30   22:03:33,539020018  127.0.0.1   127.0.0.1    TCP       66 55000 → 44424 [ACK] Seq=32 Ack=28 Win=65536 Len=0 TSval=4133606635 TSecr=4133606635
 31   22:03:33,539238410  127.0.0.1   127.0.0.1    TCP       86 55000 → 44424 [PSH, ACK] Seq=32 Ack=28 Win=65536 Len=20 TSval=4133606636 TSecr=41336
 32   22:03:33,539245011  127.0.0.1   127.0.0.1    TCP       66 44424 → 55000 [ACK] Seq=28 Ack=52 Win=65536 Len=0 TSval=4133606636 TSecr=4133606636
 37   22:04:10,859491255  127.0.0.1   127.0.0.1    TCP       80 44424 → 55000 [PSH, ACK] Seq=28 Ack=52 Win=65536 Len=14 TSval=4133643956 TSecr=41336
 38   22:04:10,859513087  127.0.0.1   127.0.0.1    TCP       66 55000 → 44424 [ACK] Seq=52 Ack=42 Win=65536 Len=0 TSval=4133643956 TSecr=4133643956
 39   22:04:10,859619596  127.0.0.1   127.0.0.1    TCP       83 55000 → 44422 [PSH, ACK] Seq=34 Ack=5 Win=65536 Len=17 TSval=4133643956 TSecr=413345
 40   22:04:10,859626307  127.0.0.1   127.0.0.1    TCP       66 44422 → 55000 [ACK] Seq=5 Ack=51 Win=65536 Len=0 TSval=4133643956 TSecr=4133643956
 41   22:04:10,859769041  127.0.0.1   127.0.0.1    TCP       83 55000 → 44424 [PSH, ACK] Seq=52 Ack=42 Win=65536 Len=17 TSval=4133643956 TSecr=41336
 42   22:04:10,859773551  127.0.0.1   127.0.0.1    TCP       66 44424 → 55000 [ACK] Seq=42 Ack=69 Win=65536 Len=0 TSval=4133643956 TSecr=4133643956
 43   22:04:10,859794478  127.0.0.1   127.0.0.1    TCP       87 55000 → 44422 [PSH, ACK] Seq=51 Ack=5 Win=65536 Len=21 TSval=4133643956 TSecr=41336
 44   22:04:10,859798169  127.0.0.1   127.0.0.1    TCP       66 44422 → 55000 [ACK] Seq=5 Ack=72 Win=65536 Len=0 TSval=4133643956 TSecr=4133643956
 45   22:04:10,859807799  127.0.0.1   127.0.0.1    TCP       87 55000 → 44424 [PSH, ACK] Seq=69 Ack=42 Win=65536 Len=21 TSval=4133643956 TSecr=41336
 46   22:04:10,859811110  127.0.0.1   127.0.0.1    TCP       66 44424 → 55000 [ACK] Seq=42 Ack=90 Win=65536 Len=0 TSval=4133643956 TSecr=4133643956
 47   22:05:00,257854181  127.0.0.1   127.0.0.1    TCP       78 44424 → 55000 [PSH, ACK] Seq=42 Ack=90 Win=65536 Len=12 TSval=4133693354 TSecr=41336
 48   22:05:00,258358748  127.0.0.1   127.0.0.1    TCP       85 55000 → 44422 [PSH, ACK] Seq=72 Ack=5 Win=65536 Len=19 TSval=4133693355 TSecr=41336
 49   22:05:00,258369912  127.0.0.1   127.0.0.1    TCP       66 44422 → 55000 [ACK] Seq=5 Ack=91 Win=65536 Len=0 TSval=4133693355 TSecr=4133693355
 50   22:05:00,258402337  127.0.0.1   127.0.0.1    TCP       66 55000 → 44424 [FIN, ACK] Seq=90 Ack=54 Win=65536 Len=0 TSval=4133693355 TSecr=41336
 51   22:05:00,258885399  127.0.0.1   127.0.0.1    TCP       66 44424 → 55000 [FIN, ACK] Seq=54 Ack=91 Win=65536 Len=0 TSval=4133693355 TSecr=41336
 52   22:05:00,258897754  127.0.0.1   127.0.0.1    TCP       66 55000 → 44424 [ACK] Seq=91 Ack=55 Win=65536 Len=0 TSval=4133693355 TSecr=4133693355
 53   22:05:47,238390946  127.0.0.1   127.0.0.1    TCP       78 44422 → 55000 [PSH, ACK] Seq=5 Ack=91 Win=65536 Len=12 TSval=4133740335 TSecr=41336
 54   22:05:47,238420337  127.0.0.1   127.0.0.1    TCP       66 55000 → 44422 [ACK] Seq=91 Ack=17 Win=65536 Len=0 TSval=4133740335 TSecr=4133740335
 55   22:05:47,239123590  127.0.0.1   127.0.0.1    TCP       66 55000 → 44422 [FIN, ACK] Seq=91 Ack=17 Win=65536 Len=0 TSval=4133740335 TSecr=413374
 56   22:05:47,239370003  127.0.0.1   127.0.0.1    TCP       66 44422 → 55000 [FIN, ACK] Seq=17 Ack=92 Win=65536 Len=0 TSval=4133740336 TSecr=413374
 57   22:05:47,239383650  127.0.0.1   127.0.0.1    TCP       66 55000 → 44422 [ACK] Seq=92 Ack=18 Win=65536 Len=0 TSval=4133740336 TSecr=4133740336
```

Lines 1-7 : Ilan connects to messenger.

Lines 8-16 : Ben connects to messenger.

Lines 25-28 : command <get_users> from Ben.

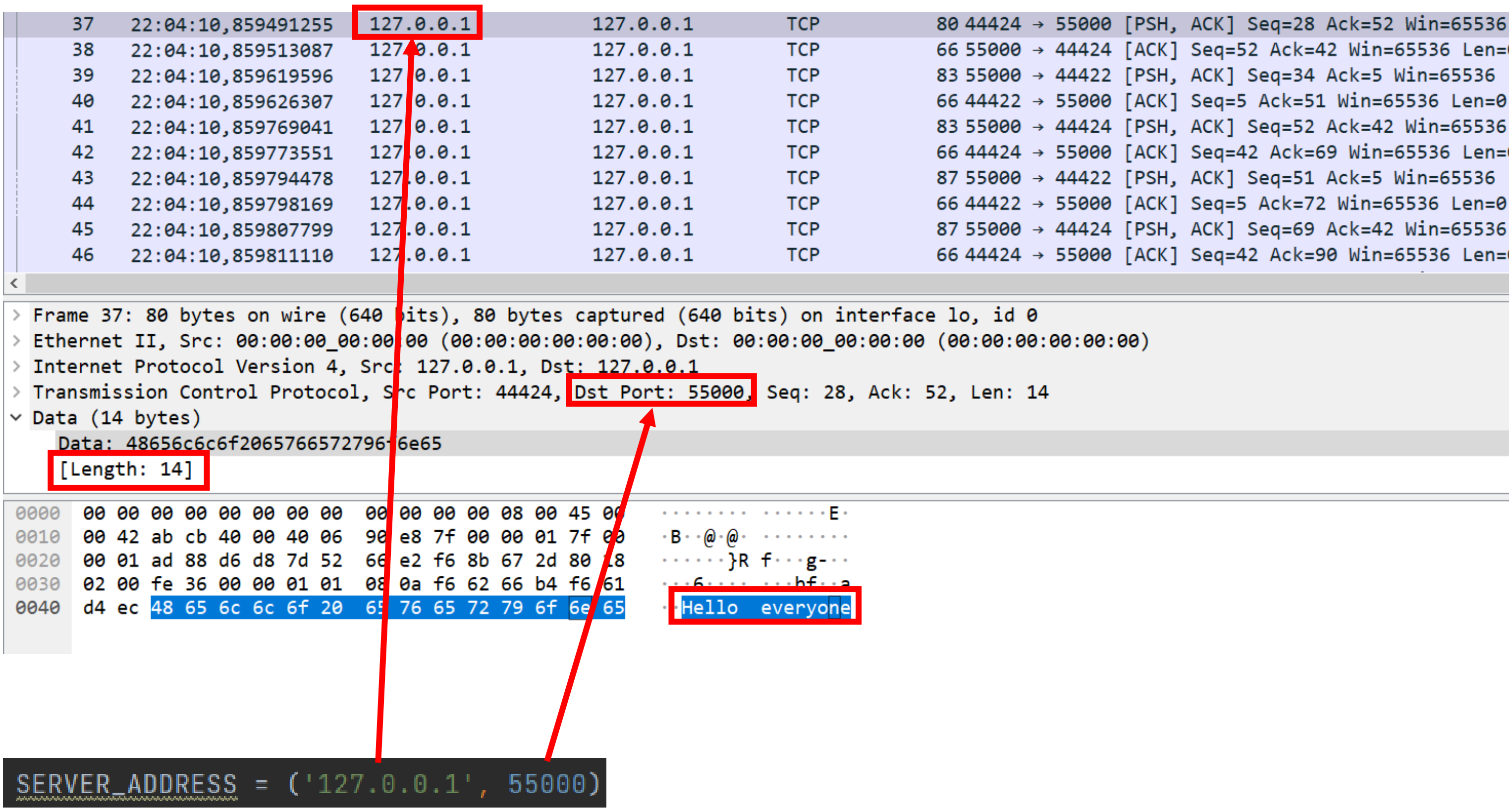
Lines 29-32 : command <set_msg_all> from Ben.

Lines 37-46 : Ben sends « Hello everyone ».

Lines 47-52 : Ben go to offline using command <disconnect>

Lines 53-57 : Ilan go to offline too with the same command.

We will dissect with precision the example of when Ben sends the message "Hello everyone" to everyone. (lines 37-46).



```
37   22:04:10,859491255   127.0.0.1   127.0.0.1   TCP   80 44424 → 55000 [PSH, ACK] Seq=28 Ack=52 Win=65536
38   22:04:10,859513087   127.0.0.1   127.0.0.1   TCP   66 55000 → 44424 [ACK] Seq=52 Ack=42 Win=65536 Len=
39   22:04:10,859619596   127.0.0.1   127.0.0.1   TCP   83 55000 → 44422 [PSH, ACK] Seq=34 Ack=5 Win=65536
40   22:04:10,859626307   127.0.0.1   127.0.0.1   TCP   66 44422 → 55000 [ACK] Seq=5 Ack=51 Win=65536 Len=0
41   22:04:10,859769041   127.0.0.1   127.0.0.1   TCP   83 55000 → 44424 [PSH, ACK] Seq=52 Ack=42 Win=65536
42   22:04:10,859773551   127.0.0.1   127.0.0.1   TCP   66 44424 → 55000 [ACK] Seq=42 Ack=69 Win=65536 Len=
43   22:04:10,859794478   127.0.0.1   127.0.0.1   TCP   87 55000 → 44422 [PSH, ACK] Seq=51 Ack=5 Win=65536
44   22:04:10,859798169   127.0.0.1   127.0.0.1   TCP   66 44422 → 55000 [ACK] Seq=5 Ack=72 Win=65536 Len=0
45   22:04:10,859807799   127.0.0.1   127.0.0.1   TCP   87 55000 → 44424 [PSH, ACK] Seq=69 Ack=42 Win=65536
46   22:04:10,859811110   127.0.0.1   127.0.0.1   TCP   66 44424 → 55000 [ACK] Seq=42 Ack=90 Win=65536 Len=
```

```
> Frame 37: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface lo, id 0
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 44424, Dst Port: 55000, Seq: 28, Ack: 52, Len: 14
v Data (14 bytes)
    Data: 48656c6c6f2065766572796f6e65
    [Length: 14]
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 42 ab cb 40 00 40 06  90 e8 7f 00 00 01 7f 00   ·B··@·@· ········
0020  00 01 ad 88 d6 d8 7d 52  66 e2 f6 8b 67 2d 80 18   ······}R f···g-··
0030  02 00 fe 36 00 00 01 01  08 0a f6 62 66 b4 f6 61   ··6····· ···bf··a
0040  d4 ec 48 65 6c 6c 6f 20  65 76 65 72 79 6f 6e 65   ··Hello  everyone
```

```
SERVER_ADDRESS = ('127.0.0.1', 55000)
```

Line 37 : Ben write a message to all the clients saying « Hello everyone ». We can see that the length of the message is 14 (5 letters for the first word + 1 space + 8 letters for the second word).

Line 38 : the server sent an ACK back to Ben.

Line 39 : the server sent to Ilan the message that he sent as the broadcast message to the user Ilan that is connected to the server at the same time.

Line 40 : Ilan sent an ACK back to the server.

Line 41 : the server sent the message to Ben saying « Hello everyone ».

Line 42 : Ben sent an ACK back to the server.

Line 43 : the server sent the message to Ilan saying « Ben : Hello everyone ».

Line 44 : Ilan sent an ACK back to the server.

Line 45 : the server sent the message to Ben saying « Ben : Hello everyone ».

Line 46 : Ben sent an ACK back to the server.

חלק ב

In this part, we were asked to build in python with the code from part A : to add a new layer for transferring files over UDP which is called FAST reliable UDP
And then we need to answer the questions :
1 - How works the system overcomes packet loss ?
2 - How the system overcomes latency problems ?
3 – Diagram of situations in which the system.
We created 2 classes : server1.py and client1.py to solve the requested exercise.

1 :

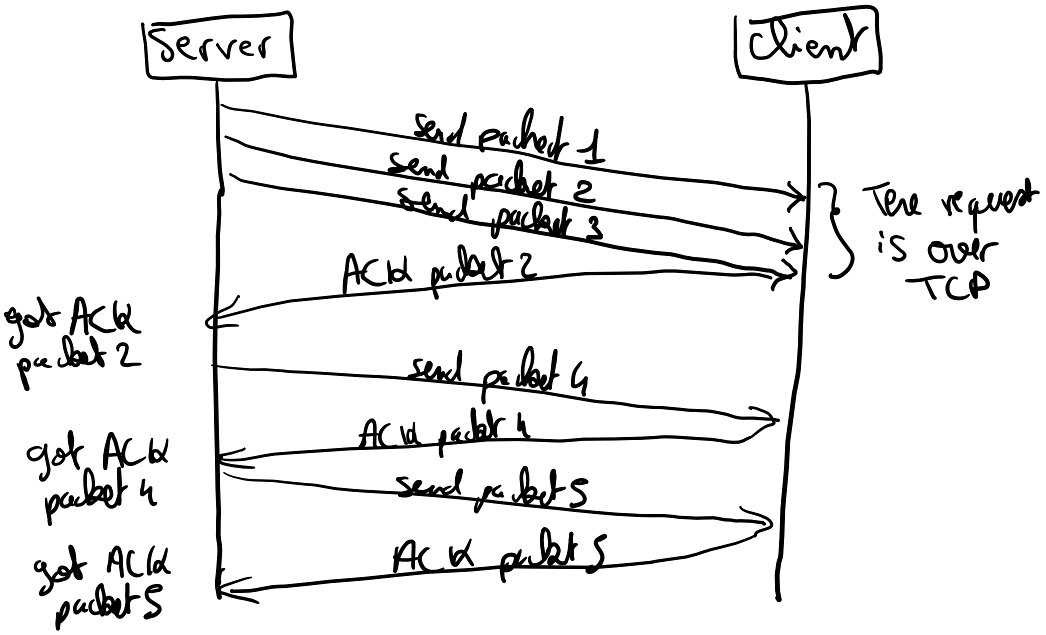
In our python code system, we are using 2 protocols : TCP and UDP.

We saw in Part A that we need TCP protocol when a user (client) wants to connect to the server messenger and then he wants to send a message to everyone or for a particular user in the messenger. In that case, if there is a packet loss, the sender resends the packet and slows down the sending of data over the network by reducing
the size of the message to send. If the sender has a packet loss, he resends the packets from the first packet that didn't receive an ack. If a user wants a file download, we are using the UDP protocol. But, we implement a RDT over UDP in this case

2 :

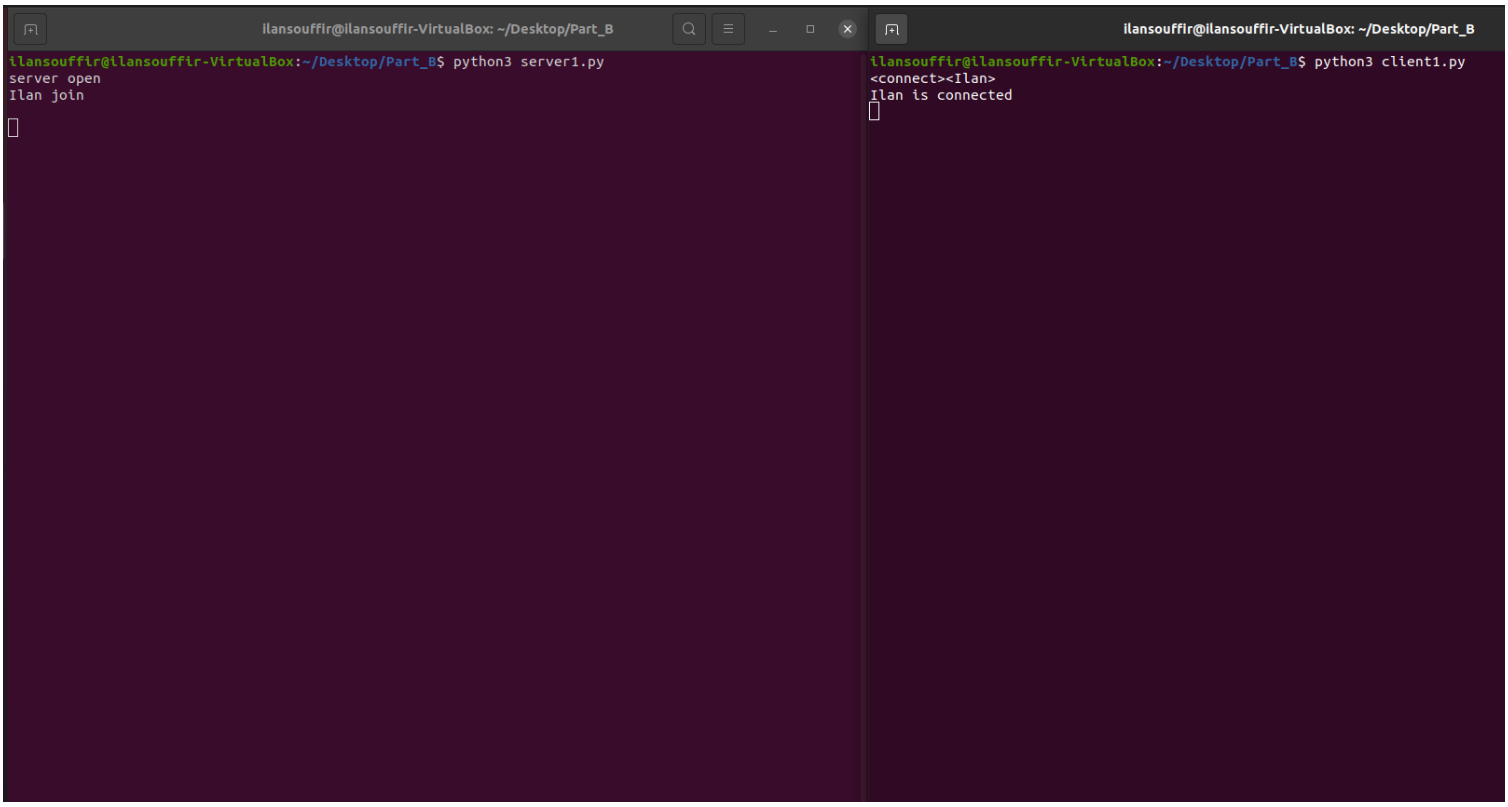
For the latency problems, we used a go-back-n RDT algorithm, we have N packets from the last packet that we received, if all the acks were received, we will send n new packets.
For the user, we send in the order we received them and the server will start sending after the first duplicate ack.

3 :



How to Run ?

1. <connect><Ilan> : create a new client with name Ilan.



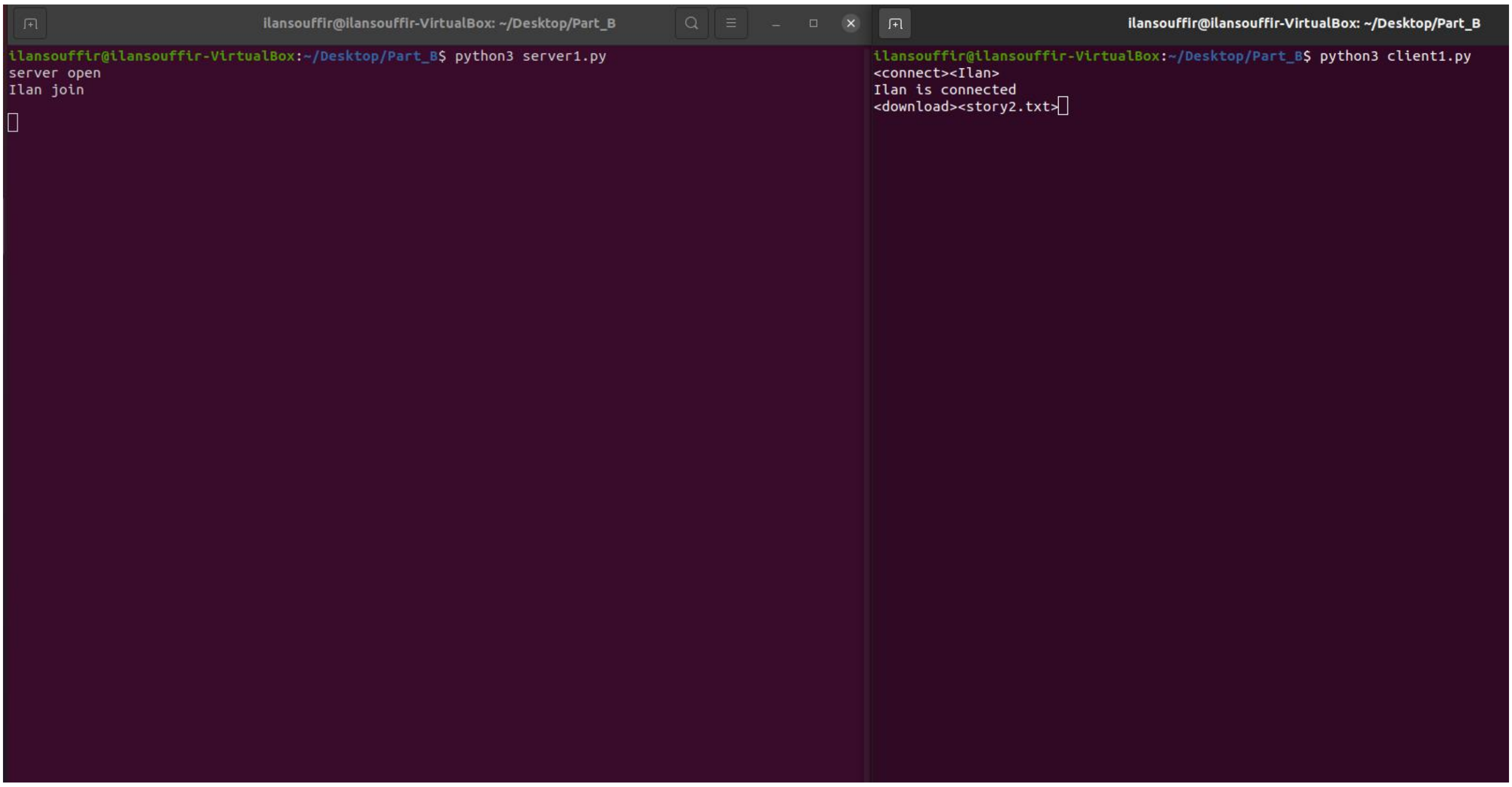
ilansouffir@ilansouffir-VirtualBox: ~/Desktop/Part_B

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/Part_B$ python3 server1.py
server open
Ilan join
```

ilansouffir@ilansouffir-VirtualBox: ~/Desktop/Part_B

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/Part_B$ python3 client1.py
<connect><Ilan>
Ilan is connected
```

2. <download><story2.txt> to download our file.txt



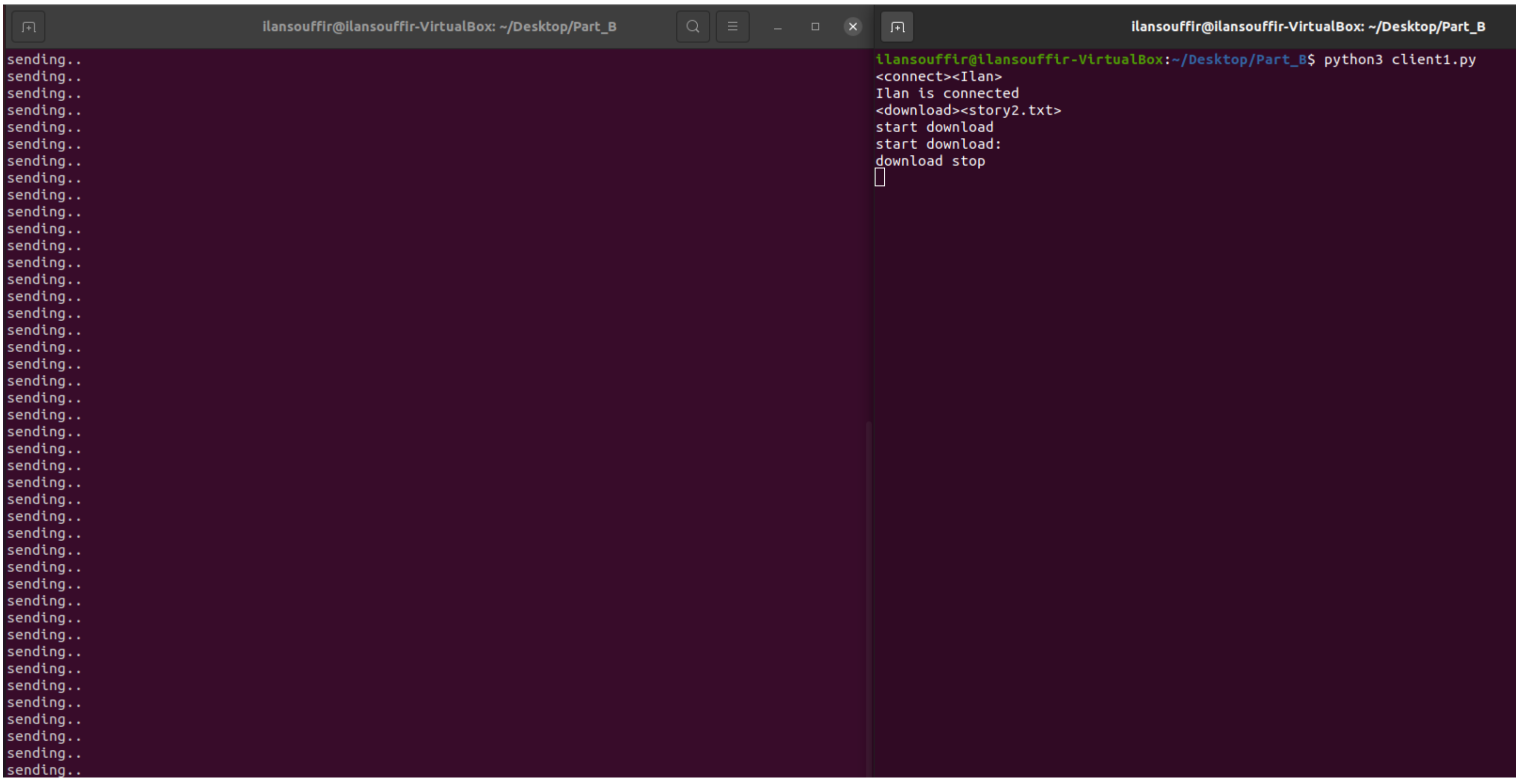
ilansouffir@ilansouffir-VirtualBox: ~/Desktop/Part_B

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/Part_B$ python3 server1.py
server open
Ilan join
```

ilansouffir@ilansouffir-VirtualBox: ~/Desktop/Part_B

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/Part_B$ python3 client1.py
<connect><Ilan>
Ilan is connected
<download><story2.txt>
```

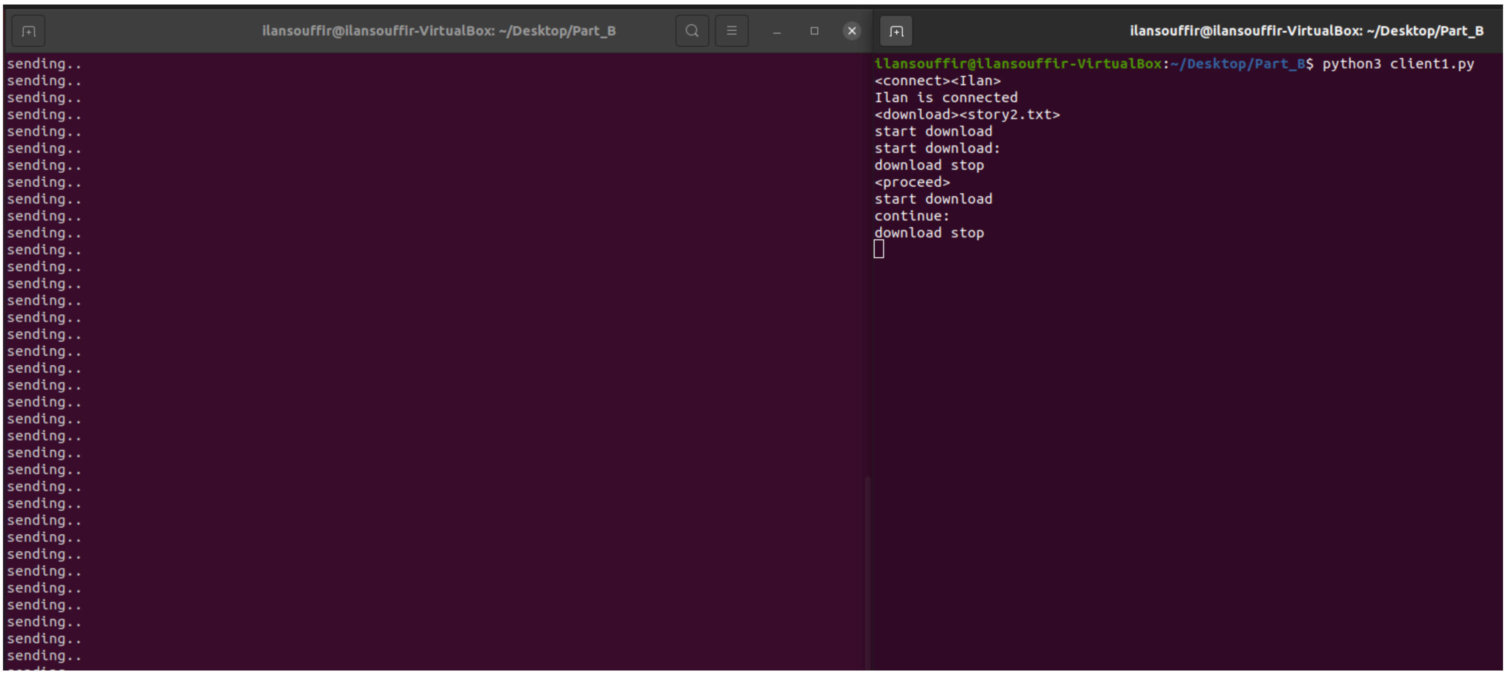
3. we can see that on the server the file is sending.


ilansouffir@ilansouffir-VirtualBox: ~/Desktop/Part_B
ilansouffir@ilansouffir-VirtualBox: ~/Desktop/Part_B

```
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/Part_B$ python3 client1.py
<connect><Ilan>
Ilan is connected
<download><story2.txt>
start download
start download:
download stop
```

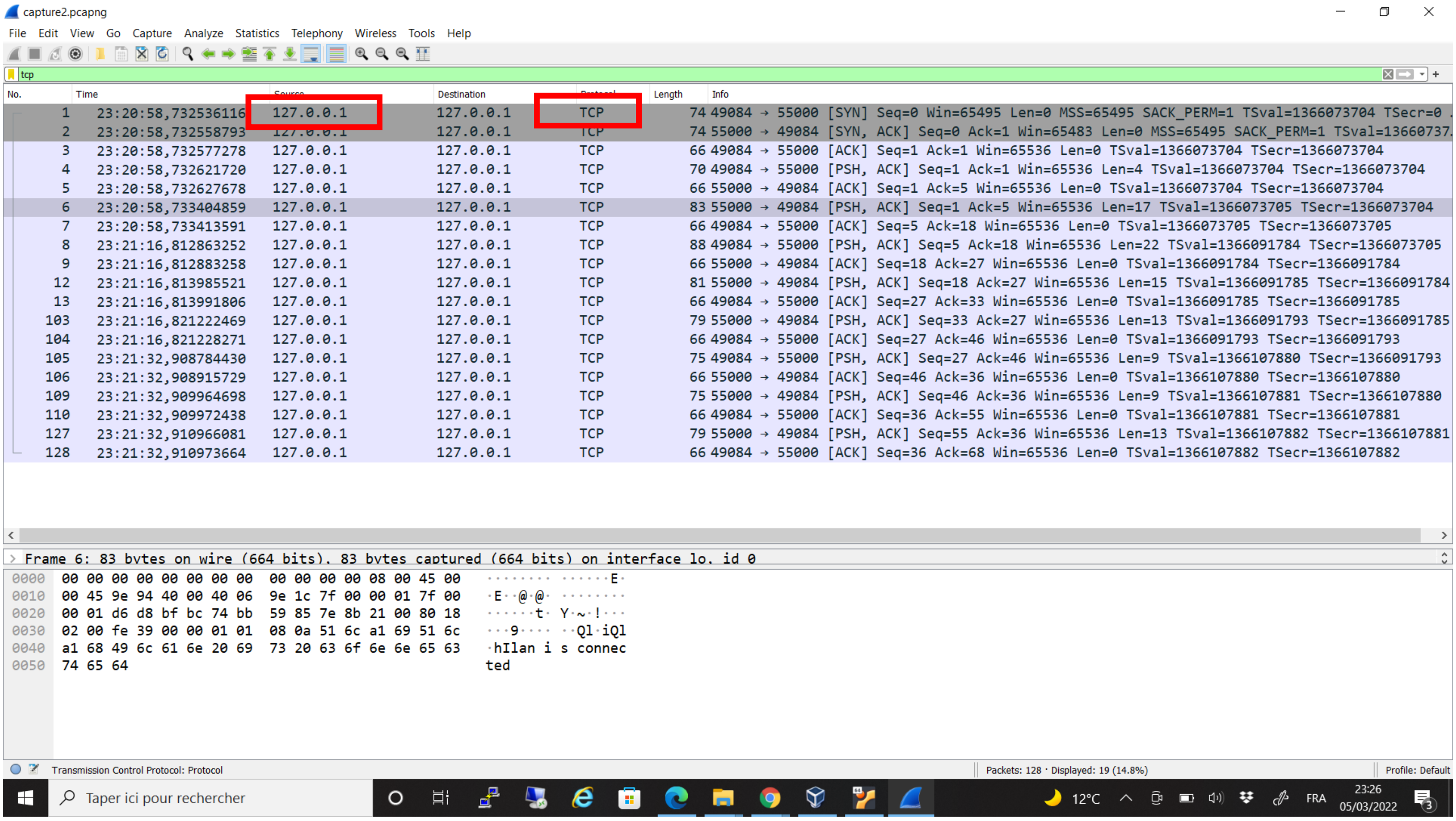
4.<proceed>


ilansouffir@ilansouffir-VirtualBox: ~/Desktop/Part_B
ilansouffir@ilansouffir-VirtualBox: ~/Desktop/Part_B

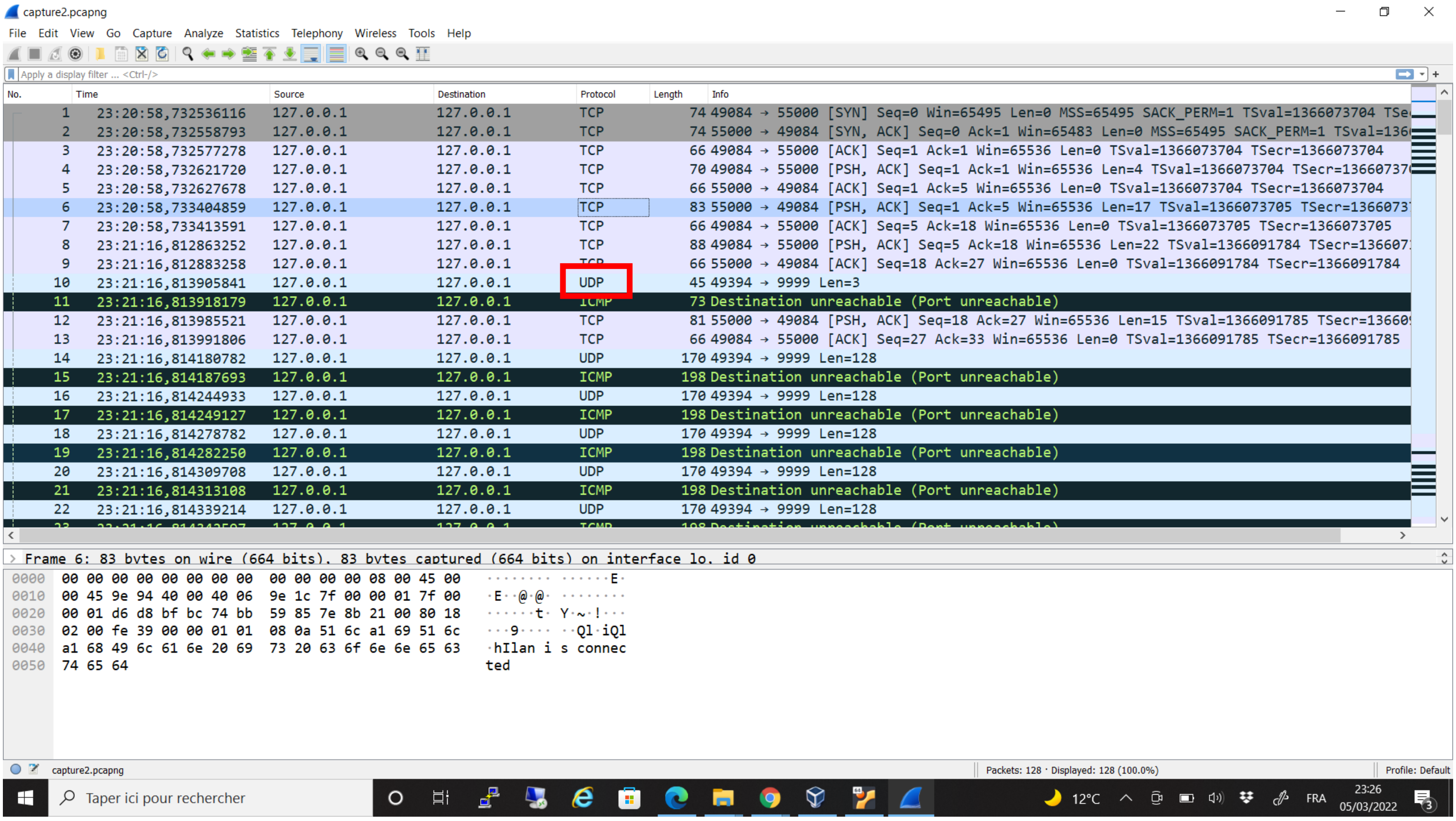
```
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
sending..
```

```
ilansouffir@ilansouffir-VirtualBox:~/Desktop/Part_B$ python3 client1.py
<connect><Ilan>
Ilan is connected
<download><story2.txt>
start download
start download:
download stop
<proceed>
start download
continue:
download stop
```

<u>Capture on Wireshark of the example seen before (capture2.pcapng) :</u>


capture2.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp
Taper ici pour rechercher
12°C
FRA
23:26
05/03/2022

```
No.   Time                Source      Destination  Protocol  Length  Info
  1   23:20:58,732536116  127.0.0.1   127.0.0.1    TCP       74 49084 → 55000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1366073704 TSecr=0 .
  2   23:20:58,732558793  127.0.0.1   127.0.0.1    TCP       74 55000 → 49084 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1366073737.
  3   23:20:58,732577278  127.0.0.1   127.0.0.1    TCP       66 49084 → 55000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1366073704 TSecr=1366073704
  4   23:20:58,732621720  127.0.0.1   127.0.0.1    TCP       70 49084 → 55000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=4 TSval=1366073704 TSecr=1366073704
  5   23:20:58,732627678  127.0.0.1   127.0.0.1    TCP       66 55000 → 49084 [ACK] Seq=1 Ack=5 Win=65536 Len=0 TSval=1366073704 TSecr=1366073704
  6   23:20:58,733404859  127.0.0.1   127.0.0.1    TCP       83 55000 → 49084 [PSH, ACK] Seq=1 Ack=5 Win=65536 Len=17 TSval=1366073705 TSecr=1366073704
  7   23:20:58,733413591  127.0.0.1   127.0.0.1    TCP       66 49084 → 55000 [ACK] Seq=5 Ack=18 Win=65536 Len=0 TSval=1366073705 TSecr=1366073705
  8   23:21:16,812863252  127.0.0.1   127.0.0.1    TCP       88 49084 → 55000 [PSH, ACK] Seq=5 Ack=18 Win=65536 Len=22 TSval=1366091784 TSecr=1366073705
  9   23:21:16,812883258  127.0.0.1   127.0.0.1    TCP       66 55000 → 49084 [ACK] Seq=18 Ack=27 Win=65536 Len=0 TSval=1366091784 TSecr=1366091784
 12   23:21:16,813985521  127.0.0.1   127.0.0.1    TCP       81 55000 → 49084 [PSH, ACK] Seq=18 Ack=27 Win=65536 Len=15 TSval=1366091785 TSecr=1366091784
 13   23:21:16,813991806  127.0.0.1   127.0.0.1    TCP       66 49084 → 55000 [ACK] Seq=27 Ack=33 Win=65536 Len=0 TSval=1366091785 TSecr=1366091785
103   23:21:16,821222469  127.0.0.1   127.0.0.1    TCP       79 55000 → 49084 [PSH, ACK] Seq=33 Ack=27 Win=65536 Len=13 TSval=1366091793 TSecr=1366091785
104   23:21:16,821228271  127.0.0.1   127.0.0.1    TCP       66 49084 → 55000 [ACK] Seq=27 Ack=46 Win=65536 Len=0 TSval=1366091793 TSecr=1366091793
105   23:21:32,908784430  127.0.0.1   127.0.0.1    TCP       75 49084 → 55000 [PSH, ACK] Seq=27 Ack=46 Win=65536 Len=9 TSval=1366107880 TSecr=1366091793
106   23:21:32,908915729  127.0.0.1   127.0.0.1    TCP       66 55000 → 49084 [ACK] Seq=46 Ack=36 Win=65536 Len=0 TSval=1366107880 TSecr=1366107880
109   23:21:32,909964698  127.0.0.1   127.0.0.1    TCP       75 55000 → 49084 [PSH, ACK] Seq=46 Ack=36 Win=65536 Len=9 TSval=1366107881 TSecr=1366107880
110   23:21:32,909972438  127.0.0.1   127.0.0.1    TCP       66 49084 → 55000 [ACK] Seq=36 Ack=55 Win=65536 Len=0 TSval=1366107881 TSecr=1366107881
127   23:21:32,910966081  127.0.0.1   127.0.0.1    TCP       79 55000 → 49084 [PSH, ACK] Seq=55 Ack=36 Win=65536 Len=13 TSval=1366107882 TSecr=1366107881
128   23:21:32,910973664  127.0.0.1   127.0.0.1    TCP       66 49084 → 55000 [ACK] Seq=36 Ack=68 Win=65536 Len=0 TSval=1366107882 TSecr=1366107882

> Frame 6: 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface lo, id 0

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 45 9e 94 40 00 40 06  9e 1c 7f 00 00 01 7f 00   ·E··@·@· ········
0020  00 01 d6 d8 bf bc 74 bb  59 85 7e 8b 21 00 80 18   ······t· Y·~·!···
0030  02 00 fe 39 00 00 01 01  08 0a 51 6c a1 69 51 6c   ···9···· ··Ql·iQl
0040  a1 68 49 6c 61 6e 20 69  73 20 63 6f 6e 6e 65 63   ·hIlan i s connec
0050  74 65 64                                           ted

Transmission Control Protocol: Protocol     Packets: 128 · Displayed: 19 (14.8%)     Profile: Default
```


capture2.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Taper ici pour rechercher
12°C
FRA
23:26
05/03/2022

```
No.   Time                Source      Destination  Protocol  Length  Info
  1   23:20:58,732536116  127.0.0.1   127.0.0.1    TCP       74 49084 → 55000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1366073704 TSe
  2   23:20:58,732558793  127.0.0.1   127.0.0.1    TCP       74 55000 → 49084 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=136
  3   23:20:58,732577278  127.0.0.1   127.0.0.1    TCP       66 49084 → 55000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1366073704 TSecr=1366073704
  4   23:20:58,732621720  127.0.0.1   127.0.0.1    TCP       70 49084 → 55000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=4 TSval=1366073704 TSecr=136607370
  5   23:20:58,732627678  127.0.0.1   127.0.0.1    TCP       66 55000 → 49084 [ACK] Seq=1 Ack=5 Win=65536 Len=0 TSval=1366073704 TSecr=1366073704
  6   23:20:58,733404859  127.0.0.1   127.0.0.1    TCP       83 55000 → 49084 [PSH, ACK] Seq=1 Ack=5 Win=65536 Len=17 TSval=1366073705 TSecr=136607370
  7   23:20:58,733413591  127.0.0.1   127.0.0.1    TCP       66 49084 → 55000 [ACK] Seq=5 Ack=18 Win=65536 Len=0 TSval=1366073705 TSecr=1366073705
  8   23:21:16,812863252  127.0.0.1   127.0.0.1    TCP       88 49084 → 55000 [PSH, ACK] Seq=5 Ack=18 Win=65536 Len=22 TSval=1366091784 TSecr=136607
  9   23:21:16,812883258  127.0.0.1   127.0.0.1    TCP       66 55000 → 49084 [ACK] Seq=18 Ack=27 Win=65536 Len=0 TSval=1366091784 TSecr=1366091784
 10   23:21:16,813905841  127.0.0.1   127.0.0.1    UDP       45 49394 → 9999 Len=3
 11   23:21:16,813918179  127.0.0.1   127.0.0.1    ICMP      73 Destination unreachable (Port unreachable)
 12   23:21:16,813985521  127.0.0.1   127.0.0.1    TCP       81 55000 → 49084 [PSH, ACK] Seq=18 Ack=27 Win=65536 Len=15 TSval=1366091785 TSecr=13660
 13   23:21:16,813991806  127.0.0.1   127.0.0.1    TCP       66 49084 → 55000 [ACK] Seq=27 Ack=33 Win=65536 Len=0 TSval=1366091785 TSecr=1366091785
 14   23:21:16,814180782  127.0.0.1   127.0.0.1    UDP      170 49394 → 9999 Len=128
 15   23:21:16,814187693  127.0.0.1   127.0.0.1    ICMP     198 Destination unreachable (Port unreachable)
 16   23:21:16,814244933  127.0.0.1   127.0.0.1    UDP      170 49394 → 9999 Len=128
 17   23:21:16,814249127  127.0.0.1   127.0.0.1    ICMP     198 Destination unreachable (Port unreachable)
 18   23:21:16,814278782  127.0.0.1   127.0.0.1    UDP      170 49394 → 9999 Len=128
 19   23:21:16,814282250  127.0.0.1   127.0.0.1    ICMP     198 Destination unreachable (Port unreachable)
 20   23:21:16,814309708  127.0.0.1   127.0.0.1    UDP      170 49394 → 9999 Len=128
 21   23:21:16,814313108  127.0.0.1   127.0.0.1    ICMP     198 Destination unreachable (Port unreachable)
 22   23:21:16,814339214  127.0.0.1   127.0.0.1    UDP      170 49394 → 9999 Len=128

> Frame 6: 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface lo, id 0

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 45 9e 94 40 00 40 06  9e 1c 7f 00 00 01 7f 00   ·E··@·@· ········
0020  00 01 d6 d8 bf bc 74 bb  59 85 7e 8b 21 00 80 18   ······t· Y·~·!···
0030  02 00 fe 39 00 00 01 01  08 0a 51 6c a1 69 51 6c   ···9···· ··Ql·iQl
0040  a1 68 49 6c 61 6e 20 69  73 20 63 6f 6e 6e 65 63   ·hIlan i s connec
0050  74 65 64                                           ted

capture2.pcapng     Packets: 128 · Displayed: 128 (100.0%)     Profile: Default
```

We can see that we are using the protocol TCP to send messages and connect to the server.

And the protocol UDP for the sends of files.

חלק ג

ענו על השאלות הבאות ללא קשר לחלקים הקודמים. החלק הזה עומד בפני עצמו :

1. בהינתן מחשב חדש המתחבר לרשת אנא תארו את כל ההודעות שעוברות החל מהחיבור הראשוני ל switch ועד שההודעה מתקבלת בצד השני של הצאט. אנא פרטו לפי הפורמט הבא:
   a. סוג הודעה, פירוט הודעה והשדות הבאים
      i. כתובת IP מקור/יעד, כתובת פורט מקור/יעד, כתובת MAC מקור/יעד, פרוטוקול שכבת התעבורה.

The host joins the network :

1. Host DHCP discover message.
Src : 10.0.2.15, port : 50, MAC : 00:37:6C:E2:EB:62
Dest : 192.168.1.1, port : 55, MAC : 19:55:Y7:R5:FR:34
Protocol : UDP

2. The DHCP server responds with a DHCP message.
Src: 192.168.1.1, port : 55, MAC : 19:55:Y7:R5:FR:34
Dest: 10.0.2.15, port : 50, MAC : 00:37:6C:E2:EB:62
Protocol: UDP

3. The host request IP address : DHCP request message.
Src : 10.0.2.15, port : 50, MAC : 00:37:6C:E2:EB:62
Dest : 192.168.1.1, port : 55, MAC : 19:55:Y7:R5:FR:34
Protocol : UDP

4. The DHCP sends IP address : DHCP Ack message.
Src: 192.168.1.1, port : 55, MAC : 19:55:Y7:R5:FR:34
Dest: 10.0.2.15, port : 50, MAC : 00:37:6C:E2:EB:62
Protocol: UDP

The new Client joins the server :

1. He sends a connect request to the server.
Src : 127.0.0.1, 55015, MAC : 00:34:8U:Y6:CF:44
Dest : 127.0.0.1, 55000, MAC : 00:34:8U:Y6:CF:44
Protocol : TCP

2. The server responds with an accept message.
Src : 127.0.0.1, 55000, MAC : 00:34:8U:Y6:CF:44
Dest : 127.0.0.1, 55015, MAC : 00:34:8U:Y6:CF:44
Protocol : TCP

3. The client sends a request 'login' to the server with a TCP Socket.
Src : 127.0.0.1, 55015, MAC : 00:34:8U:Y6:CF:44
Dest : 127.0.0.1, 55000, MAC : 00:34:8U:Y6:CF:44
Protocol : TCP

4. The server sends a response 'login' to the client.
Src : 127.0.0.1, 55000, MAC : 00:34:8U:Y6:CF:44
Dest : 127.0.0.1, 55015, MAC : 00:34:8U:Y6:CF:44
Protocol : TCP

The server informs all the other clients in the network that there is a new client :

1. Server sends a message to all the clients on the network that there is a new client.
Src : 127.0.0.1, 55000, MAC : 00:34:8U:Y6:CF:44
Dest : to all the clients on the server.
Protocol : TCP.

---

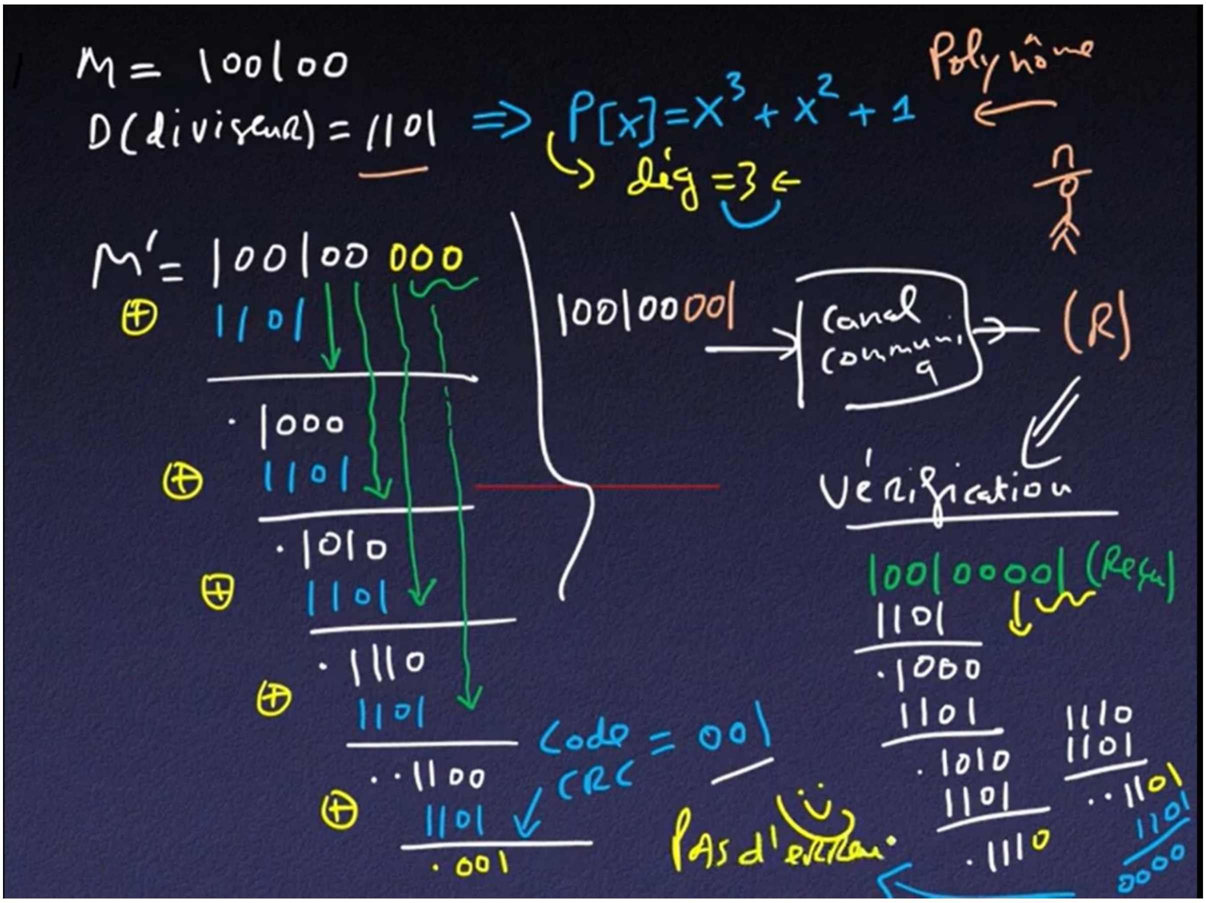
2. הסבירו מה זה CRC

From internet :

CRC (cyclic redundancy check) is an error detecting code used to determine if a certain packet/block of data has been corrupted. Its commonly used in network protocols such as TCP/IP for data verification reasons and more. The CRC algorithm computes a checksum for each specific data set that is sent through it. This algorithm utilizes a polynomial key and the transferred data.

In summary, The Cyclic Redundancy Check is a number mathematically calculated for a packet by its source computer, and then recalculated by the destination computer. If the original and recalculated versions at the destination computer differ, the packet is corrupt and needs to be resent or ignored.

Exemple to calculate the CRC :



---

3. מה ההבדל בין http 1.0 ,http 1.1, http 2.0, QUIC

<u>From internet :</u>

<u>http 1.0 :</u>

-Browser-friendly protocol
-Provided header fields including rich metadata about both request and response (HTTP version number, status code, content type)
-Response: not limited to hypertext (Content-Type header provided ability to transmit files other than plain HTML files — e.g. scripts, stylesheets, media)
-Methods supported: GET , HEAD , POST
-Connection nature: terminated immediately after the response

Etablishing a new connection for each request – major problem :

HTTP/1.0 required to open up a new connection for each request (and close it immediately after the response was sent). Each time a new connection establishes, a TCP three-way handshake should also occur. For better performance, it was crucial to reduce these round-trips between client and server. HTTP/1.1 solved this with persistent connections.

http 1.1 :

-This is the HTTP version currently in common use.
-Introduced critical performance optimizations and feature enhancements — persistent and pipelined connections, chunked transfers,compression/decompression, content negotiations, virtual hosting (a server with a single IP Address hosting multiple domains), faster response and great bandwidth savings by adding cache support.
-Methods supported: GET , HEAD , POST , PUT , DELETE , TRACE , OPTIONS
-Connection nature: long-lived

Keep-Alive header :

-The Keep-Alive header was used prior to HTTP/1.1 and was obsoleted by HTTP/1.1 making persistent connections the default behavior. Keep-Alive header can be used to define policies for long-lived communication between hosts (i.e. allows a connection to stay active until an event occurs). This laid foundation for persistence, reusable connections, pipelining, and many more enhanced capabilities in modern web communication protocols.
-Client, server, or any intermediary can provide information for Keep-Alive header independently. Also, a host can add timeout and max parameters in order to set a timeout or limit maximum request count per connection.

-HTTP pipelining, multiple connections, and many more improvements have been implemented, thanks to the Keep-Alive header’s behavior.

Upgrade header :

-With Upgrade header introduced in HTTP/1.1, it is possible to start a connection using a commonly-used protocol, such as HTTP/1.1, then request that the connection switch to an enhanced protocol type like HTTP/2.0 or WebSockets.
-In an upgraded protocol connection, max parameter (maximum request count) is not present. The upgraded protocol can provide new policies for timeout parameter (if not specifically defined, it uses default timeout value in underlying protocol).

http 2.0 :

More visual media was displayed and the volume and size of scripts adding interactivity also increased. Much more data was transmitted over significantly more HTTP requests and this created more complexity and overhead for HTTP/1.1 connections.

The HTTP/2 protocol differs from HTTP/1.1 in a few ways:

-It's a binary protocol rather than a text protocol. It can't be read and created manually. Despite this hurdle, it allows for the implementation of improved optimization techniques.

-It's a multiplexed protocol. Parallel requests can be made over the same connection, removing the constraints of the HTTP/1.x protocol.

-It compresses headers. As these are often similar among a set of requests, this removes the duplication and overhead of data transmitted.

-It allows a server to populate data in a client cache through a mechanism called the server push.

High-traffic websites showed the most rapid adoption in an effort to save on data transfer overhead and subsequent budgets.

This rapid adoption was likely because HTTP/2 didn't require changes to websites and applications. To use it, only an up-to-date server that communicated with a recent browser was necessary. Only a limited set of groups was needed to trigger adoption, and as legacy browser and server versions were renewed, usage was naturally increased, without significant work for web developers.

HTTP hasn't stopped evolving since the release of HTTP/2. Like with HTTP/1.x, HTTP's extensibility is still being used to add new features. Notably, we can cite new extensions of the HTTP protocol that appeared in 2016:

-Support for Alt-Svc allowed the dissociation of the identification and the location of a given resource. This meant a smarter CDN caching mechanism.

-The introduction of Client-Hints allowed the browser or client to proactively communicate information about its requirements and hardware constraints to the server.

-The introduction of security-related prefixes in the Cookie header helped guarantee that secure cookies couldn't be altered.

QUIC :

QUIC is a new application layer protocol proposed by Google to be used for transferring web pages. QUIC is a novel approach and is starting from a fresh slate unlike HTTP.

The biggest difference between QUIC and HTTP is QUIC runs on top of UDP whereas HTTP runs on top of TCP. QUIC moves the TCP reliability, congestion control, and in order delivery to the application layer thus giving it more control and allowing for changes to be applied easier. For example, one of the features of QUIC is its congestion control algorithm is pluggable thus you can choose whichever congestion control algorithm you like without having to change the underlying network stack. QUIC offers the same multiplexing capabilities as HTTP/2 without the issue of head of line blocking. QUIC creates multiple independent streams over its single UDP connection and a loss on one of the streams doesn't cause the other streams to become blocked. QUIC also reduces the connection setup time by introducing a 1 RTT handshake unlike TCP which relies on a 3 RTT handshake.

---

4. למה צריך מספרי port?

Ports identify what process on the host the received traffic should be sent to. A computer can have multiple simultaneous connections, all receiving data for different processes (mail, web, database …) on that computer. A port is always associated with a protocol.

Some of these port numbers are specifically defined and always associated with a specific type of service -- for example, File Transfer Protocol (FTP) is always port number 21 and Hypertext Transfer Protocol web traffic is always port 80. These are called *well-known ports* and go from 0 to 1023.

In other words, the IP address identifies the computer host, and the port number specifies what specific process is running on that host.

---

5. מה זה subnet ולמה צריך את זה?

A subnet is a network inside a network. Subnets make networks more efficient. Through subnetting, network traffic can travel a shorter distance without passing through unnecessary routers to reach its destination.

In a network, there could be millions of connected devices, and it could take some time for the data to find the right device. This is where subnetting comes in handy. subnetting narrows down the IP address to usage within a range of devices and shorten the path for a specific packet because of it.

---

6. למה צריך כתובות mac למה לא מספיק לעבוד עם כתובות ip?

A media access control address (MAC address) is a unique identifier assigned to a network interface controller (NIC) for use as a network address in communications within a network segment. This use is common in most IEEE 802 networking technologies, including Ethernet, Wi-Fi, and Bluetooth.

An IP address is a unique address that identifies a device on the internet or a local network. IP stands for "Internet Protocol," which is the set of rules governing the format of data sent via the internet or local network.

In essence, IP addresses are the identifier that allows information to be sent between devices on a network: they contain location information and make devices accessible for communication. The internet needs a way to differentiate between different computers, routers, and websites. IP addresses provide a way of doing so and form an essential part of how the internet works.

Both MAC Address and IP Address are used to uniquely define a device on the internet. The main difference between MAC and IP address is that MAC Address is used to ensure the physical address of the computer. It uniquely identifies the devices on a network. While IP addresses are used to uniquely identifies the connection of the network with that device takes part in a network.

Even if your computer has an IP address, it still needs a MAC address to find other machines on the same network (especially the router/gateway to the rest of the network/internet), since every layer is using underlying layers.

7. מה ההבדל בין Router Switch Nat?

First we will see the difference between a router and a switch :
The switch and router are the two essential components of a network. Although both are the connecting devices in a network, and sometimes people assume them as the same, both have different functionality.
A switch allows you to have all your peripherals connected to the same place and distributes the flow on the different ports. It is rather recommended for professional use.
A router offers additional functions (security, WiFi connection, etc.). It is rather recommended within a home, especially for gaming enthusiasts.

A router is a device that allows communication between your computer and the Internet, while a switch (or switch) allows you to connect several devices within the same Ethernet network.
Router store IP address in the routing table whereas Switch store MAC address in a lookup table. Routers supports NAT while Switches does not.

And now we will see the difference between NAT and both of them :
What is a NAT ? Network address translation (NAT) stands for network address translation. It’s a way to map multiple local private addresses to a public one before transferring the information. Organizations that want multiple devices to employ a single IP address use NAT, as do most home routers.
Also, it does the translation of port numbers in the packet that will be routed to the destination. It then makes the corresponding entries of IP address and port number in the NAT table. NAT generally operates on a router.
The global difference is that NAT is an algorithm and Routers and Switches are computers that allow communication between networks and devices.
NAT is a platform that builds on a Router since its works on the Network layer and Switches works in the Link layer.

---

8. שיטות להתגבר על המחסור בIPv4 ולפרט?

There are methods to overcome the lack of IPv4 addresses such as :

1. Retrieval of address blocks

use of formerly reserved address blocks (like 14.0.0.0/8). Stanford University thus returned block 36.0.0.0/8 to the IANA in 2000 and Interop did the same for block 45.0.0.0/8 in 2010.

2. IPv6

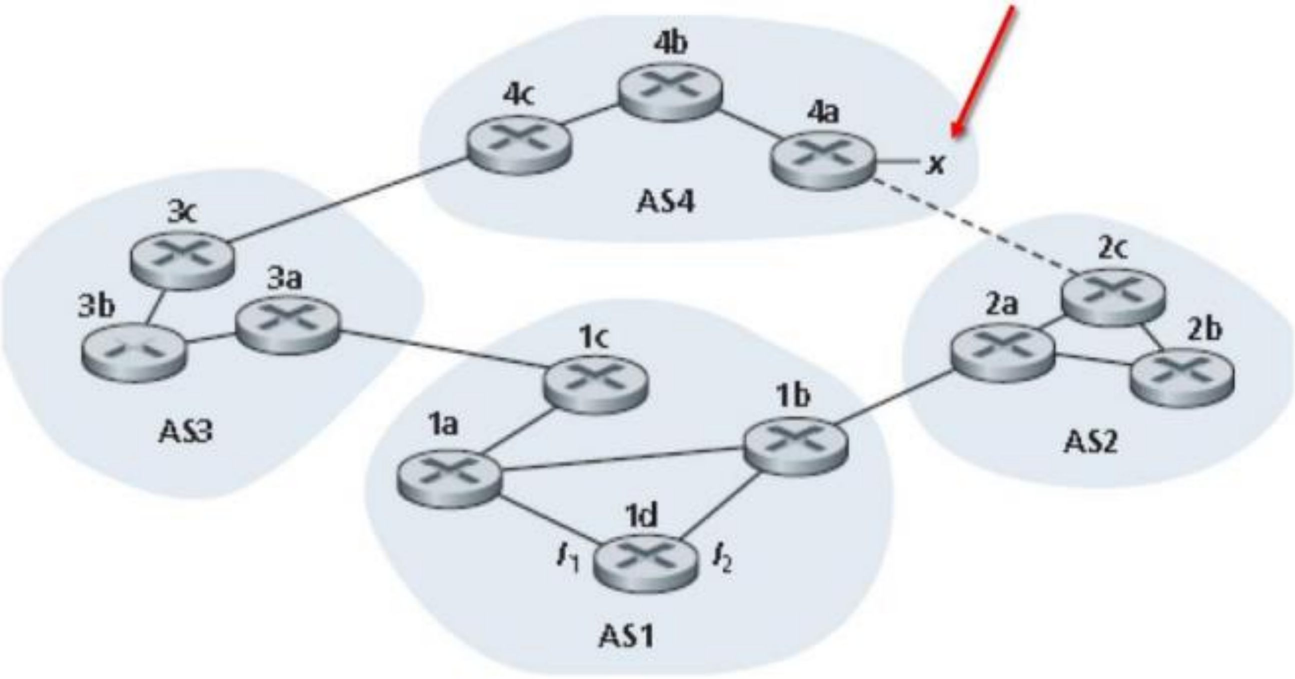
Accelerating the transition to IPv6: the only solution to preserve the internet. Accelerating the transition from IPv4 to IPv6 is the only sustainable solution, because only an almost total transition to IPv6 can allow content providers to do without IPv4.

3. NAT

To ensure the continuity of Internet access despite the exhaustion of IPv4 addresses, operators are considering the deployment of large-scale address translators (Large Scale NAT, Carrier-Grade NAT). A public address would thus be shared by many clients simultaneously. These methodes saves a lot of IPv4 and reduce the need.

9. נתונה הרשת הבאה.

a. AS2, AS3 מריצים OSPF
b. AS1, AS4 מריצים RIP
c. בין ה-Ass רק BGP
d. אין חיבור פיזי בין AS2, AS4
e. בעזרת איזה פרוטוקול לומד הנתב 3c על תת רשת x
f. בעזרת איזה פרוטוקול לומד הנתב 3a על תת רשת x
g. בעזרת איזה פרוטוקול לומד הנתב 1c על תת רשת x
h. בעזרת איזה פרוטוקול לומד הנתב 2c על תת רשת x

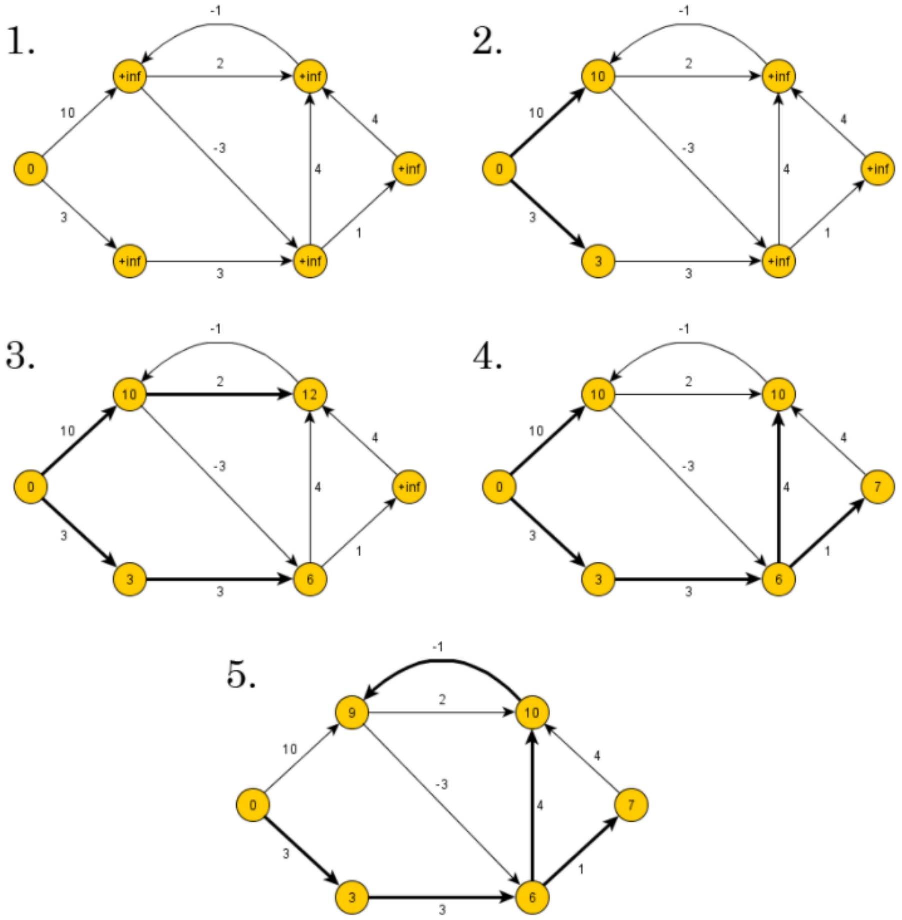


OSPF(Open Shortest Path First) : While routers connect networks using the IP protocol, OSPF technology is a protocol used to determine the best path that packets can take to travel through a series of connected networks.

RIP (Routing Information Protocol): is a distance-vector routing protocol. Routers running the distance-vector protocol send all or a portion of their routing tables in routing-update messages to their neighbors. You can use RIP to configure the hosts as part of a RIP network.

RIP use the algorithm of Bellman-Ford. It allows each router to communicate to neighboring routers.

Exemple of Bellman-Ford algorithm :



<u>BGP (Border Gateway Protoco) :</u> is an external route exchange protocol (an EGP), used in particular on the Internet network. Its main purpose is to exchange network routing and reachability information (called prefixes) between Autonomous Systems (AS). Since it travels over TCP, it is considered to belong to the application layer of the OSI model.

<u>Answers :</u>

e. eBGP

f. iBGP

g. eBGP

h. iBGP