



الجامعة اللبنانية الأمريكية
Lebanese American University

School of Arts and Science

Department of Computer Science and Mathematics

HZA Messenger



A Chatting Application

By:

Ahmad Al Maaz ID:202204713

Hadi Al Hassan ID: 202202352

Zeina Mershad ID:202209100

Contents

1	Introduction	1
2	Phase I: Environment Setup	1
2.1	Virtual Machine Installation	1
2.2	Linux Netem install	1
3	Phase II: Sending Messages	2
3.1	Basic Connectivity Between Two Peers	2
3.1.1	High Level Concept	2
3.1.2	Implementation	3
3.2	Both Peers sending and Receiving Messages	4
3.3	Reliability of the messages	4
3.3.1	Utilization of stop and wait	4
3.3.2	3 Way Handshake	5
3.3.3	Packet Size	5
3.3.4	AsciiSum	6
3.3.5	Timeout	7
3.3.6	Packet Structure	9
4	Phase III: File Transfer	10
4.1	Testing over Netem	11
5	Phase IV: GUI	12
6	Testing over Unreliable conditions	14
6.1	Delay	14
6.2	Packet Loss	15
6.3	Corrupted Packets	15
6.4	Out of Order packets	16
7	Work Distribution	17
8	Sources We Used	17

1 Introduction

This report introduces HZA MESSENGER, a client server chatting application that sends messages over UDP, and files using TCP.

In this document, we discuss the high level design of our app, the implementation decisions we made, and showcase our code, in boring detail. We also present our application using a GUI. Throughout the report, we mention any challenges we faced, and how we fixed them, a part Dr. Ayman likes to see.

2 Phase I: Environment Setup

2.1 Virtual Machine Installation

Virtual Box was already installed from a previous course.

We installed Ubuntu from the official website and created the virtual machine. Python 3 was already installed on the machine, to retrieve the code for testing we installed git on the machine using the `sudo apt-get install git-all` command

2.2 Linux Netem install

To install netem, we opened the terminal after granting the account root privileges and entered the following command: `sudo apt-get install iproute2`

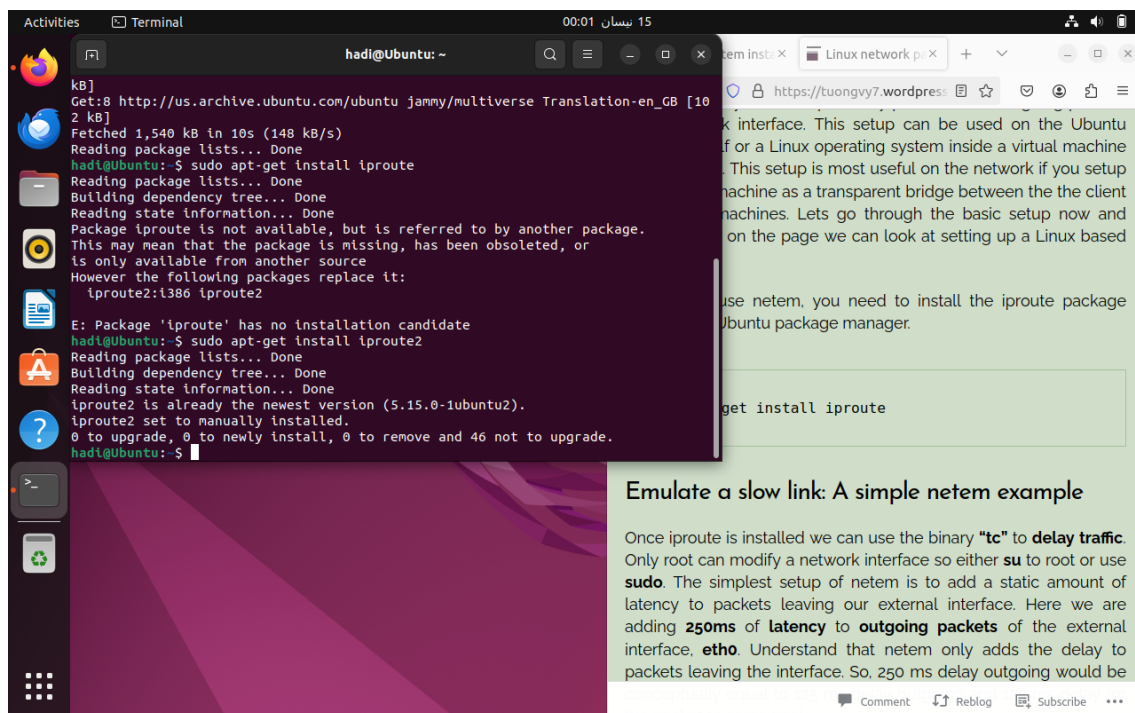


Figure 1: Netem installation

3 Phase II: Sending Messages

3.1 Basic Connectivity Between Two Peers

3.1.1 High Level Concept

We start off simple, with a straightforward UDP connection. With messages being sent from one side (client) to another side (Server). The server will be always on, waiting for a connection.

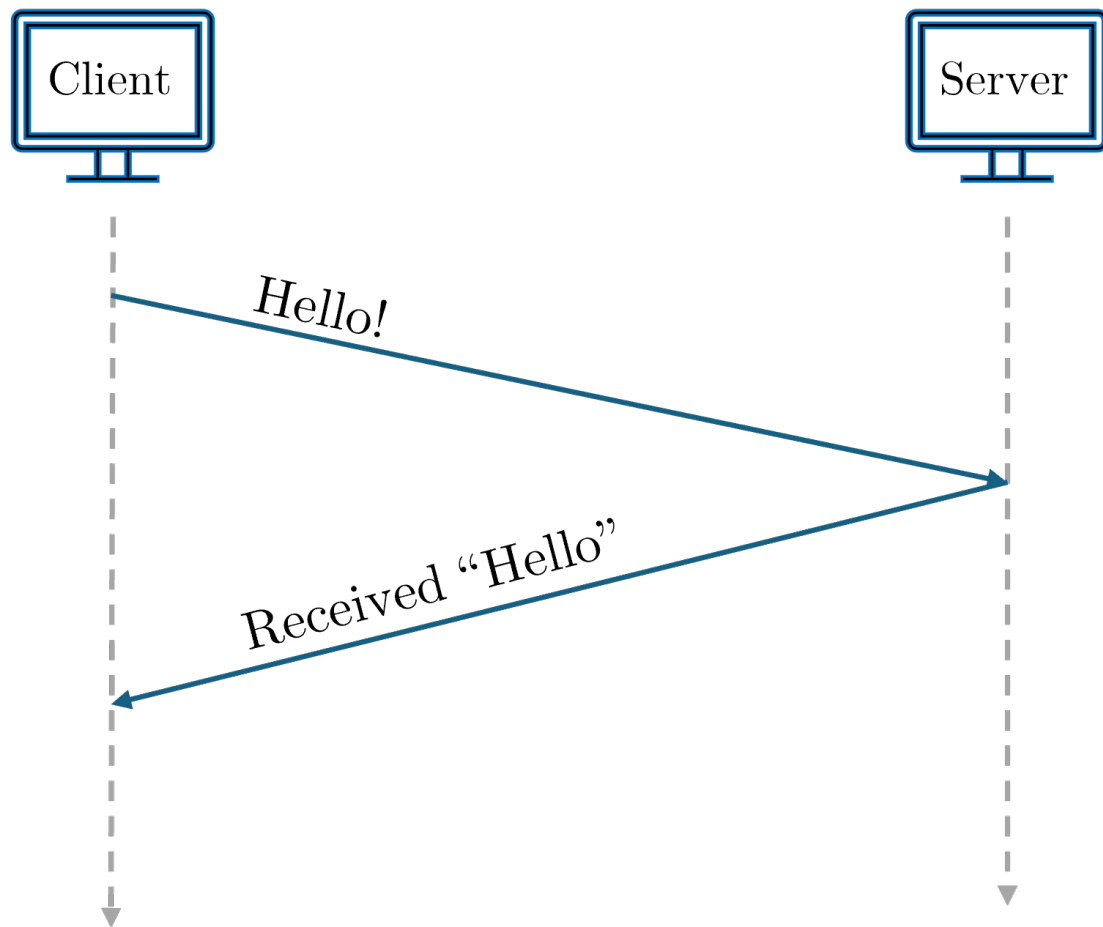


Figure 2: High Level look at Phase 2 a)

This will be the starting point of our project, where we will begin to add on features and start considering reliability as we go.

3.1.2 Implementation

The implementation was fairly easy, having studied this connection in Chapter 2.

```
1  '''
2  Alice is the Client in Phase 2. a)
3  We are Using UDP
4  '''
5  from socket import *
6
7  port = 42069
8  Server_address = "127.0.0.1"
9  client_socket = socket(AF_INET,SOCK_DGRAM)
10
11 message = input("Enter a message: ")
12
13 try:
14     client_socket.sendto(message.encode(),(Server_address,port))
15     print("Message has been sent to server!")
16
17     response,address = client_socket.recvfrom(2048)
18     print(f"Received \"{response.decode()}\" from Server!")
19
20 except Exception as e:
21     print("An error occurred:", e)
22
23 finally:
24     client_socket.close()
```

```
1  '''
2  Bob is the Server in Phase 2. a)
3  We are Using UDP
4  '''
5  from socket import *
6
7  port = 42069
8  Server_socket = socket(AF_INET,SOCK_DGRAM)
9
10 Server_socket.bind(("127.0.0.1",port))
11 try:
12     while True:
13         print("*****Server is active*****\nAwaiting message from Client")
14         message, client_address = Server_socket.recvfrom(2048)
15         print(f"Client sent \"{message.decode()}\" ")
16         print("Replying to client")
17
18         Server_socket.sendto("Well Received Friend!".encode(),client_address)
19
20 except Exception as e:
21     print("An error occurred:", e)
22
23 finally:
24     print("Connection Terminated")
25     Server_socket.close()
26
27
28
```

3.2 Both Peers sending and Receiving Messages

for this section we utilized multi-threading, where the user would send on one thread, and receive on another.

```
1 receive_thread = threading.Thread(target=listen_to_packets)
2 receive_thread.start()
3 receive_thread = threading.Thread(target=listen_to_input)
4 receive_thread.start()
```

3.3 Reliability of the messages

This section discusses the features we implemented to make our application reliable in sending its messages.

3.3.1 Utilization of stop and wait

We opted to choose *stop and wait*, where we would send the data, packet by packet. This would ensure in order delivery of the message.

Each packet would be sent with a sequence number, and we do not send another until the sender has acknowledged

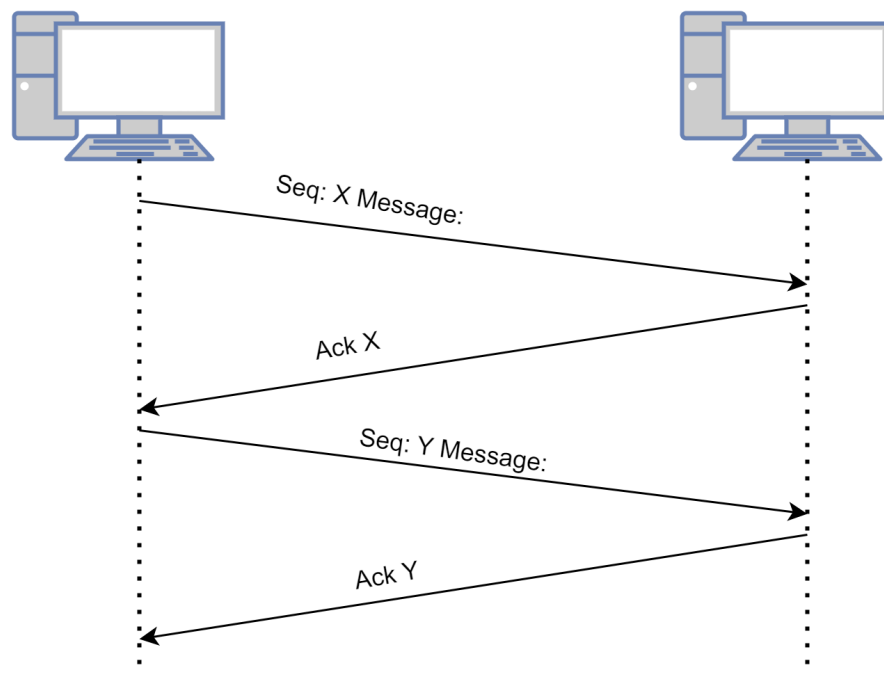


Figure 3: Stop and Wait

3.3.2 3 Way Handshake

TCP utilizes a 3-way handshake to synchronize. The client sends a **SYN** message to the server, the server **Acknowledges** the SYN with a **SYN-ACK** message, to which the client acknowledges the message with an **Ack** (as illustrated in figure 3).

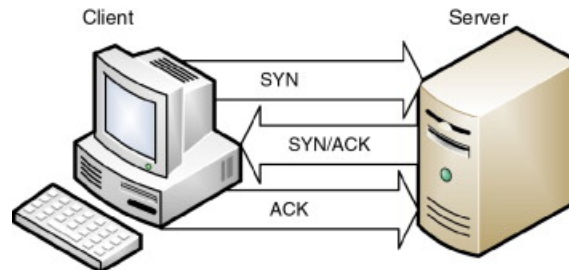


Figure 4: Taken from chapter 3: CISSP Study Guide (Second Edition)

We utilized such 3-way establishment in our application, to establish a reliable connection between the two peers:

```
1 if want_to_initiate == "y":
2     print("trying to connect")
3     send_packet_with_timeout(DataPacket(id, seq, "SYN", False, calculate_ascii_comb("SYN")))
4     connection_state = ConnectionState.SYN_SENT
5     send_packet(pickle.dumps(DataPacket(id, seq, "ACK", False, calculate_ascii_comb("ACK"),0)))
6     connection_state = ConnectionState.ESTABLISHED
7 else:
8     message, peer1_address = peer_socket.recvfrom(1000)
9     send_packet_with_timeout(DataPacket(id, seq, "SYNACK", False, calculate_ascii_comb("SYNACK"),0))
10    connection_state = ConnectionState.ESTABLISHED
```

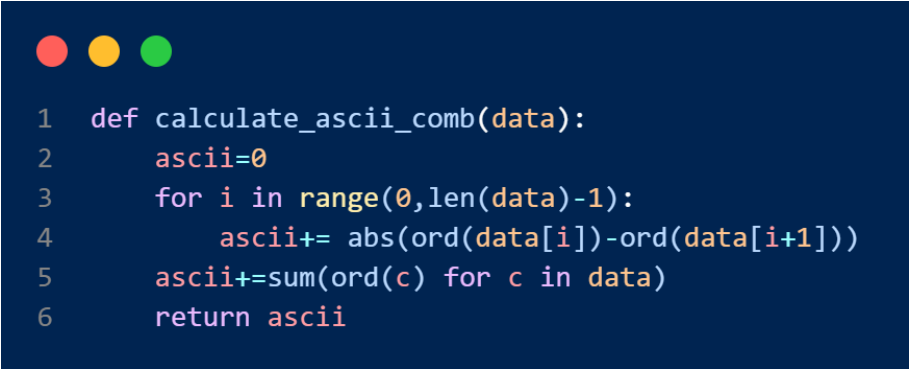
3.3.3 Packet Size

we started off with a mere 6 bytes, and after making sure our application was reliable, we upped it to 1000 bytes.

3.3.4 AsciiSum

UDP utilizes a checksum, where it takes blocks of 16 bits, adds them together, and does the 1's compliment.

Our application implements it's own checksum, dubbed **AsciiSum**



```
1 def calculate_ascii_comb(data):
2     ascii=0
3     for i in range(0,len(data)-1):
4         ascii+= abs(ord(data[i])-ord(data[i+1]))
5     ascii+=sum(ord(c) for c in data)
6     return ascii
```

Where we sum the difference of the ascii value of each 2 adjacent chars, and then add them with the sum of all the chars in the message.

- ① The sum of the difference of each two adjacent chars will ensure that flipped bits will be detected (accounting for all possible permutations of a message)
- ② The sum of the ascii of all the chars will account for bit shifts (example below to illustrate).

For example, consider the example **abc**, with ascii values of, 97, 98, 99, if we were to only sum the values of each char, it wouldn't be able to differentiate between **abc** and **cab**, **bac**.

And summing difference between each two adjacent chars wouldn't be enough, for example the sum of **abc** and **def** would be equal, and therefore pass the check.

We cannot claim full safety against all scenarios of packet corruption, but this technique is should cover most, if not all cases of flipped/switched bits.

3.3.5 Timeout

We set our timer to 0.2 seconds, in which the sender resends the packet if they don't receive an ack during the timeout period.

```
1 def send_packet_with_timeout(dataPacket):
2     while True:
3         try:
4             peer_socket.sendto(pickle.dumps(dataPacket), ('localhost', sender_port))
5             peer_socket.settimeout(.2)
6             message, peer1_address = peer_socket.recvfrom(1000)
7             packet = pickle.loads(message)
8             validate_packet(packet, seq, clientSeq=clientSeq)
9             peer_socket.settimeout(0)
10            return packet
11        except socket.timeout:
12            print("Timeout occurred. Retrying...")
13        except CorruptedPacket as e:
14            print(e)
15        except Exception as e:
16            print(e)
17        pass
```

This implementation would cover all scenarios of timeout:

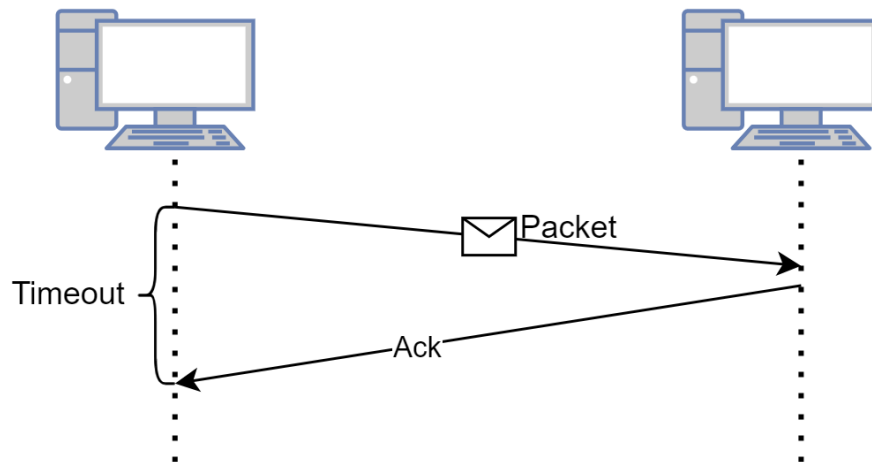


Figure 5: Timeout Scenario 1

In the case of an ack that isnt received:

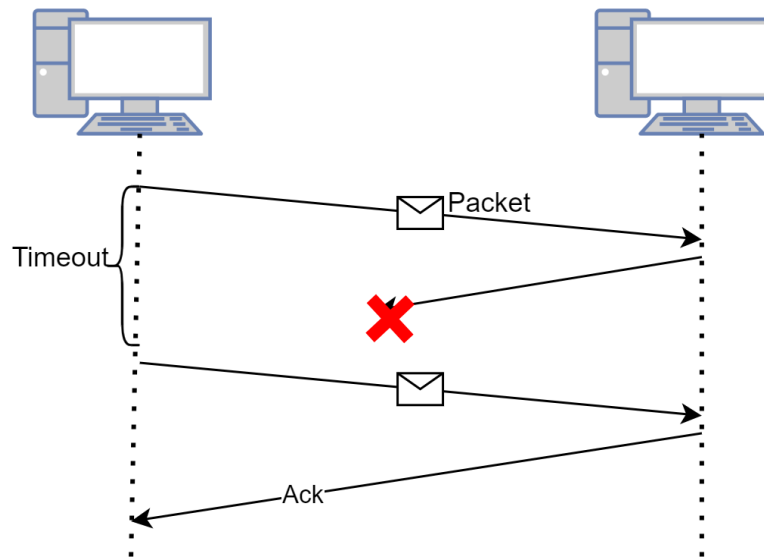


Figure 6: Timeout Scenario 2

Or a packet that did not arrive:

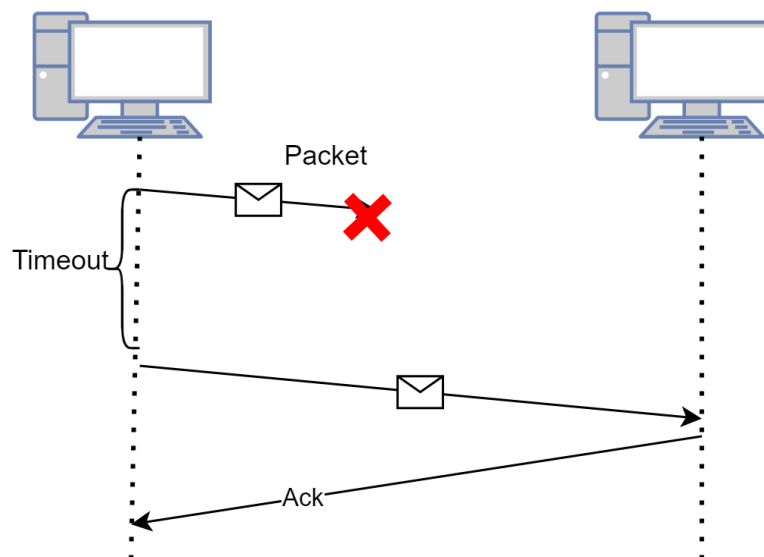
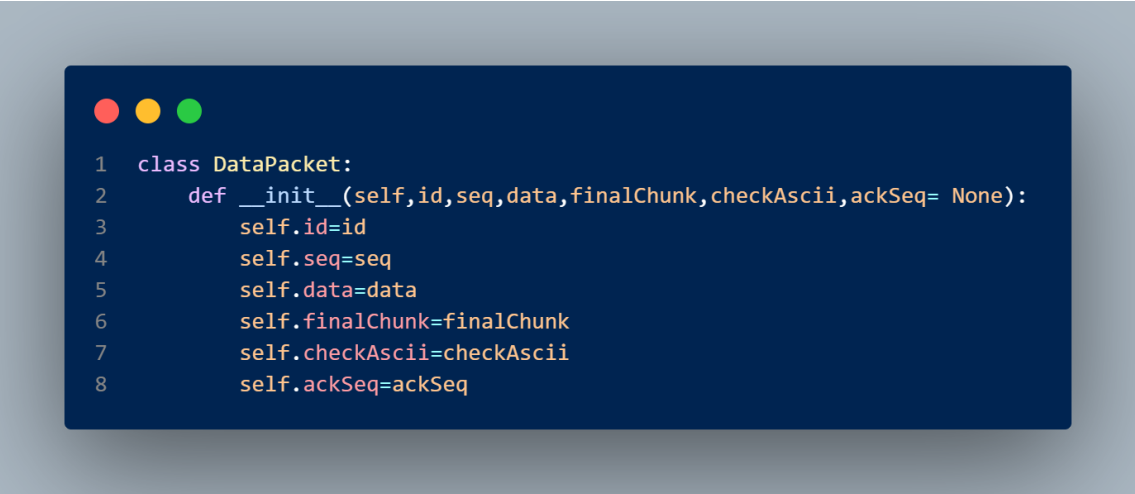


Figure 7: Timeout Scenario 3

3.3.6 Packet Structure



```
1 class DataPacket:
2     def __init__(self, id, seq, data, finalChunk, checkAscii, ackSeq= None):
3         self.id=id
4         self.seq=seq
5         self.data=data
6         self.finalChunk=finalChunk
7         self.checkAscii=checkAscii
8         self.ackSeq=ackSeq
```

Figure 8: DataPacket.py class

Our packet class consists of the following:

- *id*: used for packet reordering.
- *seq*: current sequence number of the packet.
- *data*: contents being sent.
- *finalChunk*: boolean, indicates if it is the last chunk being sent.
- *checkAscii*: our custom checksum.
- *ackSeq*: interchanges between 0 and 1, between sender and receiver during handshaking, for regular sent packets we keep them as the default **none** value.

4 Phase III: File Transfer

File transfer in this application is implemented over TCP, and was tested over `netem`.

File Transfer Restriction:

- To prevent file duplication, the sender of a file cannot re-download it, since the sender already has a copy of that file.
- If two files share the same name but contain different content, the most recent one will be kept. File Transfer Interface: There's a section in the graphical user interface that contains a dictionary with file names and their corresponding base64 encoded content. This allows for conversion between binary and base64 formats. When a file is received, it is initially in binary format, but it's converted to base64 and stored in a dictionary for easy display.
- each file ,once sent, will exhibit an increase up to 30

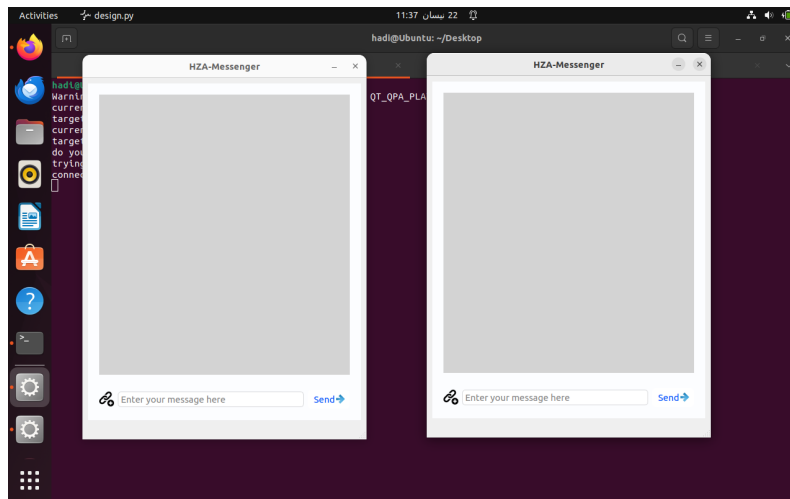
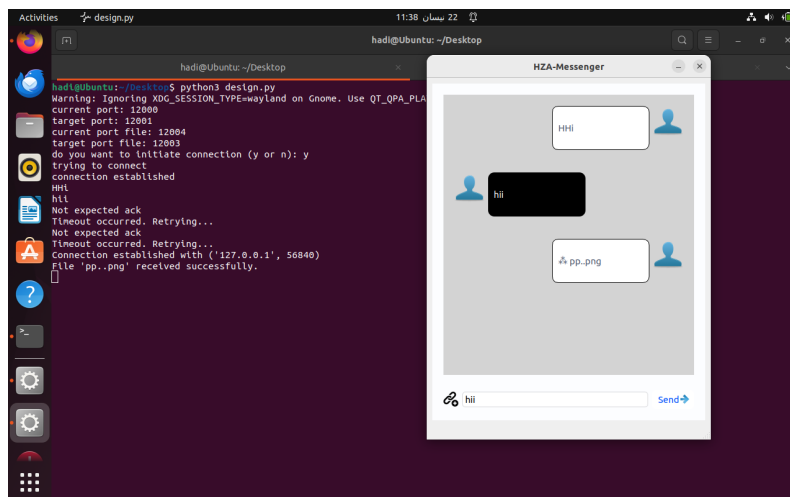


Figure 9: GUI with File Transfer



4.1 Testing over Netem

```
sudo tc qdisc add dev lo root loss 50 %
```

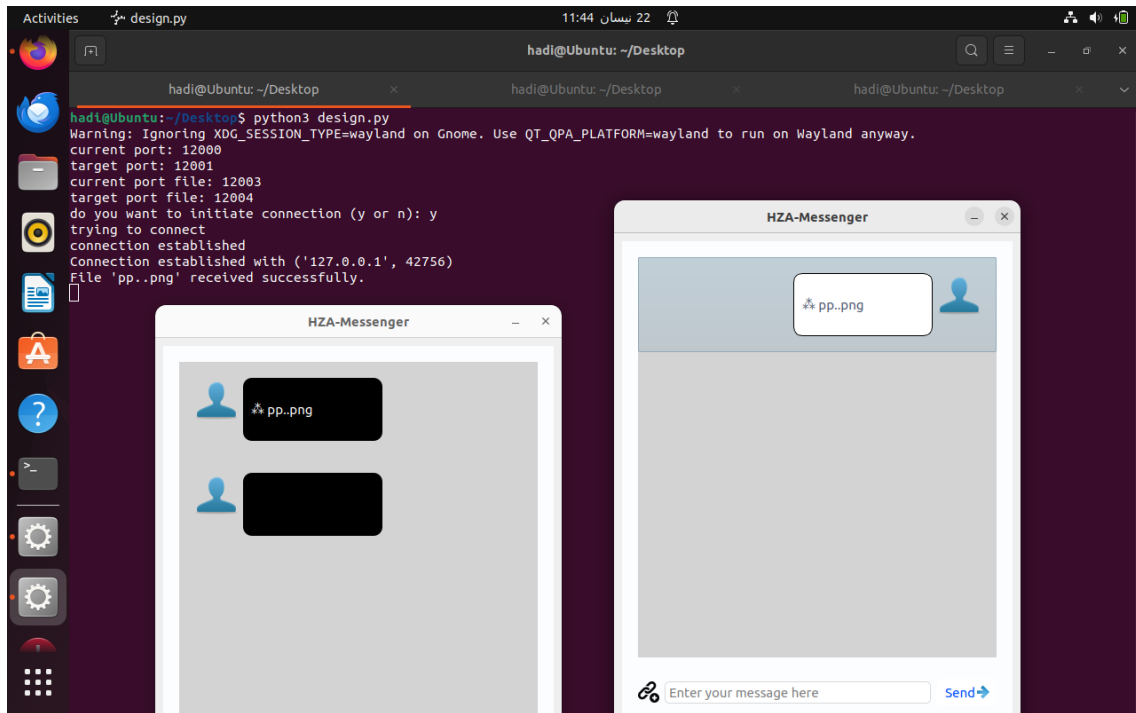


Figure 10: Successful file transfer (50% packet loss)

5 Phase IV: GUI

Our GUI of choice is `PyQt5`, as it is versatile, and allows for easy design using a graphical interface to drag and drop the elements, in addition to its `PyUic5` tool, that allows for conversion from a `.ui` file to a `.py`.

Ironically, the troubleshooting took us more time than learning how to use the actual software.

to install the software, we ran the following command: `pip install PyQt5`.

Afterwards we faced some issues, for starters the `QTdesigner` was not installed when we ran the command (although it was supposed to be). We fixed the issue by installing qt designer standalone from this website: <https://build-system.fman.io/qt-designer-download>.

We also had trouble running the `PyUic5` command, we spent the longest time trouble-shooting this, eventually 2 things helped re-mediate this issue: first, we added the path of the location of our python scripts into our `PATH` environment variables (a tip we got from an Indian off YouTube), second was from a comment on a support thread, that recommended us to uninstall the packages and reinstall them, which fixed our issue, ironically, unplugging and plugging back in (metaphorically) worked.

To run the app, run the `design.py` file, and enter your port of your choosing, and the port of your contact, the other user will do too. Users must then indicate whether they want to initiate the connection or not, one must type `n`, the other `y`. The “Contactee” must press `n`, before the other presses `y`.

Code Testing

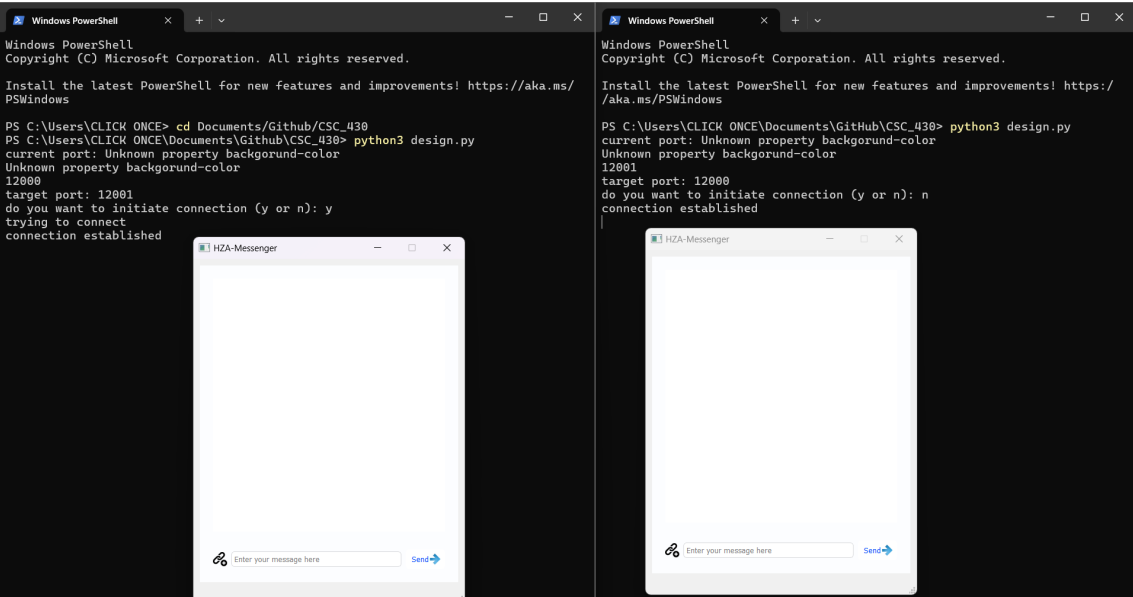


Figure 11: Initialization

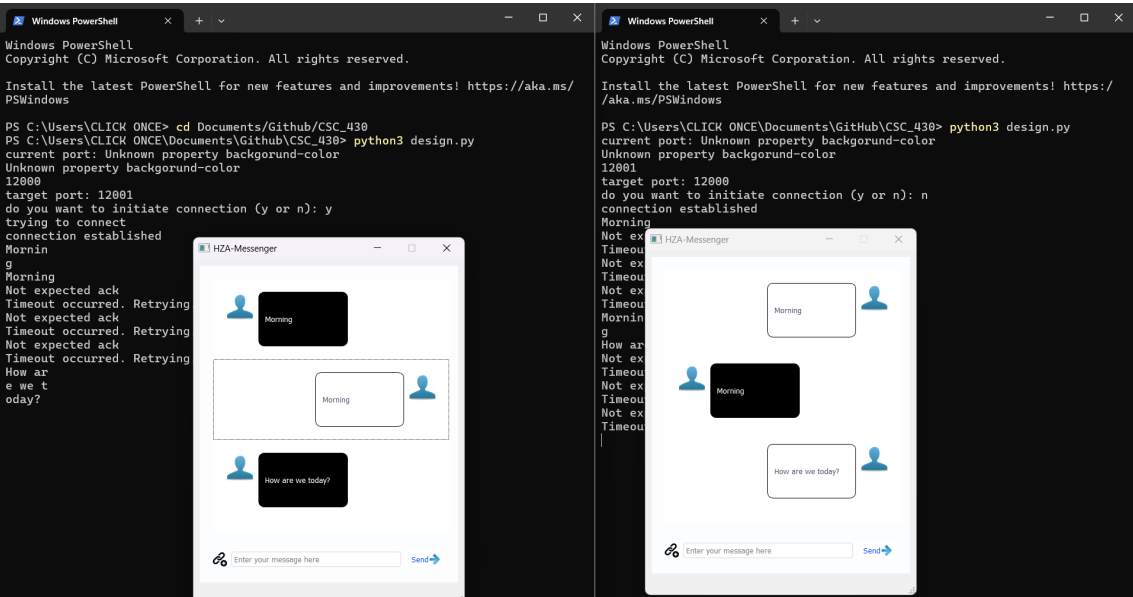


Figure 12: Testing in Action

6 Testing over Unreliable conditions

We used Linux `netem` to simulate delay, packet loss, corrupted packets and out-of-order packets.

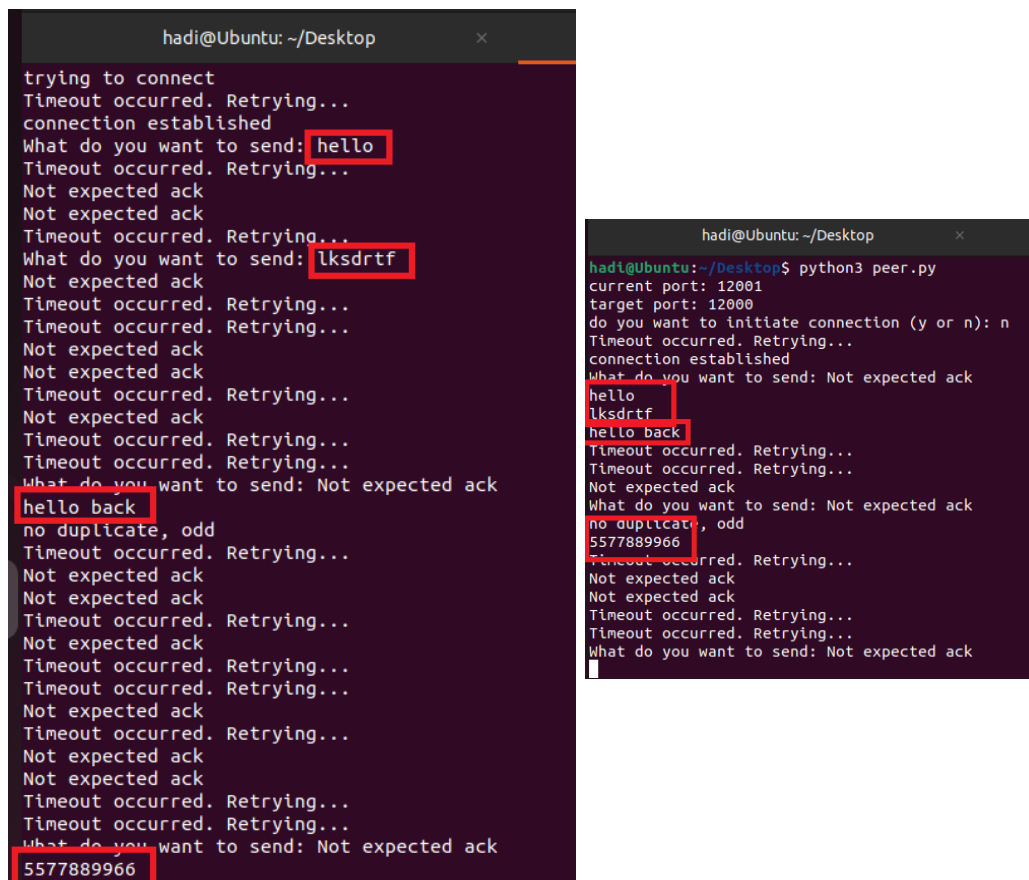
The commands are very straightforward, we took the format from a lab sheet from the university of new Mexico. It is worth noting that the commands did not work immediately, we had trouble shoot online, and found solutions through support forum threads, mentioned in the sources section below.

for example, we had to place our host's own name in the command, instead of the `eth2` name in the report, we found the host name by running the `ip addr` command, and got the name for the local host.

To undo any changes we would use the command `sudo tc qdisc dev del lo root`

6.1 Delay

we simulated an average delay of 100ms, give or take 10ms each time. Using the command `sudo tc qdisc add dev lo root netem delay 100ms 10ms 25%`, and our application was able to send the messages.



```
hadi@Ubuntu: ~/Desktop
trying to connect
Timeout occurred. Retrying...
connection established
What do you want to send: hello
Timeout occurred. Retrying...
Not expected ack
Not expected ack
Timeout occurred. Retrying...
What do you want to send: lksdrtf
Not expected ack
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Not expected ack
Not expected ack
Timeout occurred. Retrying...
Not expected ack
Timeout occurred. Retrying...
Timeout occurred. Retrying...
What do you want to send: Not expected ack
hello back
no duplicate, odd
Timeout occurred. Retrying...
Not expected ack
Not expected ack
Timeout occurred. Retrying...
Not expected ack
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Not expected ack
Timeout occurred. Retrying...
Not expected ack
Timeout occurred. Retrying...
Timeout occurred. Retrying...
What do you want to send: Not expected ack
5577889966

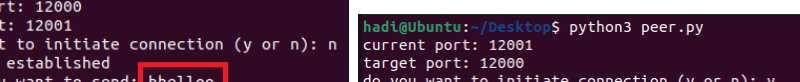
hadi@Ubuntu: ~/Desktop
hadi@Ubuntu:~/Desktop$ python3 peer.py
current port: 12001
target port: 12000
do you want to initiate connection (y or n): n
Timeout occurred. Retrying...
connection established
What do you want to send: Not expected ack
hello
lksdrtf
hello back
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Not expected ack
What do you want to send: Not expected ack
no duplicate, odd
5577889966
Timeout occurred. Retrying...
Not expected ack
Not expected ack
Timeout occurred. Retrying...
Timeout occurred. Retrying...
What do you want to send: Not expected ack
```

Figure 13: Delay Simulation (100ms)

We simulated packet loss through the command `sudo tc qdisc add dev lo netem loss 50%`, in which we dropped 50% of the packets being sent.

[illegible]

```
command: sudo tc qdisc change dev lo root netem corrupt 50%
```



```
hadi@Ubuntu:~/Desktop$ python3 peer.py
current port: 12000
target port: 12001
do you want to initiate connection (y or n): n
connection established
what do you want to send: hhelloo
recieveed
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
What do you want to send: █

hadi@Ubuntu:~/Desktop$ python3 peer.py
current port: 12001
target port: 12000
do you want to initiate connection (y or n): y
trying to connect
connection established
What do you want to send:
What do you want to send: What do you want to send: hhelloo
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
What do you want to send: recieveed
█
```

6.4 Out of Order packets

This test is pretty useless since we are doing stop and wait, but nonetheless we simulated.

```
hadi@Ubuntu:~$ cd Desktop
hadi@Ubuntu:~/Desktop$ python3 peer.py
current port: 12000
target port: 12001
do you want to initiate connection (y or n): y
trying to connect
connection established
What do you want to send: hi there
kifak jawad
Timeout occurred. Retrying...
What do you want to send: m2*** me*** eSt l dinne sarle 3 shhor ma 3am tale3 wala li8a, kel hada bihebne fal, kel shabe tarakoune,
w eli 28 sinne...
testinng simukateesdsdsdddsd

Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
What do you want to send:

hadi@Ubuntu:~/Desktop$ python3 peer.py
current port: 12001
target port: 12000
do you want to initiate connection (y or n): n
connection established
What do you want to send: hi there
Timeout occurred. Retrying...
What do you want to send: kifak jawad
m2*** me*** eSt l dinne sarle 3 shhor ma 3am tale3 wala li8a, kel hada bihebne fal, kel shabe tarakoune, w eli 28 sinne...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
Timeout occurred. Retrying...
What do you want to send: sdsdsdddsd
Timeout occurred. Retrying...
What do you want to send: testinng simukateee
```

The testing proves that stop and waiting is the perfect, albeit slow, method for sending packets reliably.

7 Work Distribution

The instructors favorite section of the report,

1. The application's high level design and features, were a collaborative effort between Zeina, Ahmad and Hadi.
2. The implementation of application, was done by Ahmad and Zeina. The choice of GUI was suggested by Ahmad, and the final version was done by Ahmad and Zeina.
3. Testing the code and application was performed using `netem`, was performed by Hadi.
4. The report was typesetted in \LaTeX by Hadi, reviewed by Zeina and Ahmad.

8 Sources We Used

- *The University of New Mexico's* Traffic Control Manual For Lab1 provided us with the insight to use the `netem` commands. <https://www.cs.unm.edu/~crandall/netsfall13/TCtutorial.pdf>
- A support thread on *Arch Linux's* website, helped us resolve an issue with `netem` commands, specifically the comment added by user `loqs`, who highlighted that the other commands should be used with the keyword `add`, and not `change`. <https://bbs.archlinux.org/viewtopic.php?id=238231>
- *OpenAI's* generative ai `chatgpt`, was a good resource for troubleshooting and figuring out errors in `netem` and our GUI.
- *Tech with Tim's* youtube tutorials, that taught us `PyQt5` from scratch in such short time. <https://youtu.be/Vde5SH8e10Q?si=fVeAWyno-VFJ-PI5>
- User *Ahmed Zeid's* comment on the QT Forum Support chat, that helped us fix an error with `PyUic5` <https://forum.qt.io/topic/123172/troubles-using-pyuic5/9>
- *Moustafa Oumar's* reply on a *Stack Overflow* thread, that helped us with errors with `PyUic5`. <https://stackoverflow.com/questions/47153250/pyuic5-is-not-recognized-as-an-internal-or-external-command>
- *Linux manual page's* `tc-netem(8)` guide. <https://man7.org/linux/man-pages/man8/tc-netem.8.html>