LAU
الجامعة اللبنانية الأميركية
Lebanese American University

CSC430 - Computer Networks
Phase 2: Preparation Phase

Roula Ghaleb
Abdallah Tourbah

For: Dr. Ayman Tajjedine
Deadline: April 23, 2023

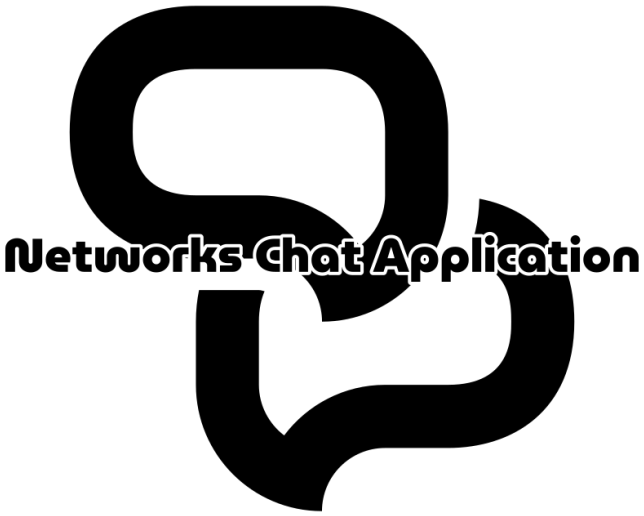


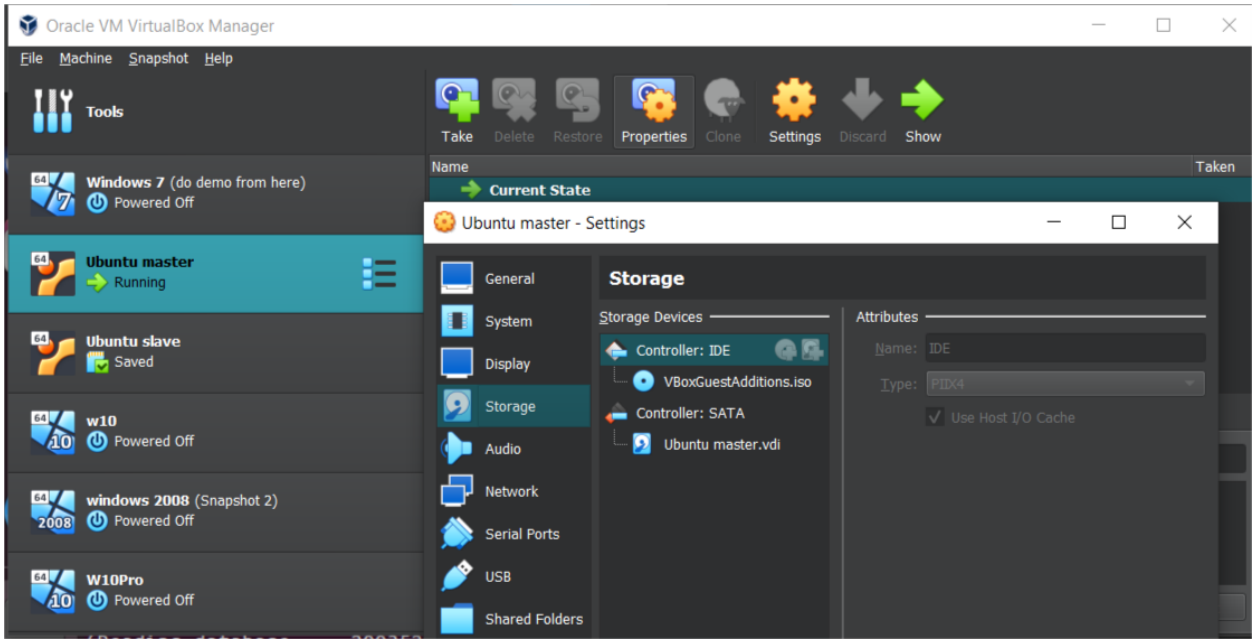
Table of Contents

1. Thought Process **3**
2. Server and Client **3**
3. Bidirectional Communication **3**
4. Message Reliability **3**
5. Unreliable Network Conditions **3**

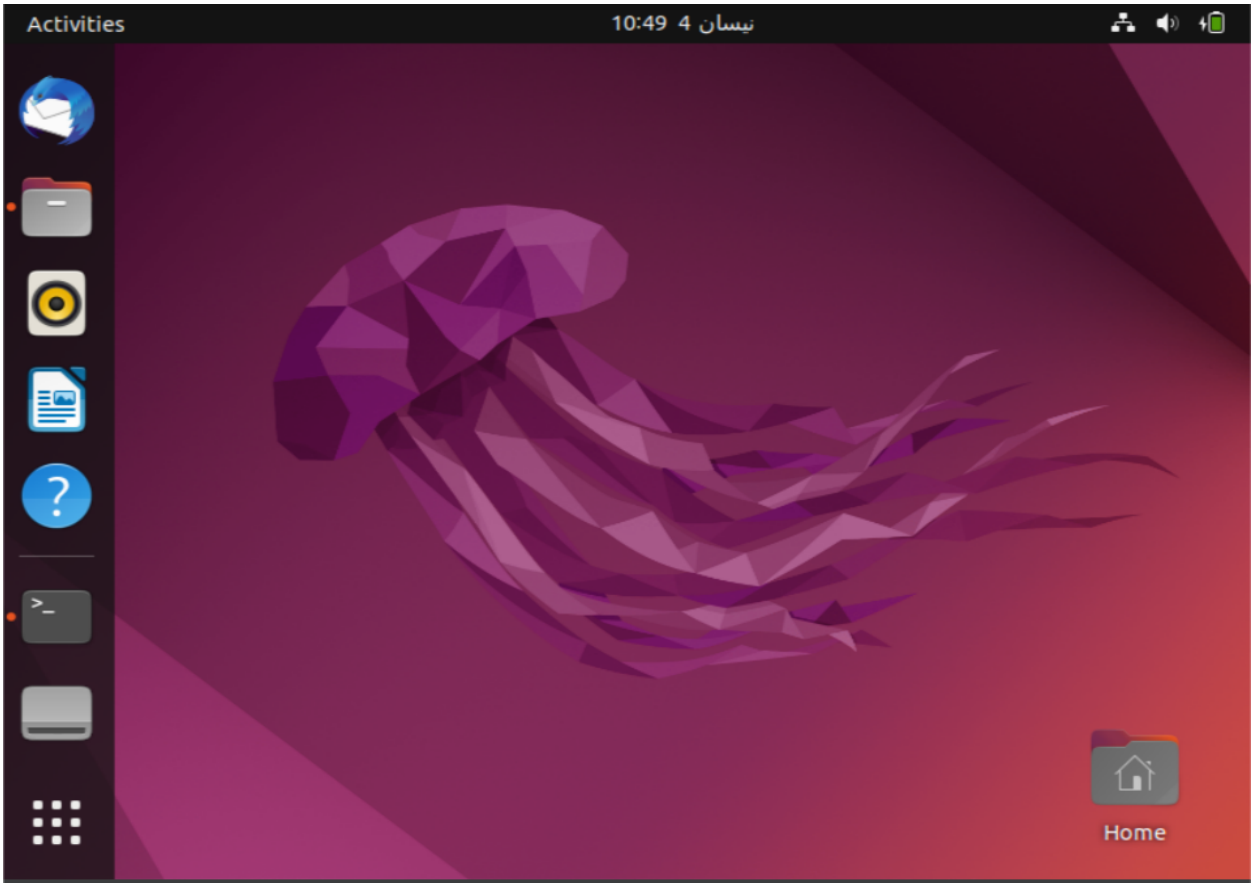
1. Phase 1

Installing and Running Ubuntu

The first step we did was install an Ubuntu 22.04 release .vdi file from google and setting it up on our virtual machine:

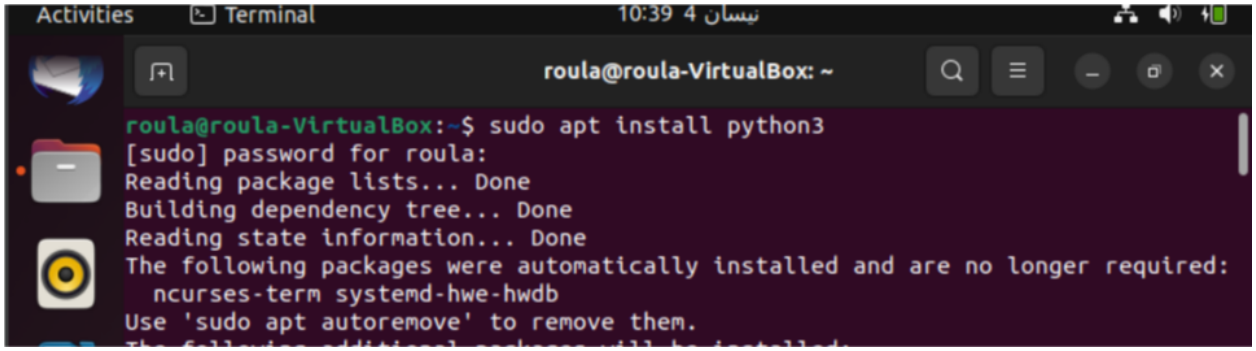


Below is a screenshot of the working and signed-in Ubuntu OS:

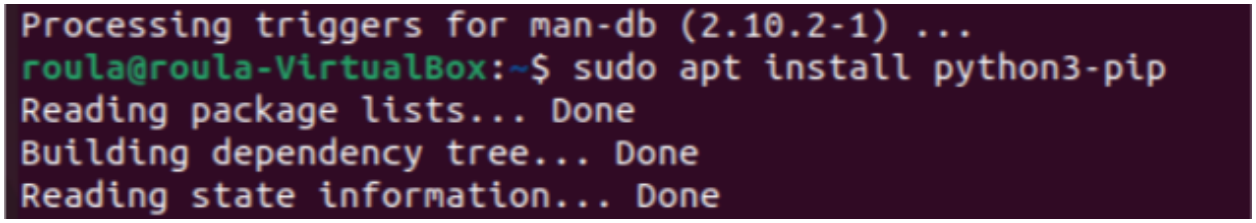


Installing and Running Python

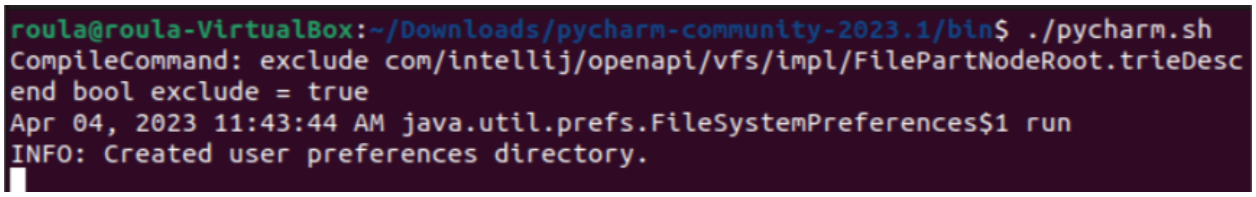
Inside the terminal, we used the command “sudo apt install python3” to install Python 3

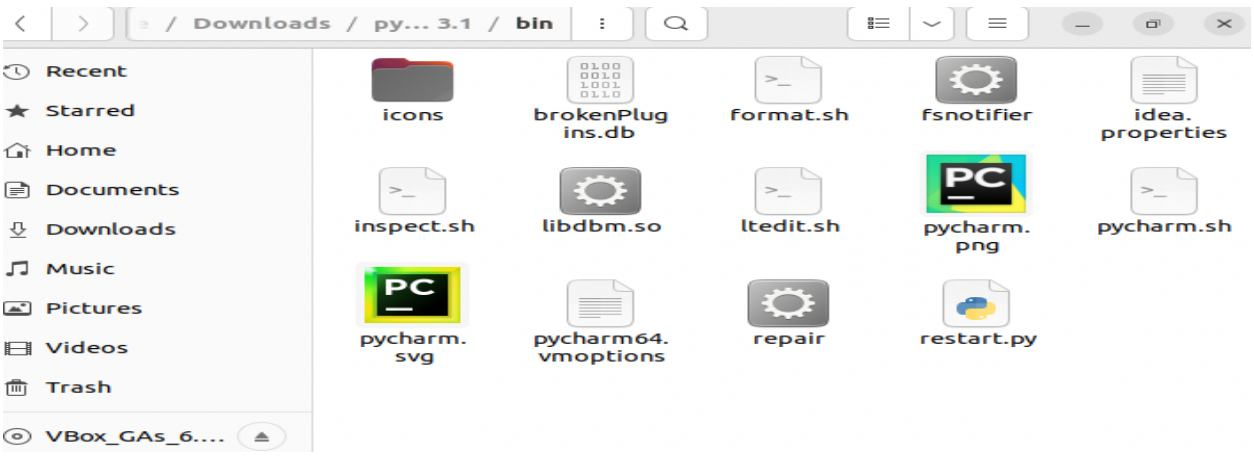


After its successful installation, we installed also Python Pip to be able to manage any packages we might need in the future:

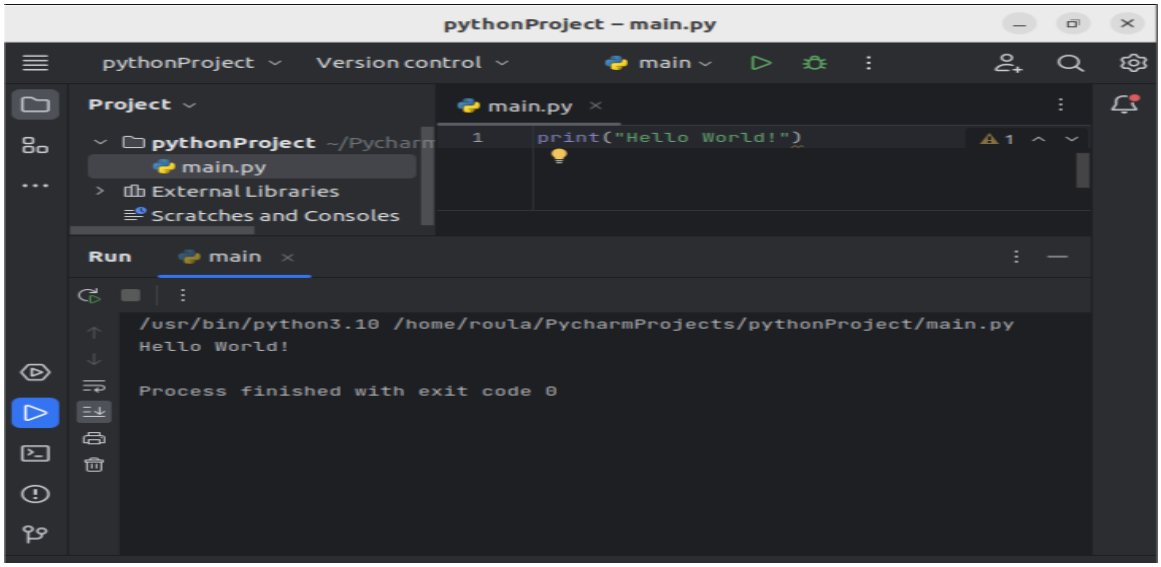


We chose to use PyCharm IDE, after installing it, we ran it using the “./pycharm.sh” command which refers to the PyCharm shell script. We first had to go into the bin folder directory of PyCharm to be able to successfully run this.





After successfully running the command pycharm application opens up and we tested Python in PyCharm by running very basic python code to see if there process runs correctly:



Installing and Running Netem

We used the following tutorial for the installation: https://srtlab.github.io/srt-cookbook/how-to-articles/using-netem-to-emulate-networks.html

First, we had to install some modules to get a fully functional networking emulator. Below are the two main commands we used to correctly install the packages:

```
roula@roula-VirtualBox:~$ sudo apt install linux-modules-extra-$(uname -r)
Reading package lists... Done
```

```
roula@roula-VirtualBox:~$ sudo apt install --reinstall linux-image-generic
Reading package lists... Done
```

We installed the iproute2 package since it contains the netem utility:

```
roula@roula-VirtualBox:~$ sudo apt-get install iproute2
```

After that we checked our device name as it is different than the one provided in the tutorial (enp0s3 in onde):

```
roula@roula-VirtualBox:~$ sudo lshw -C network
  *-network
       description: Ethernet interface
       product: 82540EM Gigabit Ethernet Controller
       vendor: Intel Corporation
       physical id: 3
       bus info: pci@0000:00:03.0
       logical name: enp0s3
       version: 02
```

After that, we added 250ms of delay to packets leaving interface enp0s3:

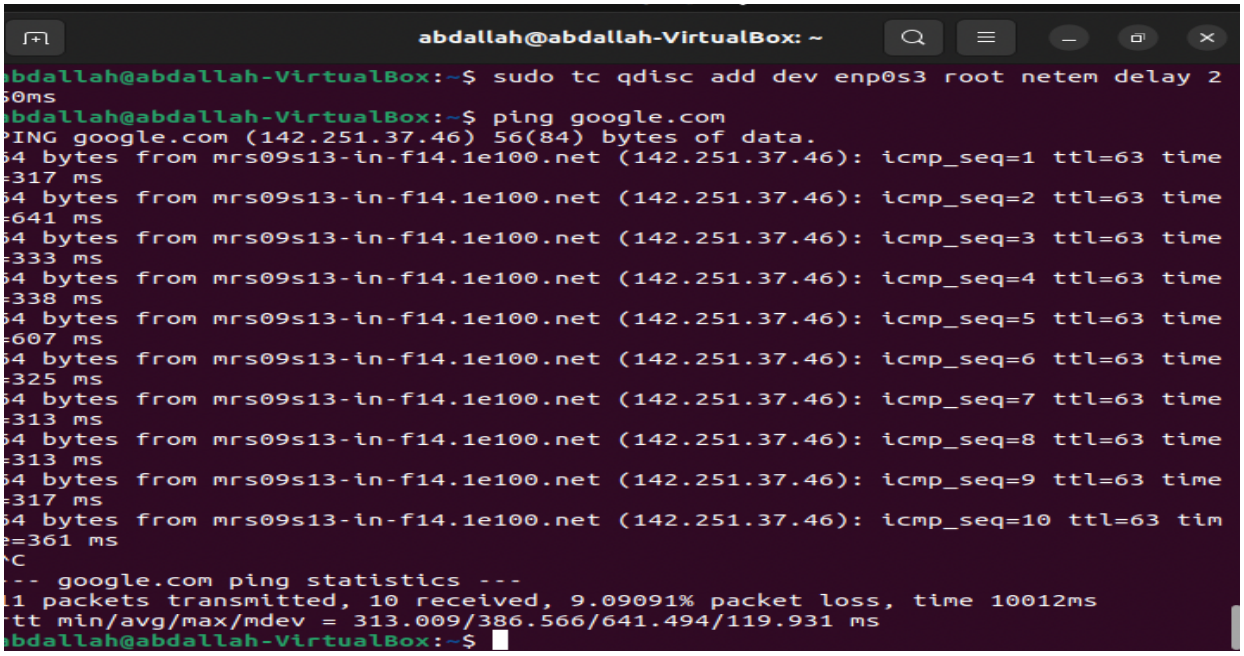
```
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc del dev enp0s3 root
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc add dev enp0s3 root netem delay 2
50ms
abdallah@abdallah-VirtualBox:~$
```

And then we pinged google.com back and check for delay timeWe also tried pinging localhost and check for any delay and we found out there was a indeed a delay.

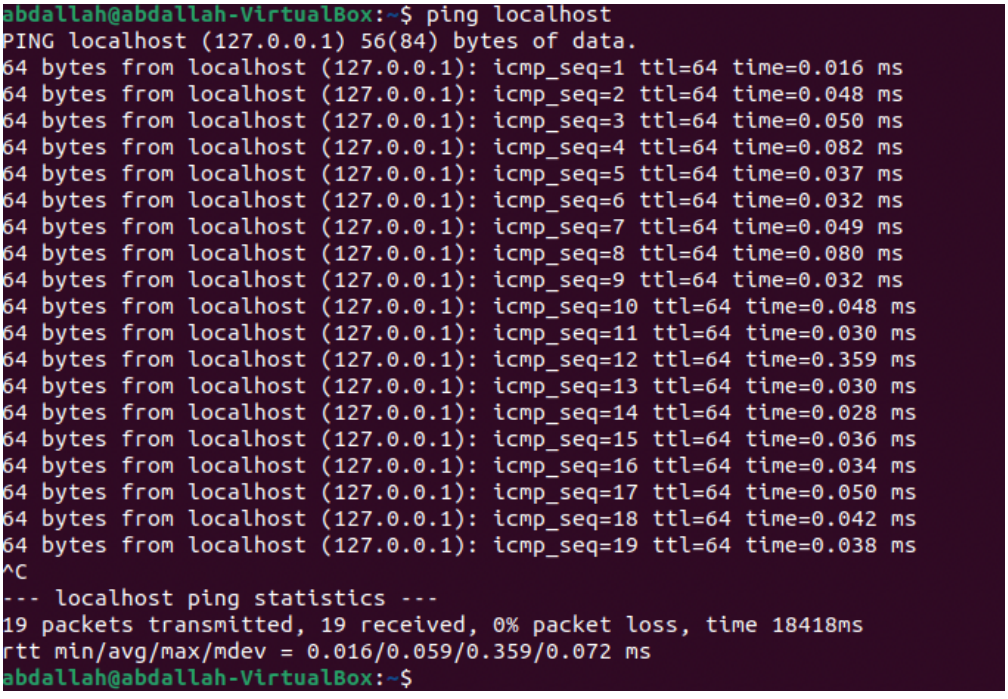
Google ping without delay:

```
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc del dev enp0s3 root
abdallah@abdallah-VirtualBox:~$ ping google.com
Ubuntu Software (142.251.37.46) 56(84) bytes of data.
64 bytes from mrs09s13-in-f14.1e100.net (142.251.37.46): icmp_seq=1 ttl=63 time
=154 ms
64 bytes from mrs09s13-in-f14.1e100.net (142.251.37.46): icmp_seq=3 ttl=63 time
=120 ms
^C
--- google.com ping statistics ---
3 packets transmitted, 2 received, 33.3333% packet loss, time 2016ms
rtt min/avg/max/mdev = 119.713/136.819/153.925/17.106 ms
abdallah@abdallah-VirtualBox:~$ ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.019 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.038 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.052 ms
64 bytes from localhost (127.0.0.1): icmp_seq=4 ttl=64 time=0.053 ms
64 bytes from localhost (127.0.0.1): icmp_seq=5 ttl=64 time=0.051 ms
64 bytes from localhost (127.0.0.1): icmp_seq=6 ttl=64 time=0.036 ms
^C
--- localhost ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5111ms
rtt min/avg/max/mdev = 0.019/0.041/0.053/0.012 ms
abdallah@abdallah-VirtualBox:~$
```

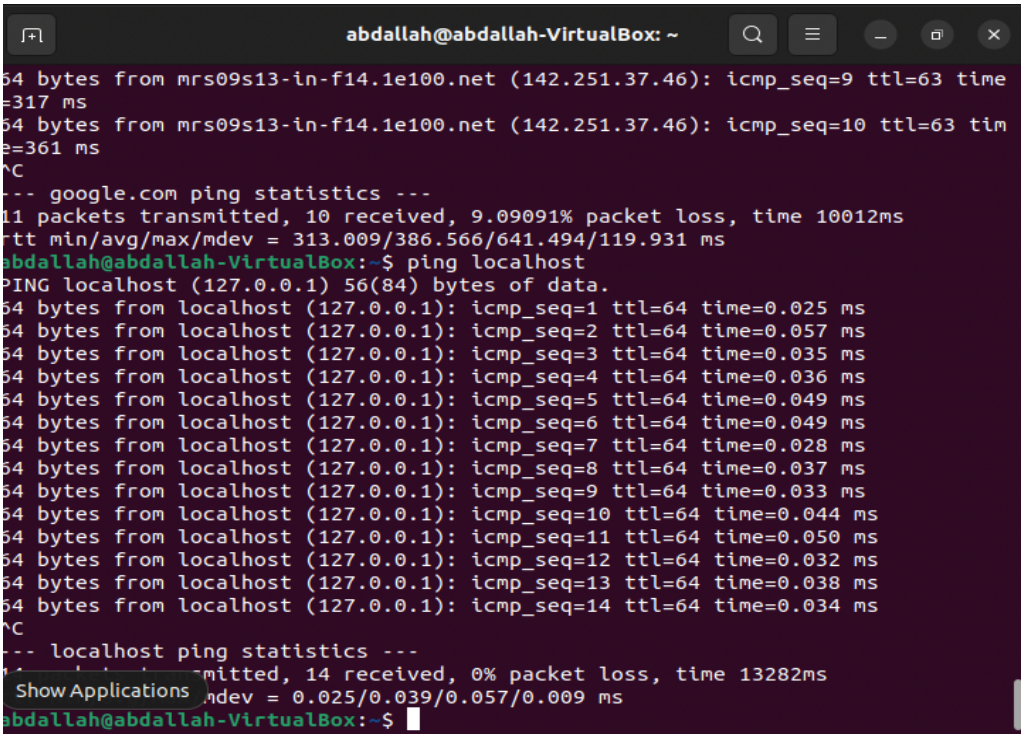
Google ping with delay:



Localhost ping without delay:



Localhost ping with delay:



Some errors we faced:

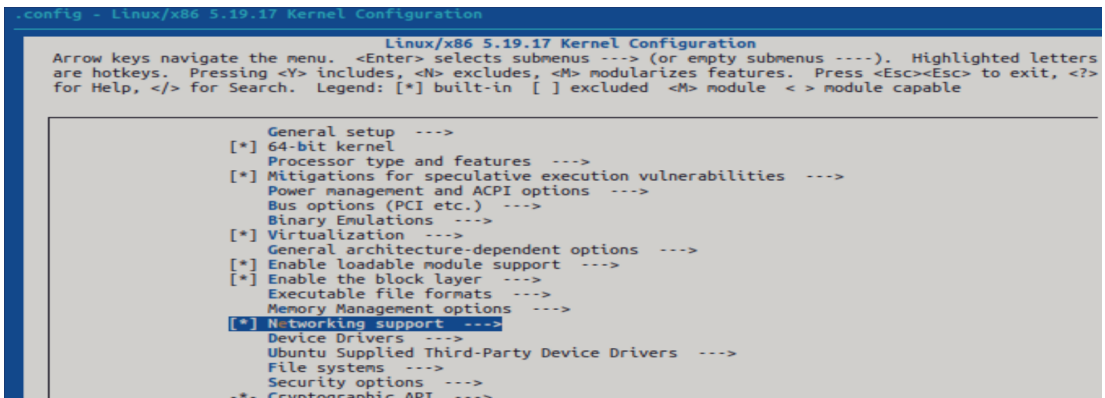
Then we typed ‘sudo modprobe netem’ we got an error while checking if netem was installed properly that looked to be a common error

```
roula@roula-VirtualBox:~$ sudo modprobe netem
[sudo] password for roula:
modprobe: FATAL: Module netem not found in directory /lib/modules/5.19.0-32-gen
eric
```

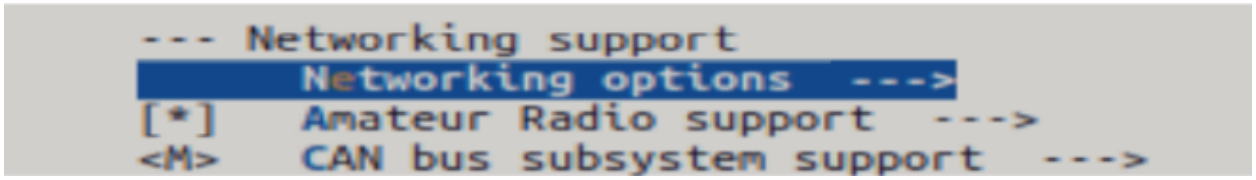
After that we fixed the error by downloading the latest kernel from firefox which is version 6.2.10 the latest kernel package and enabled netem related kernel option after installing the package we updated terminal using ‘$sudo apt update’ and ‘$sudo apt upgrade’ as recommended by the resource website

Then we went into the kernel source directory by running: cd Downloads/linux-6.2.10.tar.xz

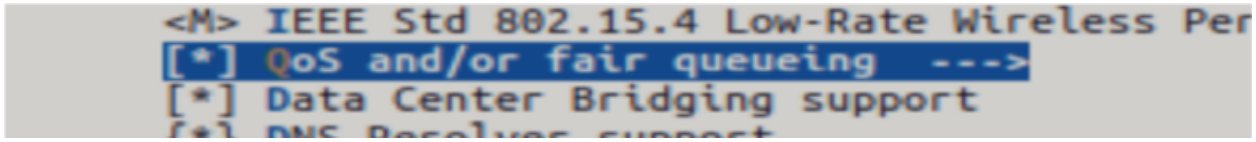
And then running: sudo make menuconfig, we get this menu which we need to choose networking support from:



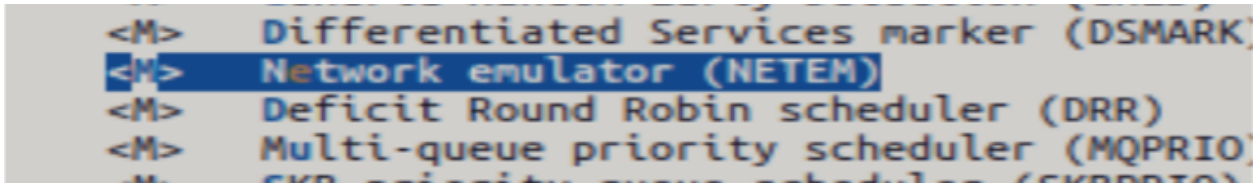
Then clicking networking options

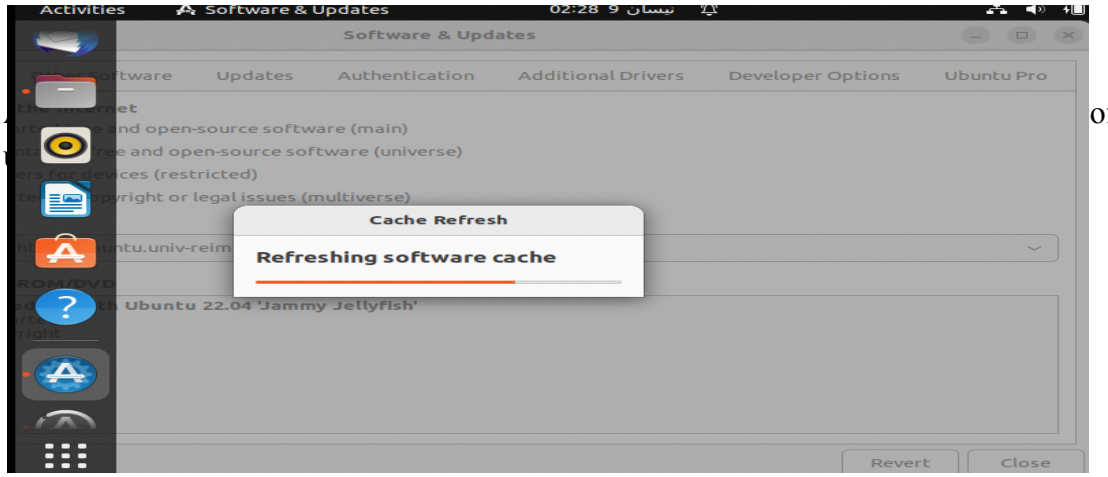


Then clicking QOS and/or fair queuing



and then finding the network emulator (NETEM) module and turning on the options inside:





oftware &

2. Thought Process

As this is a big project with many steps, we decided to make a rough plan to follow with the main steps to be taken. The main reference for this project is from the book Computer Networking: A Top-down Approach 20th Edition by Jim Kurose and Keith Ross released in 2020. We will share this rough step-by-step to give more understanding of our decision-making. We believe this will provide useful insight into our approach's thoughts and methods, and we hope it will improve the clarity of our project and report.

Phase 2:

1. Establish basic connectivity between two peers
Create UDP sockets for the server and client
Bind the server socket to a specified IP address and port number
Server: listen for incoming messages from client
Client: send messages to the server

2. Implement bidirectional communication
Make both peers act as servers and clients simultaneously(one for sending messages and one for receiving messages). Use the select library for help
Ensure both peers are always listening for incoming messages by making both peers listen for incoming messages and send messages simultaneously

3. Implement message reliability
Add sequence numbers
Add acknowledgment messages
Handle duplicate messages by discarding and sending a new acknowledgment, missing messages by message retransmission, and error detection

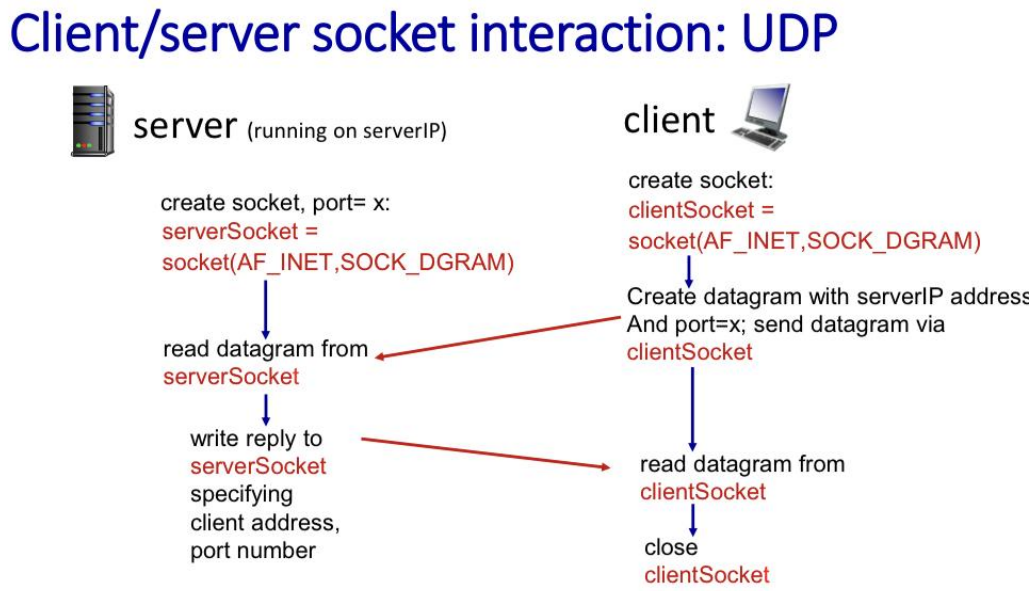
4. Test application under unreliable network conditions
Test with dropped, damaged, delayed, and duplicated packets using netem

Phase 2

Server and Client

a. Basic connectivity between the two peers – One peer is the client sending messages to the other [server] peer over UDP.

Explanation



To form a basic UDP connection between a client and a server, we can use a very basic method as noted by Jim Kurose and Keith Ross(2020). The below diagram from the book "Computer Networking: A Top-Down Approach" clearly demonstrates how this works. The code we will use illustrates the use of sockets for communication over a network using UDP.

The process starts with creating a port on the server side, which involves running on a particular port number. When creating the socket, a datagram socket is needed for UDP (SOCK_DGRAM). Then, the socket, the address family which is called INET as in the internet, indicates that an IPV4 socket is needed.
As for the client side, the client also needs to create a UDP socket, specify both the port number and IP address of the server, and send a datagram through the socket. The server side then needs to be listening and reading the datagram that arrived at this socket. The server then reads the datagram, replies to the same socket, and specifies the IP address and port number that it learned from the incoming packet. The client can then read the packet that it got in return, and it can close the socket if no more packets are to be sent (Kurose & Ross, 2020).

client.py:

```
from socket import *

serverName = '127.0.0.1'
serverPort = 12000
```

```
clientSocket = socket(AF_INET, SOCK_DGRAM)

while True:
   message = input("Enter a message to send to the server or type 'exit' to quit: ")

   if message.lower() == 'exit':
       clientSocket.sendto(message.encode(), (serverName, serverPort))
       break

   clientSocket.sendto(message.encode(), (serverName, serverPort))
   modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
   print(f"Received message from the server: {modifiedMessage.decode()}")

clientSocket.close()
```

The client script (client.py) first establishes a connection to the server by specifying the server's IP address '127.0.0.1' which is the localhost, and the server port number ‘12000'. We then create a UDP socket with the socket(AF_INET, SOCK_DGRAM). After the initializations, the client enters a loop. The user is prompted to input a message to send to the server. If the user types 'exit', the client sends the 'exit' message to the server and breaks out of the loop, which terminates the user’s connection. For any other message, the client sends the message to the server and waits for a response. Upon receiving a response, the client prints the modified message received from the server and continues to prompt the user for input. After the loop is terminated, the client closes the socket using clientSocket.close().

server.py:

```
from socket import *

serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))

print("The server is ready to receive")

while True:
   message, clientAddress = serverSocket.recvfrom(2048)
   decoded_message = message.decode()

   if decoded_message.lower() == 'exit':
       print(f"Client {clientAddress} has left by typing 'exit'")
   else:
```

```
        print(f"Received message: {decoded_message} from {clientAddress}")
        serverSocket.sendto(message, clientAddress)
```

The server listens for incoming messages from the client on the port 120000. To do this, the socket() method is used, which defines the address family and socket type (AF_INET and SOCK_DGRAM since we want this over UDP, respectively). The server then uses the bind() function to connect this socket to the specified port.
Once the server is configured, it enters a loop in which it continuously listens for incoming messages from clients. When a message is received, the server checks to see if the content is 'exit'. If this is the case, the server informs that the client has quit. If the message content is anything other than that, the server broadcasts the message together with the sender's address before returning the original message to the client. We could have used a 'break' statement after a user leaves the connection and terminate the server's connection by closing the socket. We could have also added another way to end the server connection, which is by a client inputting a certain text such as 'terminate_server'. We concluded that this might be a poor decision as a malicious or unkowing user could input this and abnormally terminate the server. So in the end, we kept the server running and accepting clients.

Code Demo

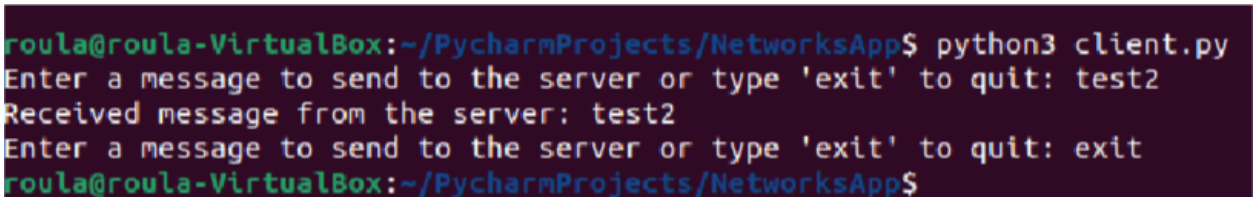
Running server.py

```
roula@roula-VirtualBox:~/PycharmProjects/NetworksApp$ python3 server.py
The server is ready to receive
Received message: test1 from ('127.0.0.1', 50852)
Received message: test2 from ('127.0.0.1', 55686)
Client ('127.0.0.1', 55686) has left by typing 'exit'
Received message: test3 from ('127.0.0.1', 50852)
Client ('127.0.0.1', 50852) has left by typing 'exit'
```

Running client.py

```
roula@roula-VirtualBox:~/PycharmProjects/NetworksApp$ python3 client.py
Enter a message to send to the server or type 'exit' to quit: test1
Received message from the server: test1
Enter a message to send to the server or type 'exit' to quit: test3
Received message from the server: test3
Enter a message to send to the server or type 'exit' to quit: exit
roula@roula-VirtualBox:~/PycharmProjects/NetworksApp$
```

Running another client.py simultaneously:



b. Bidirectional Communication

Explanation

For this section, we had to make our scripts function as both servers and clients, making them peers. This worked using the 'sockets' library, but one thing we found to be a bit faulty. Using UDP only and not TCP, we could not let two peers act as both clients and servers, and have them receive messages independently, without having them wait for a message from their peer first. An appropriate library we found to use was the 'select' library. The reason for our choice was that the select library goes hand-in-hand with the socket library. It helps manage multiple sockets simultaneously. It helps efficiently monitor socket states and smoothly handles multiple connections. Other ways to do this included using multi-threading or I/O blocking methods. After some research and trial, we found the 'select' library to be the simplest and most concise way to achieve bidirectional communication. This library also works with both UDP and TCP transport protocols. Moreover, we found out that there was an issue with the input() method because it blocks the program until input is provided. The 'select' and 'sys' libraries help with this issue as they check for the user input and the incoming messages. The explanation for these will be provided along with their code below.

We decided to rename our scripts into peer1 and peer2 instead of client and server.
They are similar code-wise, but have differing server and peer numbers - they are switched in each. For now, when one peer ends his their connection, the other peer's connection also ends.

Now , to explain how the code works, we initialize the server and peer addresses. Then a peer will bind to its server address and communicate with the other peer using its peer address. Then of course, we have to create the UDP socket and bind it to the server address. Now, our peers are ready to receive and send messages. We included sys.stdin to monitor user input. Once the select.select() method takes as input the sys.stdin the program can know if there is standard user input or not. All in all, this helps us monitor both sending and receiving messages to receive data

continuously. The select.select() method checks if there is input on the sys.stdin. If there is, it means that a client has typed a message, so the program checks if the message is ‘exit’ to terminate the connection. If it is not ‘exit’, the message is sent to the other peer. Then, the program checks to see if there is a message on the socket, if there is that means a message has been successfully sent and it received it. After decoding it, it checks whether it is an ‘exit’ to display the appropriate message or whether it is a normal message to be displayed.

peer1.py:

```
import sys
import socket
import select

server_address = ('127.0.0.1', 12000)
peer_address = ('127.0.0.1', 12001)

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(server_address)

print("Peer 1 ready to send and receive messages")

while True:
    ready_to_read, _, _ = select.select([sys.stdin, sock], [], [])

    for s in ready_to_read:
        if s == sys.stdin:
            message = input()
            if message.lower() == 'exit':
                sock.sendto(message.encode(), peer_address)
                sock.close()
                sys.exit()

            sock.sendto(message.encode(), peer_address)
        elif s == sock:
            received_message, _ = sock.recvfrom(2048)
            decoded_message = received_message.decode()
            if decoded_message.lower() == 'exit':
                print("Peer 2 has left by typing 'exit'")
                sock.close()
                sys.exit()
            else:
```

```
                print(f"Received message from Peer 2: {decoded_message}")
```

peer2.py:

```
import sys
import socket
import select

server_address = ('127.0.0.1', 12001)
peer_address = ('127.0.0.1', 12000)

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(server_address)

print("Peer 2 ready to send and receive messages")

while True:
    ready_to_read, _, _ = select.select([sys.stdin, sock], [], [])

    for s in ready_to_read:
        if s == sys.stdin:
            message = input()
            if message.lower() == 'exit':
                sock.sendto(message.encode(), peer_address)
                sock.close()
                sys.exit()

            sock.sendto(message.encode(), peer_address)
        elif s == sock:
            received_message, _ = sock.recvfrom(2048)
            decoded_message = received_message.decode()
            if decoded_message.lower() == 'exit':
                print("Peer 1 has left by typing 'exit'")
                sock.close()
                sys.exit()
            else:
                print(f"Received message from Peer 1: {decoded_message}")
```

Code demo:

peer1.py:

```
roula@roula-VirtualBox:~/PycharmProjects/NetworksApp$ python3 peer1.py
Peer 1 ready to send and receive messages
pls work
Received message from Peer 2: yay it worked
Peer 2 has left by typing 'exit'
roula@roula-VirtualBox:~/PycharmProjects/NetworksApp$
```

peer2.py:

```
roula@roula-VirtualBox:~/PycharmProjects/NetworksApp$ python3 peer2.py
Peer 2 ready to send and receive messages
Received message from Peer 1: pls work
yay it worked
exit
```

c. Message Reliability

To make the messages reliable, we will be using the “stop-and-wait” approach that adds acknowledgement at the receiver side before another message is sent by a sender as Jim Kurose and Keith Ross(2020) mentioned. So, we will be adding acks, sequence numbers, and timeouts.

peer1.py

```
import sys
import socket
import select
import time
import PySimpleGUI as sg #for the GUI

CRC32_TABLE = [
   0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f,
   0xe963a535, 0x9e6495a3, 0x0edb8832, 0x79dcb8a4, 0xe0d5e91e, 0x97d2d988,
   0x09b64c2b, 0x7eb17cbd, 0xe7b82d07, 0x90bf1d91, 0x1db71064, 0x6ab020f2,
   0xf3b97148, 0x84be41de, 0x1adad47d, 0x6ddde4eb, 0xf4d4b551, 0x83d385c7,
   0x136c9856, 0x646ba8c0, 0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9,
   0xfa0f3d63, 0x8d080df5, 0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172,
   0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b, 0x35b5a8fa, 0x42b2986c,
   0xdbbbc9d6, 0xacbcf940, 0x32d86ce3, 0x45df5c75, 0xdcd60dcf, 0xabd13d59,
   0x26d930ac, 0x51de003a, 0xc8d75180, 0xbfd06116, 0x21b4f4b5, 0x56b3c423,
   0xcfba9599, 0xb8bda50f, 0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924,
   0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d, 0x76dc4190, 0x01db7106,
```

```
    0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f, 0x9fbfe4a5, 0xe8b8d433,
    0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818, 0x7f6a0dbb, 0x086d3d2d,
    0x91646c97, 0xe6635c01, 0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e,
    0x6c0695ed, 0x1b01a57b, 0x8208f4c1, 0xf50fc457, 0x65b0d9c6, 0x12b7e950,
    0x8bbeb8ea, 0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65,
    0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2, 0x4adfa541, 0x3dd895d7,
    0xa4d1c46d, 0xd3d6f4fb, 0x4369e96a, 0x346ed9fc, 0xad678846, 0xda60b8d0,
    0x44042d73, 0x33031de5, 0xaa0a4c5f, 0xdd0d7cc9, 0x5005713c, 0x270241aa,
    0xbe0b1010, 0xc90c2086, 0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
    0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17, 0x2eb40d81,
    0xb7bd5c3b, 0xc0ba6cad, 0xedb88320, 0x9abfb3b6, 0x03b6e20c, 0x74b1d29a,
    0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683, 0xe3630b12, 0x94643b84,
    0x0d6d6a3e, 0x7a6a5aa8, 0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1,
    0xf00f9344, 0x8708a3d2, 0x1e01f268, 0x6906c2fe, 0xf762575d, 0x806567cb,
    0x196c3671, 0x6e6b06e7, 0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc,
    0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5, 0xd6d6a3e8, 0xa1d1937e,
    0x38d8c2c4, 0x4fdff252, 0xd1bb67f1, 0xa6bc5767, 0x3fb506dd, 0x48b2364b,
    0xd80d2bda, 0xaf0a1b4c, 0x36034af6, 0x41047a60, 0xdf60efc3, 0xa867df55,
    0x316e8eef, 0x4669be79, 0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236,
    0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f, 0xc5ba3bbe, 0xb2bd0b28,
    0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b, 0x5bdeae1d,
    0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a, 0x9c0906a9, 0xeb0e363f,
    0x72076785, 0x05005713, 0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38,
    0x92d28e9b, 0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21, 0x86d3d2d4, 0xf1d4e242,
    0x68ddb3f8, 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777,
    0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c, 0x8f659eff, 0xf862ae69,
    0x616bffd3, 0x166ccf45, 0xa00ae278, 0xd70dd2ee, 0x4e048354, 0x3903b3c2,
    0xa7672661, 0xd06016f7, 0x4969474d, 0x3e6e77db, 0xaed16a4a, 0xd9d65adc,
    0x40df0b66, 0x37d83bf0, 0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f, 0x30b5ffe9,
    0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605, 0xcdd70693,
    0x54de5729, 0x23d967bf, 0xb3667a2e, 0xc4614ab8, 0x5d681b02, 0x2a6f2b94,
    0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d]

RETRANSMISSION_TIMEOUT = 2
MAX_RETRANSMISSIONS = 6
RETRANSMISSION_LIMIT = 60  #1 min

def crc32(data):
    crc = 0xffffffff
    for byte in data:
        crc = (crc >> 8) ^ CRC32_TABLE[(crc ^ byte) & 0xff]
    return crc ^ 0xffffffff


def send_message(message, sequence_nb, retransmissions=0):
    if message.lower() == 'exit':
        sock.sendto(message.encode(), peer_address)
```

```
        sock.close()
        sys.exit()

    crc32_value = crc32(message.encode())
    message = f"{sequence_nb}: {crc32_value}: {message}"
    sock.sendto(message.encode(), peer_address)
    unacknowledged_messages[sequence_nb] = (message, time.time(), retransmissions)

    if retransmissions > 0:
        print(f"Retransmitting message {sequence_nb} (retransmission
{retransmissions}): {message}")

    return sequence_nb + 1


def receive_message(sock):
    received_message, _ = sock.recvfrom(2048)
    decoded_message = received_message.decode()

    if decoded_message.lower() == 'exit':
        print("Peer 2 has left by typing or pressing 'exit'")
        sock.close()
        sys.exit()
    elif decoded_message.startswith("ACK:"):
        ack_sequence_nb = int(decoded_message.split(" ")[1])
        unacknowledged_messages.pop(ack_sequence_nb, None)
        print(f"Received ACK for message {ack_sequence_nb}")
    else:
        message_sequence_nb, crc32_received, message_text = decoded_message.split(":
", 2)
        message_sequence_nb = int(message_sequence_nb)
        crc32_received = int(crc32_received)

        if crc32(message_text.encode()) == crc32_received:
            if message_sequence_nb not in received_messages:
                received_messages[message_sequence_nb] = message_text
                print(f"Received message from Peer 2: {message_text}")

            ack_message = f"ACK: {message_sequence_nb}"
            sock.sendto(ack_message.encode(), peer_address)

        else:
            print("CRC32 mismatch, discarding message")

server_address = ('127.0.0.1', 12000)
peer_address = ('127.0.0.1', 12001)
```

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(server_address)
sock.setblocking(0)

print("Peer 1 ready to send and receive messages")

sequence_nb = 1
unacknowledged_messages = {}
received_messages = {}
last_retransmission_check = time.time()

#GUI

sg.theme('DarkTeal2')

layout = [[sg.Multiline(size=(100, 30), key='-OUTPUT-', autoscroll=True,
disabled=True, reroute_stdout=True, text_color='white')],
          [sg.Input(key='-INPUT-', size=(90, 2))],
          [sg.Button('Send', button_color=('white', 'darkgreen'), border_width=5,
pad=(10, 10)),
           sg.Button('Exit', button_color=('white', 'darkred'), border_width=5,
pad=(10, 10))]]

window = sg.Window('Peer 1', layout, size=(1000, 800),
element_justification='center')


def retransmit_dropped_packet(sequence_nb, message, retransmissions):
   return send_message(message, sequence_nb, retransmissions + 1)

while True:
   event, values = window.read(timeout=100)

   if event in (sg.WIN_CLOSED, 'Exit'):
       sock.sendto("exit".encode(), peer_address)
       sock.close()
       break

   elif event == 'Send':
       message = values['-INPUT-']
       crc32_value = crc32(message.encode())
       sequence_nb = send_message(message, sequence_nb, 0)
       window['-INPUT-'].update('')
       window['-OUTPUT-'].update('', append=True, text_color_for_value='white',
font=("Helvetica", 10, "bold"))

   ready_to_read, _, _ = select.select([sock], [], [], 0)
```

```
   for s in ready_to_read:
       if s == sock:
           received_msg = receive_message(sock)
           if received_msg:
               print(received_msg)

   current_time = time.time()
   if current_time - last_retransmission_check > RETRANSMISSION_TIMEOUT:
       for seq_nb, (message, timestamp, retransmissions) in
list(unacknowledged_messages.items()):
           if current_time - timestamp > RETRANSMISSION_TIMEOUT:
               if retransmissions < MAX_RETRANSMISSIONS:
                   unacknowledged_messages.pop(seq_nb, None)
                   retransmit_dropped_packet(seq_nb, message.split(': ', 2)[-1],
retransmissions)
               else:
                   print(f"Message {seq_nb} dropped after reaching maximum
retransmissions")
                   unacknowledged_messages.pop(seq_nb, None)

       last_retransmission_check = current_time

   #formatting the GUI
   window['-OUTPUT-'].Widget.configure(highlightthickness=2,
highlightbackground='coral')
   window['-INPUT-'].Widget.configure(highlightthickness=2,
highlightbackground='coral')
   window['-OUTPUT-'].set_vscroll_position(1)

   #italicizing and coloring ACKs and errors
   output_widget = window['-OUTPUT-'].Widget
   output_widget.tag_configure("ACK", foreground="blue", font=("Helvetica", 10,
"italic"))
   output_widget.tag_configure("error", foreground="red", font=("Helvetica", 10,
"italic"))

   lines = output_widget.get("1.0", "end-1c").split("\n")
   for i, line in enumerate(lines):
       if "Received ACK" in line:
           output_widget.tag_add("ACK", f"{i + 1}.0", f"{i + 1}.end")
       elif "CRC32 mismatch" in line:
           output_widget.tag_add("error", f"{i + 1}.0", f"{i + 1}.end")
           window.close()
```

peer2.py

```
import sys
import socket
import select
import time
import PySimpleGUI as sg

CRC32_TABLE = [
   0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f,
   0xe963a535, 0x9e6495a3, 0x0edb8832, 0x79dcb8a4, 0xe0d5e91e, 0x97d2d988,
   0x09b64c2b, 0x7eb17cbd, 0xe7b82d07, 0x90bf1d91, 0x1db71064, 0x6ab020f2,
   0xf3b97148, 0x84be41de, 0x1adad47d, 0x6ddde4eb, 0xf4d4b551, 0x83d385c7,
   0x136c9856, 0x646ba8c0, 0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9,
   0xfa0f3d63, 0x8d080df5, 0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172,
   0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b, 0x35b5a8fa, 0x42b2986c,
   0xdbbbc9d6, 0xacbcf940, 0x32d86ce3, 0x45df5c75, 0xdcd60dcf, 0xabd13d59,
   0x26d930ac, 0x51de003a, 0xc8d75180, 0xbfd06116, 0x21b4f4b5, 0x56b3c423,
   0xcfba9599, 0xb8bda50f, 0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924,
   0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d, 0x76dc4190, 0x01db7106,
   0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f, 0x9fbfe4a5, 0xe8b8d433,
   0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818, 0x7f6a0dbb, 0x086d3d2d,
   0x91646c97, 0xe6635c01, 0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e,
   0x6c0695ed, 0x1b01a57b, 0x8208f4c1, 0xf50fc457, 0x65b0d9c6, 0x12b7e950,
   0x8bbeb8ea, 0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65,
   0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2, 0x4adfa541, 0x3dd895d7,
   0xa4d1c46d, 0xd3d6f4fb, 0x4369e96a, 0x346ed9fc, 0xad678846, 0xda60b8d0,
   0x44042d73, 0x33031de5, 0xaa0a4c5f, 0xdd0d7cc9, 0x5005713c, 0x270241aa,
   0xbe0b1010, 0xc90c2086, 0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
   0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17, 0x2eb40d81,
   0xb7bd5c3b, 0xc0ba6cad, 0xedb88320, 0x9abfb3b6, 0x03b6e20c, 0x74b1d29a,
   0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683, 0xe3630b12, 0x94643b84,
   0x0d6d6a3e, 0x7a6a5aa8, 0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1,
   0xf00f9344, 0x8708a3d2, 0x1e01f268, 0x6906c2fe, 0xf762575d, 0x806567cb,
   0x196c3671, 0x6e6b06e7, 0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc,
   0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5, 0xd6d6a3e8, 0xa1d1937e,
   0x38d8c2c4, 0x4fdff252, 0xd1bb67f1, 0xa6bc5767, 0x3fb506dd, 0x48b2364b,
   0xd80d2bda, 0xaf0a1b4c, 0x36034af6, 0x41047a60, 0xdf60efc3, 0xa867df55,
   0x316e8eef, 0x4669be79, 0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236,
   0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f, 0xc5ba3bbe, 0xb2bd0b28,
   0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b, 0x5bdeae1d,
   0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a, 0x9c0906a9, 0xeb0e363f,
   0x72076785, 0x05005713, 0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38,
   0x92d28e9b, 0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21, 0x86d3d2d4, 0xf1d4e242,
   0x68ddb3f8, 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777,
   0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c, 0x8f659eff, 0xf862ae69,
   0x616bffd3, 0x166ccf45, 0xa00ae278, 0xd70dd2ee, 0x4e048354, 0x3903b3c2,
   0xa7672661, 0xd06016f7, 0x4969474d, 0x3e6e77db, 0xaed16a4a, 0xd9d65adc,
```

```
    0x40df0b66, 0x37d83bf0, 0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f, 0x30b5ffe9,
    0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605, 0xcdd70693,
    0x54de5729, 0x23d967bf, 0xb3667a2e, 0xc4614ab8, 0x5d681b02, 0x2a6f2b94,
    0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d]

RETRANSMISSION_TIMEOUT = 2

def crc32(data):
    crc = 0xffffffff
    for byte in data:
        crc = (crc >> 8) ^ CRC32_TABLE[(crc ^ byte) & 0xff]
    return crc ^ 0xffffffff
def send_message(message, sequence_nb):
    if message.lower() == 'exit':
        sock.sendto(message.encode(), peer_address)
        sock.close()
        sys.exit()

    crc32_value = crc32(message.encode())
    message = f"{sequence_nb}: {crc32_value}: {message}"
    sock.sendto(message.encode(), peer_address)
    unacknowledged_messages[sequence_nb] = (message, time.time())
    return sequence_nb + 1


def receive_message(sock):
    received_message, _ = sock.recvfrom(2048)
    decoded_message = received_message.decode()

    if decoded_message.lower() == 'exit':
        print("Peer 1 has left by typing 'exit'")
        sock.close()
        sys.exit()
    elif decoded_message.startswith("ACK:"):
        ack_sequence_nb = int(decoded_message.split(" ")[1])
        unacknowledged_messages.pop(ack_sequence_nb, None)
        print(f"Received ACK for message {ack_sequence_nb}")
    else:
        message_sequence_nb, crc32_received, message_text =
decoded_message.split(": ", 2)
        message_sequence_nb = int(message_sequence_nb)
        crc32_received = int(crc32_received)

        if crc32(message_text.encode()) == crc32_received:
            if message_sequence_nb not in received_messages:
                received_messages[message_sequence_nb] = message_text
                print(f"Received message from Peer 1: {message_text}")
```

```
            ack_message = f"ACK: {message_sequence_nb}"
            sock.sendto(ack_message.encode(), peer_address)

        else:
            print("CRC32 mismatch, discarding message")


server_address = ('127.0.0.1', 12001)
peer_address = ('127.0.0.1', 12000)

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(server_address)
sock.setblocking(0)

print("Peer 2 ready to send and receive messages")

sequence_nb = 1
unacknowledged_messages = {}
received_messages = {}
last_retransmission_check = time.time()

#GUI

sg.theme('DarkTeal3')

layout = [[sg.Multiline(size=(100, 30), key='-OUTPUT-', autoscroll=True,
disabled=True, reroute_stdout=True, text_color='white')],
          [sg.Input(key='-INPUT-', size=(90, 2))],
          [sg.Button('Send', button_color=('white', 'darkgreen'), border_width=5,
pad=(10, 10)),
           sg.Button('Exit', button_color=('white', 'darkred'), border_width=5,
pad=(10, 10))]]

window = sg.Window('Peer 2', layout, size=(1000, 800),
element_justification='center')


while True:
   event, values = window.read(timeout=100)

   if event in (sg.WIN_CLOSED, 'Exit'):
       sock.sendto("exit".encode(), peer_address)
       sock.close()
       break

   elif event == 'Send':
```

```
        message = values['-INPUT-']
        crc32_value = crc32(message.encode())
        sequence_nb = send_message(message, sequence_nb)
        window['-INPUT-'].update('')
        #GUI making sent message bold
        window['-OUTPUT-'].update('', append=True, text_color_for_value='white',
font=("Helvetica", 10, "bold"))

    ready_to_read, _, _ = select.select([sock], [], [], 0)

    for s in ready_to_read:
        if s == sock:
            received_msg = receive_message(sock)
            if received_msg:
                print(received_msg)

    current_time = time.time()
    if current_time - last_retransmission_check >= RETRANSMISSION_TIMEOUT:
        for seq_nb, (message, timestamp) in list(unacknowledged_messages.items()):
            if current_time - last_retransmission_check >= RETRANSMISSION_TIMEOUT:
                for seq_nb, (message, timestamp) in
list(unacknowledged_messages.items()):
                    if current_time - timestamp >= RETRANSMISSION_TIMEOUT:
                        if seq_nb in unacknowledged_messages:
                            print(f"Retransmitting message: {message}")
                            sock.sendto(message.encode(), peer_address)
                            unacknowledged_messages[seq_nb] = (message,
current_time)
                    else:
                        break
                last_retransmission_check = current_time

            #formatting the GUI
            window['-OUTPUT-'].Widget.configure(highlightthickness=2,
highlightbackground='coral')
            window['-INPUT-'].Widget.configure(highlightthickness=2,
highlightbackground='coral')
            window['-OUTPUT-'].set_vscroll_position(1)

            #italicizing and coloring ACKs and errors
            output_widget = window['-OUTPUT-'].Widget
            output_widget.tag_configure("ACK", foreground="blue",
font=("Helvetica", 10, "italic"))
            output_widget.tag_configure("error", foreground="red",
font=("Helvetica", 10, "italic"))

            lines = output_widget.get("1.0", "end-1c").split("\n")
```

```
            for i, line in enumerate(lines):
                if "Received ACK" in line:
                    output_widget.tag_add("ACK", f"{i + 1}.0", f"{i + 1}.end")
                elif "CRC32 mismatch" in line:
                    output_widget.tag_add("error", f"{i + 1}.0", f"{i + 1}.end")

window.close()




Phase 3
Bonus
We used PySimpleGUI. For it to work we had to install this module and Tkinter.
PySimpleGui is based on Tkint
```

This is a UDP paper to peer chatting application code.we implemented a GUI.
It uses crc32 checksum with a predefined table. We also added sequence numbers for sending and receiving and checksum value. We also added a transmission for unreached messaged, and after a certain time limit or six retransmits they get dropped, and the user knows of all this. The user types their message in the input area and clicks the Send button to send it. A sequence number and checksum value are then delivered with the message to the other peer. The we wait for the other peer's ack message. The message is taken from the list of unacknowledged messages if an ACK is received. Also then, the program retransmits the message if it is not acknowledged after a timeout period.
This trustworthy because it uses the CRC32 checksum to implement error detection, ensuring that the received message is not corrupted. The program also manages the retransmission of unacknowledged messages, ensuring that all messages are eventually delivered.

Phase 3

Peer1.py

```
import sys
import socket
import select
import time
import PySimpleGUI as sg #for the GUI
import os



CRC32_TABLE = [
    0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f,
    0xe963a535, 0x9e6495a3, 0x0edb8832, 0x79dcb8a4, 0xe0d5e91e, 0x97d2d988,
```

```
    0x09b64c2b, 0x7eb17cbd, 0xe7b82d07, 0x90bf1d91, 0x1db71064, 0x6ab020f2,
    0xf3b97148, 0x84be41de, 0x1adad47d, 0x6ddde4eb, 0xf4d4b551, 0x83d385c7,
    0x136c9856, 0x646ba8c0, 0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9,
    0xfa0f3d63, 0x8d080df5, 0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172,
    0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b, 0x35b5a8fa, 0x42b2986c,
    0xdbbbc9d6, 0xacbcf940, 0x32d86ce3, 0x45df5c75, 0xdcd60dcf, 0xabd13d59,
    0x26d930ac, 0x51de003a, 0xc8d75180, 0xbfd06116, 0x21b4f4b5, 0x56b3c423,
    0xcfba9599, 0xb8bda50f, 0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924,
    0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d, 0x76dc4190, 0x01db7106,
    0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f, 0x9fbfe4a5, 0xe8b8d433,
    0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818, 0x7f6a0dbb, 0x086d3d2d,
    0x91646c97, 0xe6635c01, 0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e,
    0x6c0695ed, 0x1b01a57b, 0x8208f4c1, 0xf50fc457, 0x65b0d9c6, 0x12b7e950,
    0x8bbeb8ea, 0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65,
    0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2, 0x4adfa541, 0x3dd895d7,
    0xa4d1c46d, 0xd3d6f4fb, 0x4369e96a, 0x346ed9fc, 0xad678846, 0xda60b8d0,
    0x44042d73, 0x33031de5, 0xaa0a4c5f, 0xdd0d7cc9, 0x5005713c, 0x270241aa,
    0xbe0b1010, 0xc90c2086, 0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
    0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17, 0x2eb40d81,
    0xb7bd5c3b, 0xc0ba6cad, 0xedb88320, 0x9abfb3b6, 0x03b6e20c, 0x74b1d29a,
    0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683, 0xe3630b12, 0x94643b84,
    0x0d6d6a3e, 0x7a6a5aa8, 0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1,
    0xf00f9344, 0x8708a3d2, 0x1e01f268, 0x6906c2fe, 0xf762575d, 0x806567cb,
    0x196c3671, 0x6e6b06e7, 0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc,
    0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5, 0xd6d6a3e8, 0xa1d1937e,
    0x38d8c2c4, 0x4fdff252, 0xd1bb67f1, 0xa6bc5767, 0x3fb506dd, 0x48b2364b,
    0xd80d2bda, 0xaf0a1b4c, 0x36034af6, 0x41047a60, 0xdf60efc3, 0xa867df55,
    0x316e8eef, 0x4669be79, 0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236,
    0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f, 0xc5ba3bbe, 0xb2bd0b28,
    0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b, 0x5bdeae1d,
    0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a, 0x9c0906a9, 0xeb0e363f,
    0x72076785, 0x05005713, 0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38,
    0x92d28e9b, 0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21, 0x86d3d2d4, 0xf1d4e242,
    0x68ddb3f8, 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777,
    0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c, 0x8f659eff, 0xf862ae69,
    0x616bffd3, 0x166ccf45, 0xa00ae278, 0xd70dd2ee, 0x4e048354, 0x3903b3c2,
    0xa7672661, 0xd06016f7, 0x4969474d, 0x3e6e77db, 0xaed16a4a, 0xd9d65adc,
    0x40df0b66, 0x37d83bf0, 0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f, 0x30b5ffe9,
    0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605, 0xcdd70693,
    0x54de5729, 0x23d967bf, 0xb3667a2e, 0xc4614ab8, 0x5d681b02, 0x2a6f2b94,
    0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d]

RETRANSMISSION_TIMEOUT = 2
MAX_RETRANSMISSIONS = 6
RETRANSMISSION_LIMIT = 60  #1 min
BUFFER_SIZE = 10000
HOST = '127.0.0.1'
FILE_PORT = 1234
```

```
def crc32(data):
   crc = 0xffffffff
   for byte in data:
       crc = (crc >> 8) ^ CRC32_TABLE[(crc ^ byte) & 0xff]
   return crc ^ 0xffffffff

def send_message(message, sequence_nb, retransmissions=0):
   if message.lower() == 'exit':
       sock.sendto(message.encode(), peer_address)
       sock.close()
       sys.exit()

   crc32_value = crc32(message.encode())
   message = f"{sequence_nb}: {crc32_value}: {message}"
   sock.sendto(message.encode(), peer_address)
   unacknowledged_messages[sequence_nb] = (message, time.time(),
retransmissions)

   if retransmissions > 0:
       print(f"Retransmitting message {sequence_nb} (retransmission
{retransmissions}): {message}")

   return sequence_nb + 1


def receive_message(sock):
   received_message, _ = sock.recvfrom(2048)
   decoded_message = received_message.decode()

   if decoded_message.lower() == 'exit':
       print("Peer 2 has left by typing or pressing 'exit'")
       sock.close()
       sys.exit()
   elif decoded_message.startswith("ACK:"):
       ack_sequence_nb = int(decoded_message.split(" ")[1])
       unacknowledged_messages.pop(ack_sequence_nb, None)
       print(f"Received ACK for message {ack_sequence_nb}")
   else:
       message_sequence_nb, crc32_received, message_text =
decoded_message.split(": ", 2)
       message_sequence_nb = int(message_sequence_nb)
       crc32_received = int(crc32_received)

       if crc32(message_text.encode()) == crc32_received:
           if message_sequence_nb not in received_messages:
               received_messages[message_sequence_nb] = message_text
               print(f"Received message from Peer 2: {message_text}")

           ack_message = f"ACK: {message_sequence_nb}"
```

```
            sock.sendto(ack_message.encode(), peer_address)

        else:
            print("CRC32 mismatch, discarding message")

def send_file(file_path):
    # Send a file to the other peer
    with open(file_path, 'rb') as file:
        data = file.read()

    # Send the file in packets
    packet_size = 1024
    packets = [data[i:i+packet_size] for i in range(0, len(data), packet_size)]
    num_packets = len(packets)
    send_message(f"START {num_packets} {file_path}", 0)

    for i, packet in enumerate(packets, start=1):
        sequence_nb = send_message(packet, i)
        time.sleep(0.001) # wait a little between packets to not overwhelm the
network

    send_message("END", num_packets+1)

def receive_file():
    # Receive a file from the other peer
    file_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    file_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    file_sock.bind((HOST, FILE_PORT))
    file_sock.listen(1)

    print('Waiting for incoming file...')
    conn, addr = file_sock.accept()
    print('File incoming from: ', addr)

    # Receive file name
    file_name = conn.recv(BUFFER_SIZE).decode()
    print('File name: ', file_name)

    # Receive file size
    file_size = int(conn.recv(BUFFER_SIZE).decode())
    print('File size: ', file_size)

    # Receive file data
    file_data = b''
    while len(file_data) < file_size:
        data = conn.recv(BUFFER_SIZE)
        file_data += data

    # Save file to disk
```

```
    with open(file_name, 'wb') as f:
        f.write(file_data)

    print('File saved: ', file_name)

    # Close the connection
    conn.close()
    file_sock.close()

server_address = ('127.0.0.1', 12000)
peer_address = ('127.0.0.1', 12001)

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(server_address)
sock.setblocking(0)

print("Peer 1 ready to send and receive messages")

sequence_nb = 1
unacknowledged_messages = {}
received_messages = {}
last_retransmission_check = time.time()

#GUI

sg.theme('DarkTeal2')

layout = [[sg.Multiline(size=(100, 30), key='-OUTPUT-', autoscroll=True,
disabled=True, reroute_stdout=True, text_color='white')],
          [sg.Input(key='-INPUT-', size=(90, 2))],
          [sg.Button('Send', button_color=('white', 'darkgreen'),
border_width=5, pad=(10, 10)),
           [sg.Text("Select a file to send:")],
           [sg.Input(key="file_path"), sg.FileBrowse()],
           [sg.Button("Send File")],
           sg.Button('Receive File', button_color=('white', 'blue'),
border_width=5, pad=(10, 10)),
          sg.Button('Exit', button_color=('white', 'darkred'), border_width=5,
pad=(10, 10))]]

window = sg.Window('Peer 1', layout, finalize=True)



def retransmit_dropped_packet(sequence_nb, message, retransmissions):
    return send_message(message, sequence_nb, retransmissions + 1)

while True:
    event, values = window.read(timeout=100)
```

```
    if event in (sg.WIN_CLOSED, 'Exit'):
        sock.sendto("exit".encode(), peer_address)
        sock.close()
        break

    elif event == '-UPLOAD-':
        file_path = values['-FILE-']
        if file_path:
            send_file(file_path)

    elif event == "Send File":
        # get file path from input field
        file_path = values["file_path"]

        # check if file exists
        if not os.path.exists(file_path):
            sg.popup_error("File not found!")
        else:
            # send file
            send_file(file_path)

    elif event == 'Send':
        message = values['-INPUT-']
        crc32_value = crc32(message.encode())
        sequence_nb = send_message(message, sequence_nb, 0)
        window['-INPUT-'].update('')
        window['-OUTPUT-'].update('', append=True, text_color_for_value='white',
font=("Helvetica", 10, "bold"))

    ready_to_read, _, _ = select.select([sock], [], [], 0)

    for s in ready_to_read:
        if s == sock:
            received_msg = receive_message(sock)
            if received_msg:
                print(received_msg)

    current_time = time.time()
    if current_time - last_retransmission_check > RETRANSMISSION_TIMEOUT:
        for seq_nb, (message, timestamp, retransmissions) in
list(unacknowledged_messages.items()):
            if current_time - timestamp > RETRANSMISSION_TIMEOUT:
                if retransmissions < MAX_RETRANSMISSIONS:
                    unacknowledged_messages.pop(seq_nb, None)
                    retransmit_dropped_packet(seq_nb, message.split(': ',
2)[-1], retransmissions)
                else:
```

```
                    print(f"Message {seq_nb} dropped after reaching maximum
retransmissions")
                    unacknowledged_messages.pop(seq_nb, None)

        last_retransmission_check = current_time

    #formatting the GUI
    window['-OUTPUT-'].Widget.configure(highlightthickness=2,
highlightbackground='coral')
    window['-INPUT-'].Widget.configure(highlightthickness=2,
highlightbackground='coral')
    window['-OUTPUT-'].set_vscroll_position(1)

    #italicizing and coloring ACKs and errors
    output_widget = window['-OUTPUT-'].Widget
    output_widget.tag_configure("ACK", foreground="blue", font=("Helvetica", 10,
"italic"))
    output_widget.tag_configure("error", foreground="red", font=("Helvetica",
10, "italic"))

    lines = output_widget.get("1.0", "end-1c").split("\n")
    for i, line in enumerate(lines):
        if "Received ACK" in line:
            output_widget.tag_add("ACK", f"{i + 1}.0", f"{i + 1}.end")
        elif "CRC32 mismatch" in line:
            output_widget.tag_add("error", f"{i + 1}.0", f"{i + 1}.end")
            window.close()
```

Peer2.py

```
import sys
import socket
import select
import time
import PySimpleGUI as sg #for the GUI
import os



CRC32_TABLE = [
    0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f,
    0xe963a535, 0x9e6495a3, 0x0edb8832, 0x79dcb8a4, 0xe0d5e91e, 0x97d2d988,
    0x09b64c2b, 0x7eb17cbd, 0xe7b82d07, 0x90bf1d91, 0x1db71064, 0x6ab020f2,
    0xf3b97148, 0x84be41de, 0x1adad47d, 0x6ddde4eb, 0xf4d4b551, 0x83d385c7,
    0x136c9856, 0x646ba8c0, 0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9,
    0xfa0f3d63, 0x8d080df5, 0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172,
    0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b, 0x35b5a8fa, 0x42b2986c,
    0xdbbbc9d6, 0xacbcf940, 0x32d86ce3, 0x45df5c75, 0xdcd60dcf, 0xabd13d59,
    0x26d930ac, 0x51de003a, 0xc8d75180, 0xbfd06116, 0x21b4f4b5, 0x56b3c423,
    0xcfba9599, 0xb8bda50f, 0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924,
```

```
    0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d, 0x76dc4190, 0x01db7106,
    0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f, 0x9fbfe4a5, 0xe8b8d433,
    0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818, 0x7f6a0dbb, 0x086d3d2d,
    0x91646c97, 0xe6635c01, 0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e,
    0x6c0695ed, 0x1b01a57b, 0x8208f4c1, 0xf50fc457, 0x65b0d9c6, 0x12b7e950,
    0x8bbeb8ea, 0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65,
    0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2, 0x4adfa541, 0x3dd895d7,
    0xa4d1c46d, 0xd3d6f4fb, 0x4369e96a, 0x346ed9fc, 0xad678846, 0xda60b8d0,
    0x44042d73, 0x33031de5, 0xaa0a4c5f, 0xdd0d7cc9, 0x5005713c, 0x270241aa,
    0xbe0b1010, 0xc90c2086, 0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
    0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17, 0x2eb40d81,
    0xb7bd5c3b, 0xc0ba6cad, 0xedb88320, 0x9abfb3b6, 0x03b6e20c, 0x74b1d29a,
    0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683, 0xe3630b12, 0x94643b84,
    0x0d6d6a3e, 0x7a6a5aa8, 0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1,
    0xf00f9344, 0x8708a3d2, 0x1e01f268, 0x6906c2fe, 0xf762575d, 0x806567cb,
    0x196c3671, 0x6e6b06e7, 0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc,
    0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5, 0xd6d6a3e8, 0xa1d1937e,
    0x38d8c2c4, 0x4fdff252, 0xd1bb67f1, 0xa6bc5767, 0x3fb506dd, 0x48b2364b,
    0xd80d2bda, 0xaf0a1b4c, 0x36034af6, 0x41047a60, 0xdf60efc3, 0xa867df55,
    0x316e8eef, 0x4669be79, 0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236,
    0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f, 0xc5ba3bbe, 0xb2bd0b28,
    0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b, 0x5bdeae1d,
    0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a, 0x9c0906a9, 0xeb0e363f,
    0x72076785, 0x05005713, 0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38,
    0x92d28e9b, 0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21, 0x86d3d2d4, 0xf1d4e242,
    0x68ddb3f8, 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777,
    0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c, 0x8f659eff, 0xf862ae69,
    0x616bffd3, 0x166ccf45, 0xa00ae278, 0xd70dd2ee, 0x4e048354, 0x3903b3c2,
    0xa7672661, 0xd06016f7, 0x4969474d, 0x3e6e77db, 0xaed16a4a, 0xd9d65adc,
    0x40df0b66, 0x37d83bf0, 0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f, 0x30b5ffe9,
    0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605, 0xcdd70693,
    0x54de5729, 0x23d967bf, 0xb3667a2e, 0xc4614ab8, 0x5d681b02, 0x2a6f2b94,
    0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d]

RETRANSMISSION_TIMEOUT = 2
MAX_RETRANSMISSIONS = 6
RETRANSMISSION_LIMIT = 60  #1 min
BUFFER_SIZE = 10000
HOST = '127.0.0.1'
FILE_PORT = 1234

def crc32(data):
    crc = 0xffffffff
    for byte in data:
        crc = (crc >> 8) ^ CRC32_TABLE[(crc ^ byte) & 0xff]
    return crc ^ 0xffffffff

def send_message(message, sequence_nb, retransmissions=0):
    if message.lower() == 'exit':
```

```
        sock.sendto(message.encode(), peer_address)
        sock.close()
        sys.exit()

    crc32_value = crc32(message.encode())
    message = f"{sequence_nb}: {crc32_value}: {message}"
    sock.sendto(message.encode(), peer_address)
    unacknowledged_messages[sequence_nb] = (message, time.time(),
retransmissions)

    if retransmissions > 0:
        print(f"Retransmitting message {sequence_nb} (retransmission
{retransmissions}): {message}")

    return sequence_nb + 1


def receive_message(sock):
    received_message, _ = sock.recvfrom(2048)
    decoded_message = received_message.decode()

    if decoded_message.lower() == 'exit':
        print("Peer 2 has left by typing or pressing 'exit'")
        sock.close()
        sys.exit()
    elif decoded_message.startswith("ACK:"):
        ack_sequence_nb = int(decoded_message.split(" ")[1])
        unacknowledged_messages.pop(ack_sequence_nb, None)
        print(f"Received ACK for message {ack_sequence_nb}")
    else:
        message_sequence_nb, crc32_received, message_text =
decoded_message.split(": ", 2)
        message_sequence_nb = int(message_sequence_nb)
        crc32_received = int(crc32_received)

        if crc32(message_text.encode()) == crc32_received:
            if message_sequence_nb not in received_messages:
                received_messages[message_sequence_nb] = message_text
                print(f"Received message from Peer 1: {message_text}")

            ack_message = f"ACK: {message_sequence_nb}"
            sock.sendto(ack_message.encode(), peer_address)

        else:
            print("CRC32 mismatch, discarding message")

def send_file(file_path):
    # Send a file to the other peer
    with open(file_path, 'rb') as file:
```

```
        data = file.read()

    # Send the file in packets
    packet_size = 1024
    packets = [data[i:i+packet_size] for i in range(0, len(data), packet_size)]
    num_packets = len(packets)
    send_message(f"START {num_packets} {file_path}", 0)

    for i, packet in enumerate(packets, start=1):
        sequence_nb = send_message(packet, i)
        time.sleep(0.001) # wait a little between packets to not overwhelm the
network

    send_message("END", num_packets+1)

def receive_file():
    # Receive a file from the other peer
    file_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    file_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    file_sock.bind((HOST, FILE_PORT))
    file_sock.listen(1)

    print('Waiting for incoming file...')
    conn, addr = file_sock.accept()
    print('File incoming from: ', addr)

    # Receive file name
    file_name = conn.recv(BUFFER_SIZE).decode()
    print('File name: ', file_name)

    # Receive file size
    file_size = int(conn.recv(BUFFER_SIZE).decode())
    print('File size: ', file_size)

    # Receive file data
    file_data = b''
    while len(file_data) < file_size:
        data = conn.recv(BUFFER_SIZE)
        file_data += data

    # Save file to disk
    with open(file_name, 'wb') as f:
        f.write(file_data)

    print('File saved: ', file_name)

    # Close the connection
    conn.close()
    file_sock.close()
```

```
server_address = ('127.0.0.1', 12001)
peer_address = ('127.0.0.1', 12000)

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(server_address)
sock.setblocking(0)

print("Peer 2 ready to send and receive messages")

sequence_nb = 1
unacknowledged_messages = {}
received_messages = {}
last_retransmission_check = time.time()

#GUI

sg.theme('DarkTeal3')

layout = [[sg.Multiline(size=(100, 30), key='-OUTPUT-', autoscroll=True,
disabled=True, reroute_stdout=True, text_color='white')],
         [sg.Input(key='-INPUT-', size=(90, 2))],
         [sg.Button('Send', button_color=('white', 'darkgreen'),
border_width=5, pad=(10, 10)),
          [sg.Text("Select a file to send:")],
          [sg.Input(key="file_path"), sg.FileBrowse()],
          [sg.Button("Send File")],
          sg.Button('Receive File', button_color=('white', 'blue'),
border_width=5, pad=(10, 10)),
          sg.Button('Exit', button_color=('white', 'darkred'), border_width=5,
pad=(10, 10))]]

window = sg.Window('Peer 2', layout, finalize=True)



def retransmit_dropped_packet(sequence_nb, message, retransmissions):
   return send_message(message, sequence_nb, retransmissions + 1)

while True:
   event, values = window.read(timeout=100)

   if event in (sg.WIN_CLOSED, 'Exit'):
       sock.sendto("exit".encode(), peer_address)
       sock.close()
       break

   elif event == '-UPLOAD-':
       file_path = values['-FILE-']
```

```
        if file_path:
            send_file(file_path)

    elif event == "Send File":
        #get file path
        file_path = values["file_path"]

        #if file exists
        if not os.path.exists(file_path):
            sg.popup_error("File not found!")
        else:
            #send file
            send_file(file_path)

    elif event == 'Send':
        message = values['-INPUT-']
        crc32_value = crc32(message.encode())
        sequence_nb = send_message(message, sequence_nb, 0)
        window['-INPUT-'].update('')
        window['-OUTPUT-'].update('', append=True, text_color_for_value='white',
font=("Helvetica", 10, "bold"))

    ready_to_read, _, _ = select.select([sock], [], [], 0)

    for s in ready_to_read:
        if s == sock:
            received_msg = receive_message(sock)
            if received_msg:
                print(received_msg)

    current_time = time.time()
    if current_time - last_retransmission_check > RETRANSMISSION_TIMEOUT:
        for seq_nb, (message, timestamp, retransmissions) in
list(unacknowledged_messages.items()):
            if current_time - timestamp > RETRANSMISSION_TIMEOUT:
                if retransmissions < MAX_RETRANSMISSIONS:
                    unacknowledged_messages.pop(seq_nb, None)
                    retransmit_dropped_packet(seq_nb, message.split(': ',
2)[-1], retransmissions)
                else:
                    print(f"Message {seq_nb} dropped after reaching maximum
retransmissions")
                    unacknowledged_messages.pop(seq_nb, None)

        last_retransmission_check = current_time

    #formatting the GUI
    window['-OUTPUT-'].Widget.configure(highlightthickness=2,
highlightbackground='coral')
```

```
   window['-INPUT-'].Widget.configure(highlightthickness=2,
highlightbackground='coral')
   window['-OUTPUT-'].set_vscroll_position(1)

   #italicizing and coloring ACKs and errors
   output_widget = window['-OUTPUT-'].Widget
   output_widget.tag_configure("ACK", foreground="blue", font=("Helvetica", 10,
"italic"))
   output_widget.tag_configure("error", foreground="red", font=("Helvetica",
10, "italic"))

   lines = output_widget.get("1.0", "end-1c").split("\n")
   for i, line in enumerate(lines):
       if "Received ACK" in line:
           output_widget.tag_add("ACK", f"{i + 1}.0", f"{i + 1}.end")
       elif "CRC32 mismatch" in line:
           output_widget.tag_add("error", f"{i + 1}.0", f"{i + 1}.end")
           window.close()
```

Here, we added a file transfer. A file is split into 1024 bytes and each sent with a seq number, and when it is sent it waits for an ack from the receiver, if its not received there is retransmission and it is ignored

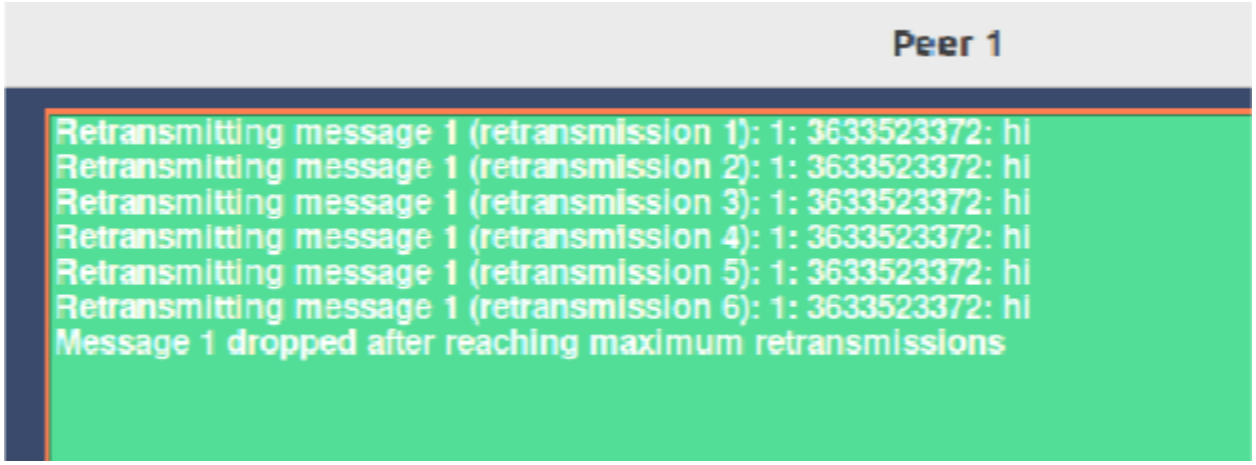
Bonus

We used PySimpleGUI. For it to work we had to install this module and Tkinter. PySimpleGui is based on Tkinter and we found it much easier to work with, so we decided to use it instead of TKinter even though TKinter is arguably one of the best Python GUIs to use.

Unreliable Network Conditions

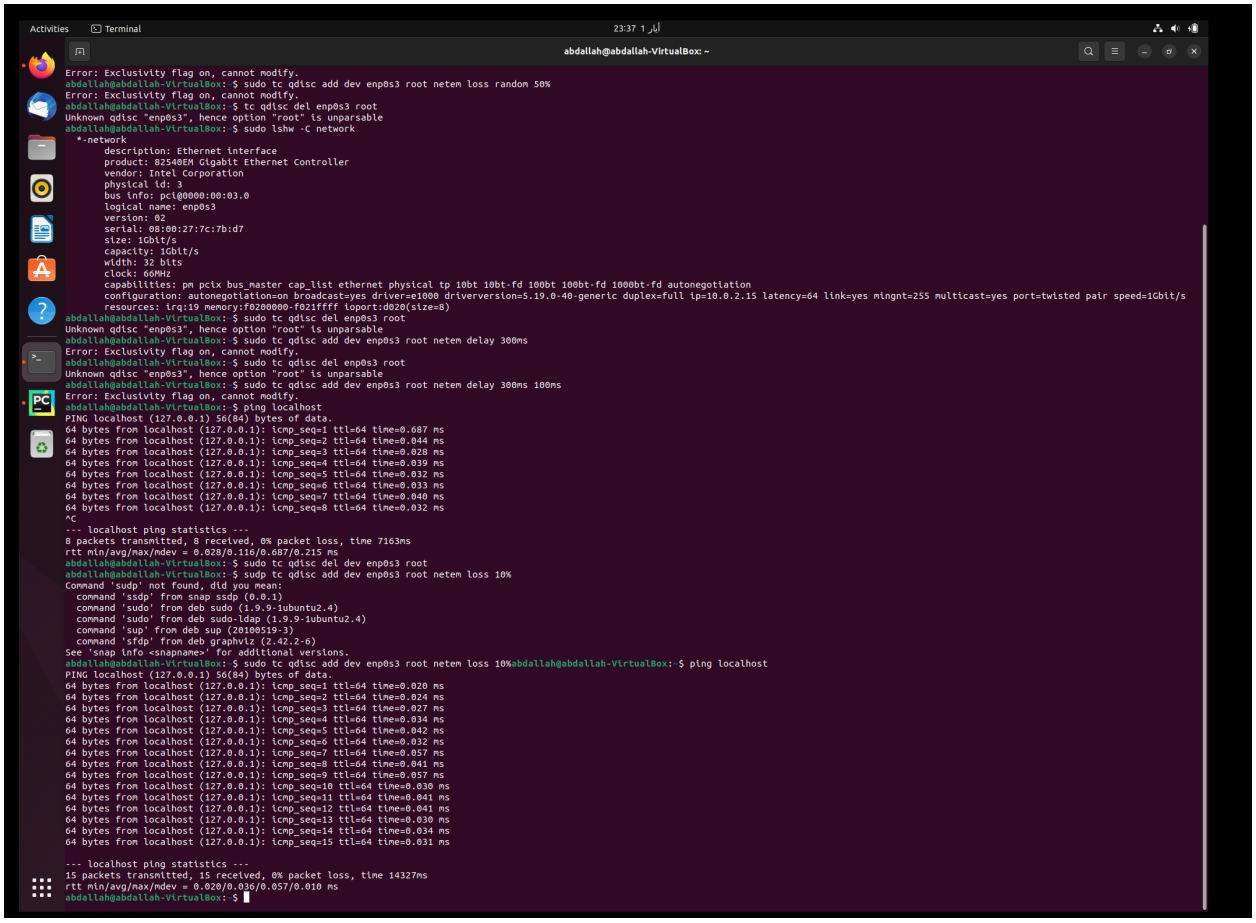
For this part, we have to use netem to induce unreliable network conditions. We will simulate packet loss, delay, . we will also show other errors.

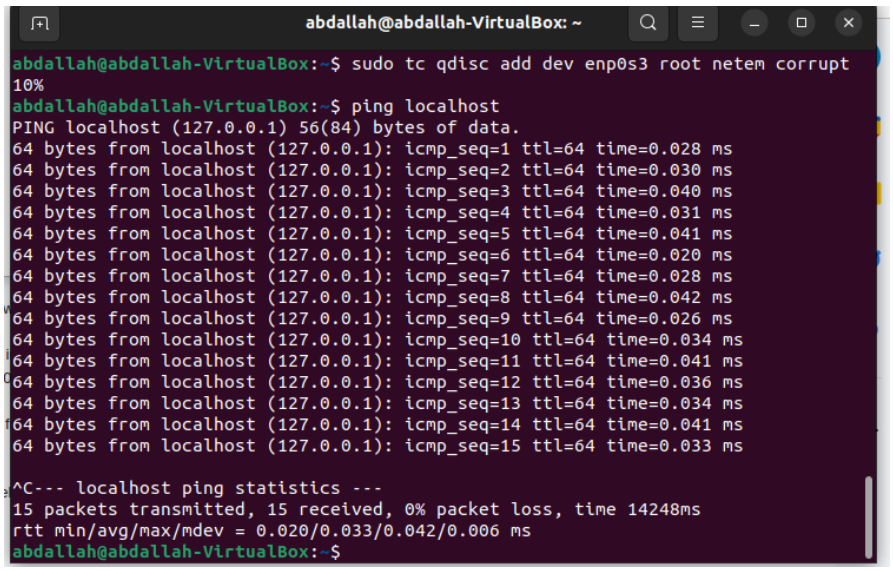
If only one peer is open and tries to send a message, it will keep retransmitting and then drop it.

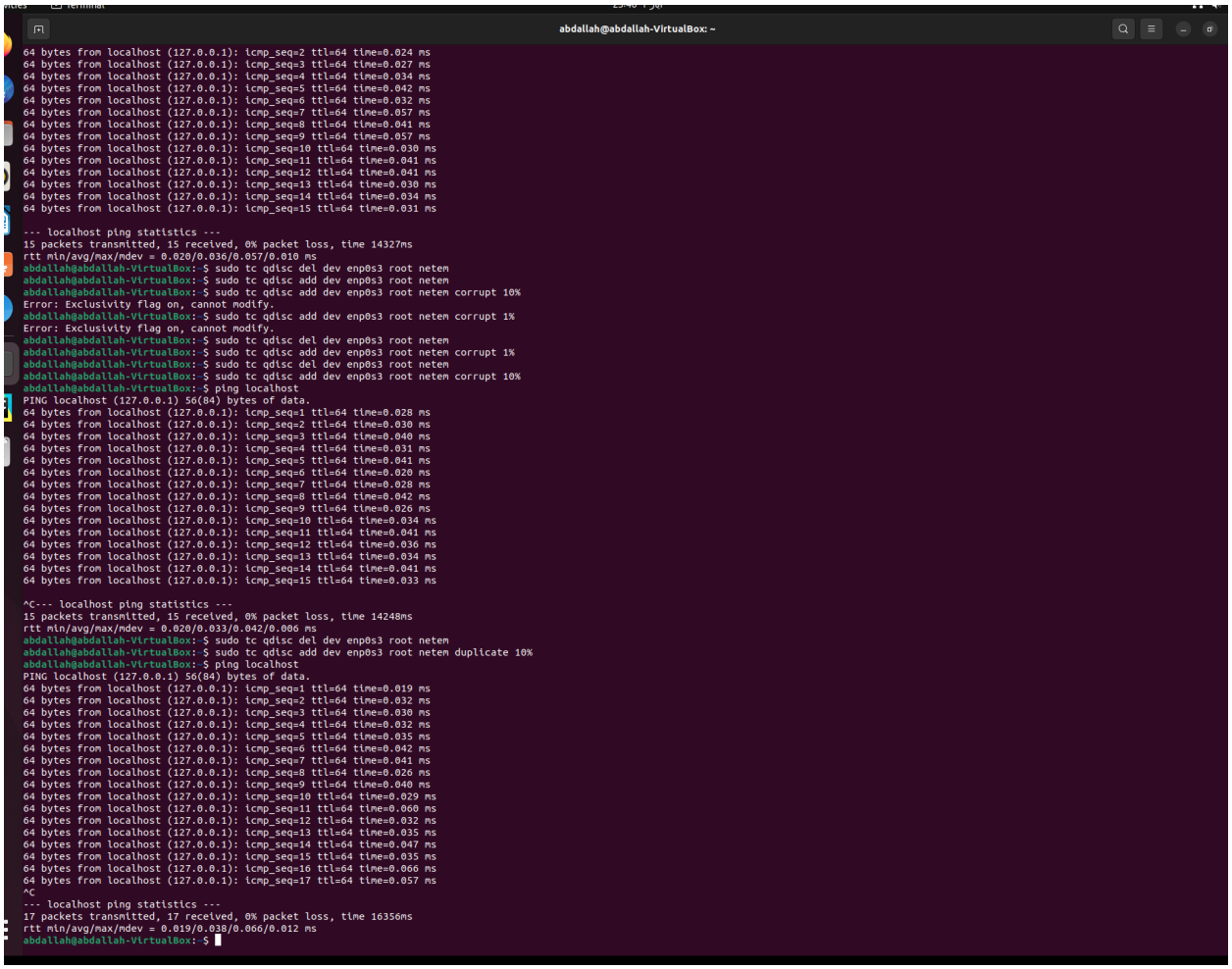


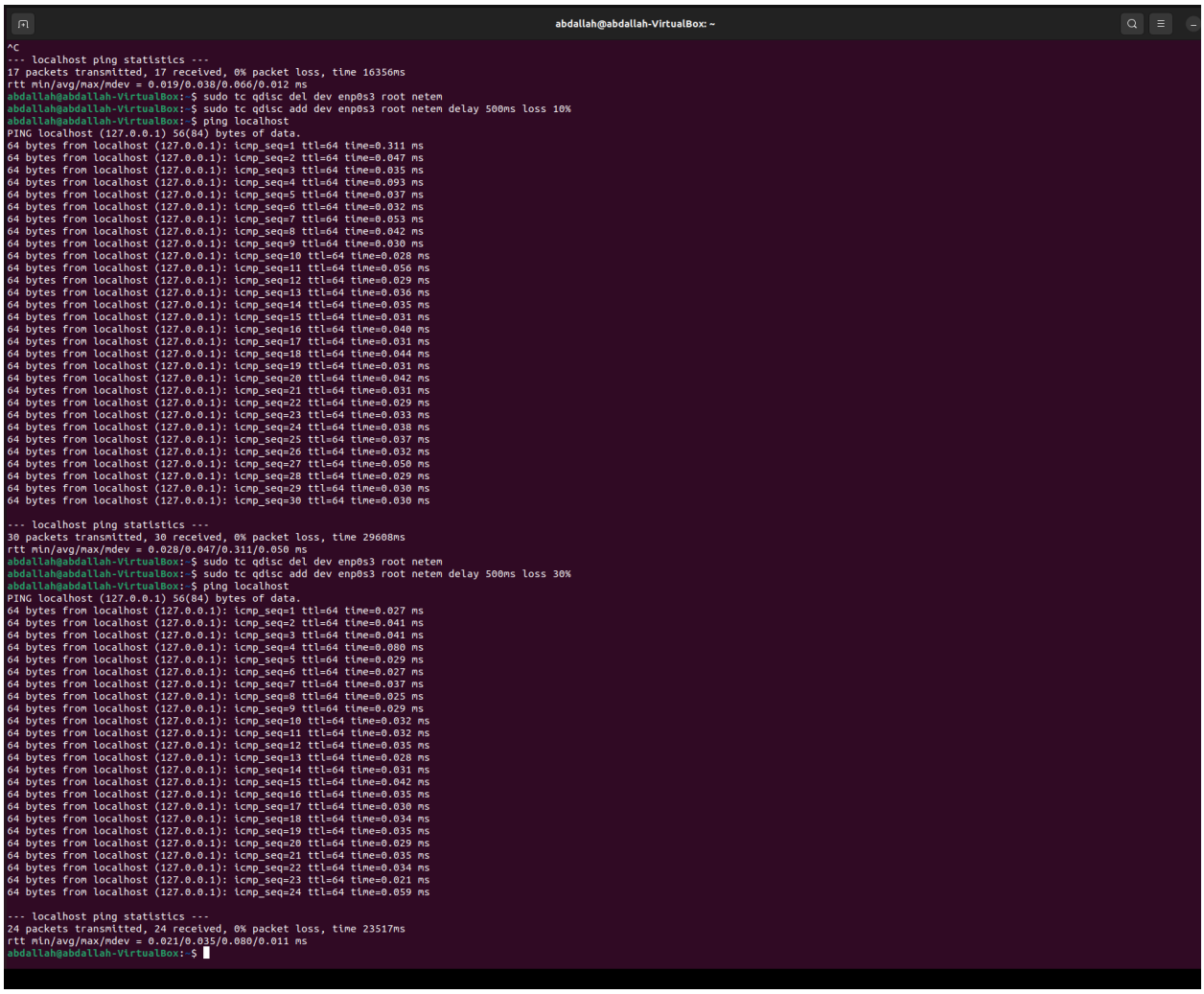
This is peer 1, we made it send a message to peer 2 but we did not open peer 2. It will keep retransmitting until it exhausts its maximum retransmission tries which is 6, where it will drop the packet.

Second question :
In the screenshots below we focused on trying delay with netem for unreliable connection with packet loss packet duplicate packet timeout too


Activities
Terminal
23:37 1 أيار
abdallah@abdallah-VirtualBox: ~
Error: Exclusivity flag on, cannot modify.
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc add dev enp0s3 root netem loss random 50%
Error: Exclusivity flag on, cannot modify.
abdallah@abdallah-VirtualBox:~$ tc qdisc del enp0s3 root
Unknown qdisc "enp0s3", hence option "root" is unparsable
abdallah@abdallah-VirtualBox:~$ sudo lshw -C network
*-network
description: Ethernet interface
product: 82540EM Gigabit Ethernet Controller
vendor: Intel Corporation
physical id: 3
bus info: pci@0000:00:03.0
logical name: enp0s3
version: 02
serial: 08:00:27:7c:7b:d7
size: 1Gbit/s
capacity: 1Gbit/s
width: 32 bits
clock: 66MHz
capabilities: pm pcix bus_master cap_list ethernet physical tp 10bt 10bt-fd 100bt 100bt-fd 1000bt-fd autonegotiation
configuration: autonegotiation=on broadcast=yes driver=e1000 driverversion=5.19.0-40-generic duplex=full ip=10.0.2.15 latency=64 link=yes mingnt=255 multicast=yes port=twisted pair speed=1Gbit/s
resources: irq:19 memory:f0200000-f021ffff ioport:d020(size=8)
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc del enp0s3 root
Unknown qdisc "enp0s3", hence option "root" is unparsable
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc add dev enp0s3 root netem delay 300ms
Error: Exclusivity flag on, cannot modify.
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc del enp0s3 root
Unknown qdisc "enp0s3", hence option "root" is unparsable
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc add dev enp0s3 root netem delay 300ms 100ms
Error: Exclusivity flag on, cannot modify.
abdallah@abdallah-VirtualBox:~$ ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.687 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.044 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.028 ms
64 bytes from localhost (127.0.0.1): icmp_seq=4 ttl=64 time=0.039 ms
64 bytes from localhost (127.0.0.1): icmp_seq=5 ttl=64 time=0.032 ms
64 bytes from localhost (127.0.0.1): icmp_seq=6 ttl=64 time=0.033 ms
64 bytes from localhost (127.0.0.1): icmp_seq=7 ttl=64 time=0.040 ms
64 bytes from localhost (127.0.0.1): icmp_seq=8 ttl=64 time=0.032 ms
^C
--- localhost ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7163ms
rtt min/avg/max/mdev = 0.028/0.116/0.687/0.215 ms
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc del dev enp0s3 root
abdallah@abdallah-VirtualBox:~$ sudp tc qdisc add dev enp0s3 root netem loss 10%
Command 'sudp' not found, did you mean:
command 'ssdp' from snap ssdp (0.0.1)
command 'sudo' from deb sudo (1.9.9-1ubuntu2.4)
command 'sudo' from deb sudo-ldap (1.9.9-1ubuntu2.4)
command 'sup' from deb sup (20100519-3)
command 'sfdp' from deb graphviz (2.42.2-6)
See 'snap info <snapname>' for additional versions.
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc add dev enp0s3 root netem loss 10%abdallah@abdallah-VirtualBox:~$ ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.020 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.024 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.027 ms
64 bytes from localhost (127.0.0.1): icmp_seq=4 ttl=64 time=0.034 ms
64 bytes from localhost (127.0.0.1): icmp_seq=5 ttl=64 time=0.042 ms
64 bytes from localhost (127.0.0.1): icmp_seq=6 ttl=64 time=0.032 ms
64 bytes from localhost (127.0.0.1): icmp_seq=7 ttl=64 time=0.057 ms
64 bytes from localhost (127.0.0.1): icmp_seq=8 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=9 ttl=64 time=0.057 ms
64 bytes from localhost (127.0.0.1): icmp_seq=10 ttl=64 time=0.030 ms
64 bytes from localhost (127.0.0.1): icmp_seq=11 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=12 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=13 ttl=64 time=0.030 ms
64 bytes from localhost (127.0.0.1): icmp_seq=14 ttl=64 time=0.034 ms
64 bytes from localhost (127.0.0.1): icmp_seq=15 ttl=64 time=0.031 ms
--- localhost ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14327ms
rtt min/avg/max/mdev = 0.020/0.036/0.057/0.010 ms
abdallah@abdallah-VirtualBox:~$


abdallah@abdallah-VirtualBox: ~
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc add dev enp0s3 root netem corrupt 10%
abdallah@abdallah-VirtualBox:~$ ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.028 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.030 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.040 ms
64 bytes from localhost (127.0.0.1): icmp_seq=4 ttl=64 time=0.031 ms
64 bytes from localhost (127.0.0.1): icmp_seq=5 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=6 ttl=64 time=0.020 ms
64 bytes from localhost (127.0.0.1): icmp_seq=7 ttl=64 time=0.028 ms
64 bytes from localhost (127.0.0.1): icmp_seq=8 ttl=64 time=0.042 ms
64 bytes from localhost (127.0.0.1): icmp_seq=9 ttl=64 time=0.026 ms
64 bytes from localhost (127.0.0.1): icmp_seq=10 ttl=64 time=0.034 ms
64 bytes from localhost (127.0.0.1): icmp_seq=11 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=12 ttl=64 time=0.036 ms
64 bytes from localhost (127.0.0.1): icmp_seq=13 ttl=64 time=0.034 ms
64 bytes from localhost (127.0.0.1): icmp_seq=14 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=15 ttl=64 time=0.033 ms
^C--- localhost ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14248ms
rtt min/avg/max/mdev = 0.020/0.033/0.042/0.006 ms
abdallah@abdallah-VirtualBox:~$


abdallah@abdallah-VirtualBox: ~
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.024 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.027 ms
64 bytes from localhost (127.0.0.1): icmp_seq=4 ttl=64 time=0.034 ms
64 bytes from localhost (127.0.0.1): icmp_seq=5 ttl=64 time=0.042 ms
64 bytes from localhost (127.0.0.1): icmp_seq=6 ttl=64 time=0.032 ms
64 bytes from localhost (127.0.0.1): icmp_seq=7 ttl=64 time=0.057 ms
64 bytes from localhost (127.0.0.1): icmp_seq=8 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=9 ttl=64 time=0.057 ms
64 bytes from localhost (127.0.0.1): icmp_seq=10 ttl=64 time=0.030 ms
64 bytes from localhost (127.0.0.1): icmp_seq=11 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=12 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=13 ttl=64 time=0.030 ms
64 bytes from localhost (127.0.0.1): icmp_seq=14 ttl=64 time=0.034 ms
64 bytes from localhost (127.0.0.1): icmp_seq=15 ttl=64 time=0.031 ms
--- localhost ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14327ms
rtt min/avg/max/mdev = 0.020/0.036/0.057/0.010 ms
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc del dev enp0s3 root netem
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc add dev enp0s3 root netem
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc add dev enp0s3 root netem corrupt 10%
Error: Exclusivity flag on, cannot modify.
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc add dev enp0s3 root netem corrupt 1%
Error: Exclusivity flag on, cannot modify.
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc del dev enp0s3 root netem
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc add dev enp0s3 root netem corrupt 1%
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc del dev enp0s3 root netem
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc add dev enp0s3 root netem corrupt 10%
abdallah@abdallah-VirtualBox:~$ ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.028 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.030 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.040 ms
64 bytes from localhost (127.0.0.1): icmp_seq=4 ttl=64 time=0.031 ms
64 bytes from localhost (127.0.0.1): icmp_seq=5 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=6 ttl=64 time=0.020 ms
64 bytes from localhost (127.0.0.1): icmp_seq=7 ttl=64 time=0.028 ms
64 bytes from localhost (127.0.0.1): icmp_seq=8 ttl=64 time=0.042 ms
64 bytes from localhost (127.0.0.1): icmp_seq=9 ttl=64 time=0.026 ms
64 bytes from localhost (127.0.0.1): icmp_seq=10 ttl=64 time=0.034 ms
64 bytes from localhost (127.0.0.1): icmp_seq=11 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=12 ttl=64 time=0.036 ms
64 bytes from localhost (127.0.0.1): icmp_seq=13 ttl=64 time=0.034 ms
64 bytes from localhost (127.0.0.1): icmp_seq=14 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=15 ttl=64 time=0.033 ms
^C--- localhost ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14248ms
rtt min/avg/max/mdev = 0.020/0.033/0.042/0.006 ms
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc del dev enp0s3 root netem
abdallah@abdallah-VirtualBox:~$ sudo tc qdisc add dev enp0s3 root netem duplicate 10%
abdallah@abdallah-VirtualBox:~$ ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.019 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.032 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.030 ms
64 bytes from localhost (127.0.0.1): icmp_seq=4 ttl=64 time=0.032 ms
64 bytes from localhost (127.0.0.1): icmp_seq=5 ttl=64 time=0.035 ms
64 bytes from localhost (127.0.0.1): icmp_seq=6 ttl=64 time=0.042 ms
64 bytes from localhost (127.0.0.1): icmp_seq=7 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=8 ttl=64 time=0.026 ms
64 bytes from localhost (127.0.0.1): icmp_seq=9 ttl=64 time=0.040 ms
64 bytes from localhost (127.0.0.1): icmp_seq=10 ttl=64 time=0.029 ms
64 bytes from localhost (127.0.0.1): icmp_seq=11 ttl=64 time=0.060 ms
64 bytes from localhost (127.0.0.1): icmp_seq=12 ttl=64 time=0.032 ms
64 bytes from localhost (127.0.0.1): icmp_seq=13 ttl=64 time=0.035 ms
64 bytes from localhost (127.0.0.1): icmp_seq=14 ttl=64 time=0.047 ms
64 bytes from localhost (127.0.0.1): icmp_seq=15 ttl=64 time=0.035 ms
64 bytes from localhost (127.0.0.1): icmp_seq=16 ttl=64 time=0.066 ms
64 bytes from localhost (127.0.0.1): icmp_seq=17 ttl=64 time=0.057 ms
^C
--- localhost ping statistics ---
17 packets transmitted, 17 received, 0% packet loss, time 16356ms
rtt min/avg/max/mdev = 0.019/0.038/0.066/0.012 ms
abdallah@abdallah-VirtualBox:~$



3. References

PYMOTW. select – Wait for I/O Efficiently - Python Module of the Week. (n.d.). Retrieved April 2023, from http://pymotw.com/2/select/

Socket - low-level networking interface. Python documentation. (n.d.). Retrieved April 2023, from https://docs.python.org/3/library/socket.html

Sys - system-specific parameters and functions. Python documentation. (n.d.). Retrieved April 2023, from https://docs.python.org/3/library/sys.html

Kurose, J. F., & Ross, K. W. (2020). *Computer networking: A top-down approach*. Pearson Education Limited.

Darwin, N. (2021, December 28). How to put a background image in Tkinter Python gui: Put background image in Python/Pycharm Gui. YouTube. Retrieved May 1, 2023, from https://youtu.be/fsL8oZXiJmY

Multiple file transfer using TCP socket in python: Folder transfer using TCP socket in python. YouTube. (2022, October 21). Retrieved May 1, 2023, from https://youtu.be/Ah3iz3pKljU

Ivanov, A. (2020, August 8). P2P twisted chat: Python. YouTube. Retrieved May 1, 2023, from https://youtu.be/1Fay1pjttLg

Bhavsar, M. (2020, December 29). File transfer using socket programming (python)(source code available). YouTube. Retrieved May 1, 2023, from https://youtu.be/SZyd7xGTBkw

Neupane, A. (2021, June 14). How to upload files with flask using python. YouTube. Retrieved May 1, 2023, from https://youtu.be/GeiUTkSAJPs

CRC. PyPI. (n.d.). Retrieved May 1, 2023, from https://pypi.org/project/crc/

Brown, G. (1986). MIT - Massachusetts Institute of Technology. crc-32.c -MIT. Retrieved May 1, 2023, from https://web.mit.edu/freebsd/head/sys/libkern/crc32.c

Python GUIs for humans. PySimpleGUI. (n.d.). Retrieved May 1, 2023, from https://www.pysimplegui.org/en/latest/

https://medium.com/docler-engineering/network-issues-simulation-how-to-test-against-bad-network-conditions-b28f651d8a96