STOP AND WAIT ARQ PROTOCOL USING SOCKETS (FLOW CONTROL)

CSE1004(NETWORK AND COMMUNICATION)LAB:L53-L54

FEBURARY 09, 2022

ANIRUDH VADERA
20BCE2940

QUESTION:

Write a python program to implement flow control mechanism which continues to send a number of frames specified by a window size even without receiving an acknowledgement (ACK) packet from the receiver. The receiver process keeps track of the sequence number of the next frame it expects to receive, and sends that number with every ACK it sends.

DESCRIPTION:

Stop and Wait Protocol:

It is the simplest flow control mechanism

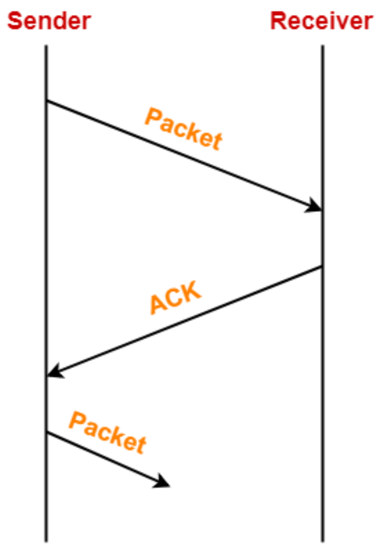
It works under the following assumptions-

- Communication channel is perfect.
- No error occurs during transmission.

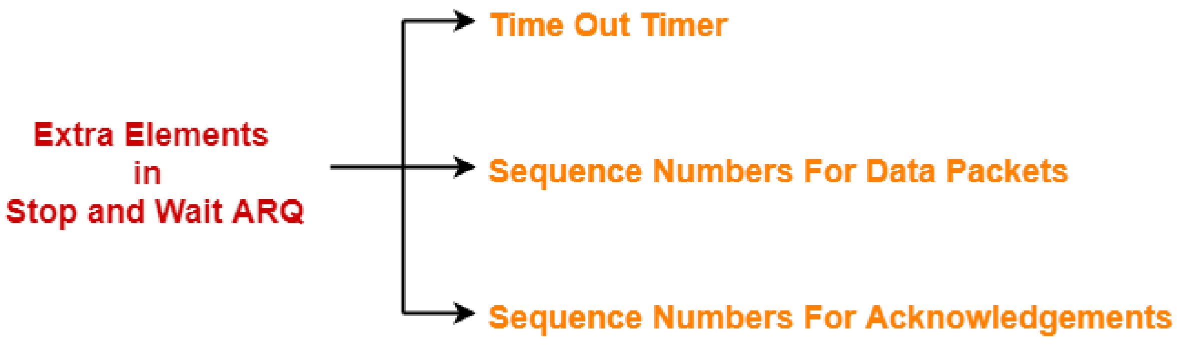
Working:

The working of a stop and wait protocol may be explained as-

- Sender sends a data packet to the receiver.
- Sender stops and waits for the acknowledgement for the sent packet from the receiver.
- Receiver receives and processes the data packet.
- Receiver sends an acknowledgement to the sender.
- After receiving the acknowledgement, sender sends the next data packet to the receiver.



Stop and Wait Protocol

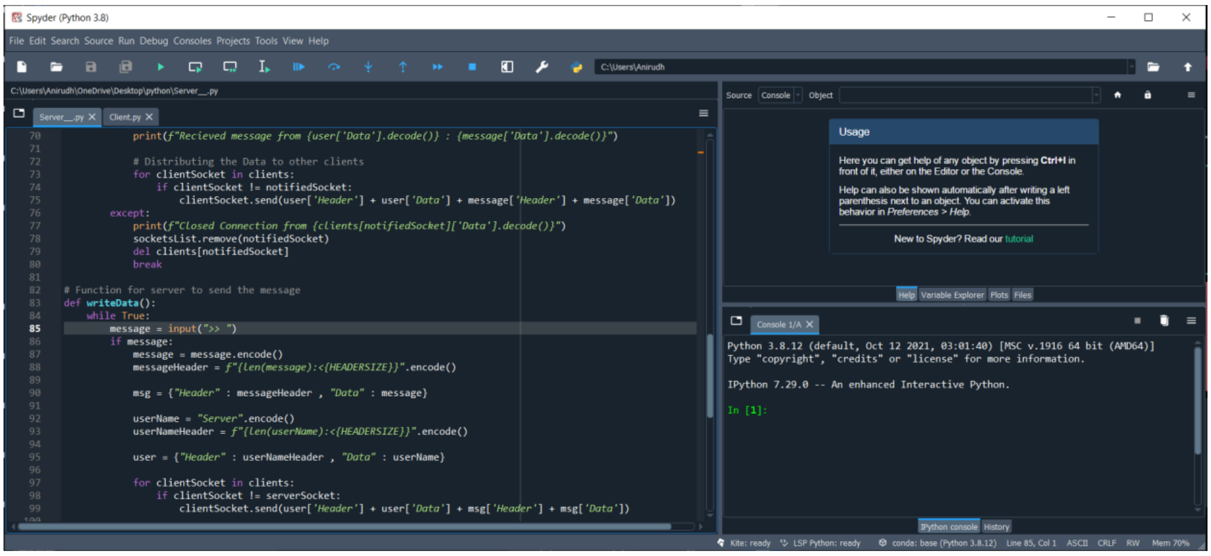

Extra Elements in Stop and Wait ARQ
Time Out Timer
Sequence Numbers For Data Packets
Sequence Numbers For Acknowledgements

PROCEDURE:

- Sender delivers a sequence number 0 data frame or packet
- After receiving the data frame, Receiver sends a sequence number 1 acknowledgment (the sequence number of the next expected data frame or packet) Because there is just a one-bit sequence number, both the transmitter and the receiver only have a buffer for one frame or packet.
- We will keep a ack_flag variable which will keep count of the current acknowledgement to be sent to the sender
- We also keep a current_frame variable in order to keep track of the current frame send to the receiver
- Initially we send the total number of frames to be received by the receiver
- We update the current_frame and ack_flag as required.

➔ **First Open your python ide**

➔ **I will be using anaconda distribution and a spyder IDE**



```
            print(f"Recieved message from {user['Data'].decode()} : {message['Data'].decode()}")

            # Distributing the Data to other clients
            for clientSocket in clients:
                if clientSocket != notifiedSocket:
                    clientSocket.send(user['Header'] + user['Data'] + message['Header'] + message['Data'])
        except:
            print(f"Closed Connection from {clients[notifiedSocket]['Data'].decode()}")
            socketsList.remove(notifiedSocket)
            del clients[notifiedSocket]
            break

# Function for server to send the message
def writeData():
    while True:
        message = input(">> ")
        if message:
            message = message.encode()
            messageHeader = f"{len(message):<{HEADERSIZE}}".encode()

            msg = {"Header" : messageHeader , "Data" : message}

            userName = "Server".encode()
            userNameHeader = f"{len(userName):<{HEADERSIZE}}".encode()

            user = {"Header" : userNameHeader , "Data" : userName}

            for clientSocket in clients:
                if clientSocket != serverSocket:
                    clientSocket.send(user['Header'] + user['Data'] + msg['Header'] + msg['Data'])
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
Python 3.8.12 (default, Oct 12 2021, 03:01:40) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.29.0 -- An enhanced Interactive Python.

In [1]:
```

Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Anirudh
C:\Users\Anirudh\OneDrive\Desktop\python\Server__.py
Server__.py
Client.py
Source Console Object
Help Variable Explorer Plots Files
Console 1/A
IPython console History

➔ **We will be using 2 files for our purpose**

➔ **A server file**

➔ **A client file**

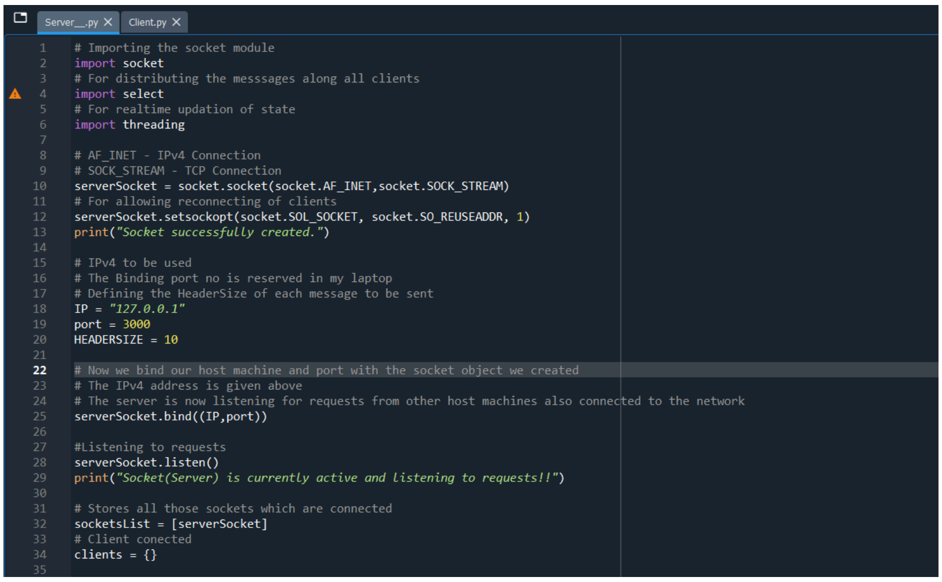
There are some common steps to be followed explained below

➔ **A detailed explanation along with the code is given further below**

Server.py: (Reciever)

1. **Import the necessary files.**
2. **Using a IPv4 connection and a TCP connection initiate the server side socket using socket.socket(socket.AF_INET,socket.SOCK_STREAM)**
3. **Bind the server using socket.bind(IP,port) method providing the IP and the port.**
4. **We now define the socketsList which stores all the sockets currently in action and make a client Dictionary which stores information about the clients.**
5. **We then define a function for reading messages using socket.recv() method**
6. **We then make a function for writing the messages to the client using the socket.send()**
7. **We then implement the stop and wait arq protocol using flag variables**

A code snippet for server.py:



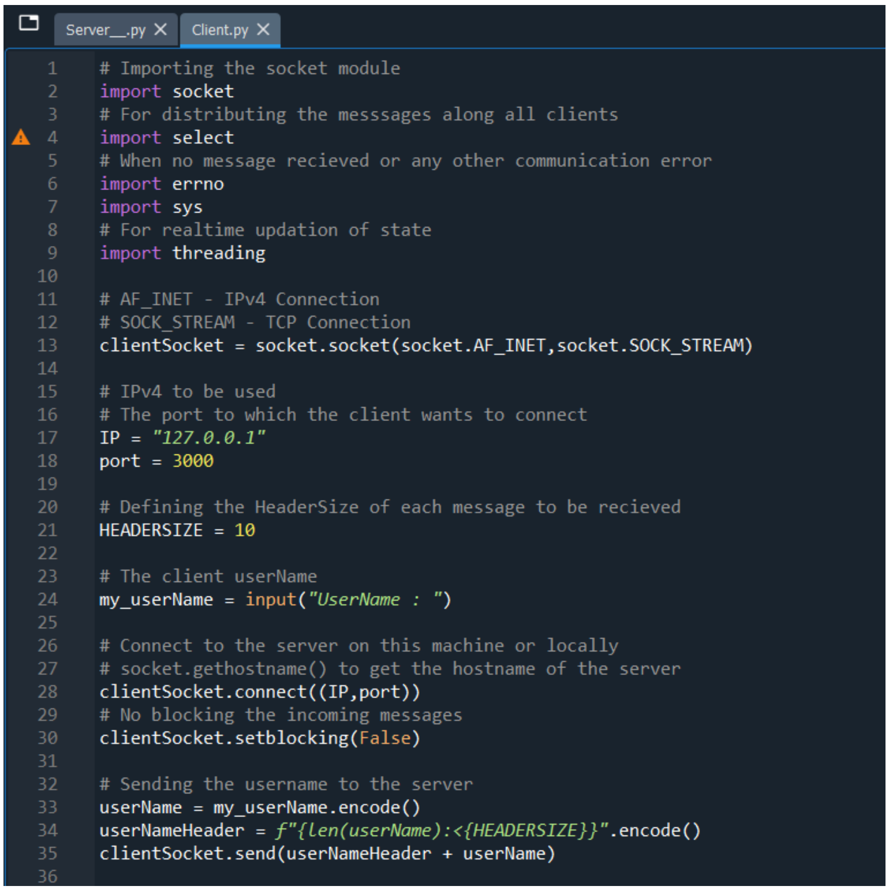
```python
# Importing the socket module
import socket
# For distributing the messsages along all clients
import select
# For realtime updation of state
import threading

# AF_INET - IPv4 Connection
# SOCK_STREAM - TCP Connection
serverSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
# For allowing reconnecting of clients
serverSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
print("Socket successfully created.")

# IPv4 to be used
# The Binding port no is reserved in my laptop
# Defining the HeaderSize of each message to be sent
IP = "127.0.0.1"
port = 3000
HEADERSIZE = 10

# Now we bind our host machine and port with the socket object we created
# The IPv4 address is given above
# The server is now listening for requests from other host machines also connected to the network
serverSocket.bind((IP,port))

#Listening to requests
serverSocket.listen()
print("Socket(Server) is currently active and listening to requests!!")

# Stores all those sockets which are connected
socketsList = [serverSocket]
# Client conected
clients = {}

```

Client.py: (Sender)

1. **Import the necessary files.**
2. **Using a IPv4 connection and a TCP connection initiate the server side socket using socket.socket(socket.AF_INET,socket.SOCK_STREAM)**
3. **Connect to the server using socket.connect(IP) function by providing the appropriate IP address**
4. **Select a username and send it to the server.**
5. **We then define a function for reading messages using socket.recv() method**
6. **We try to catch as many errors as possible in it.**

7. **We then make a function for writing the messages to the client using the socket.send()**
8. **We then implement stop and wait protocol using flag variables**

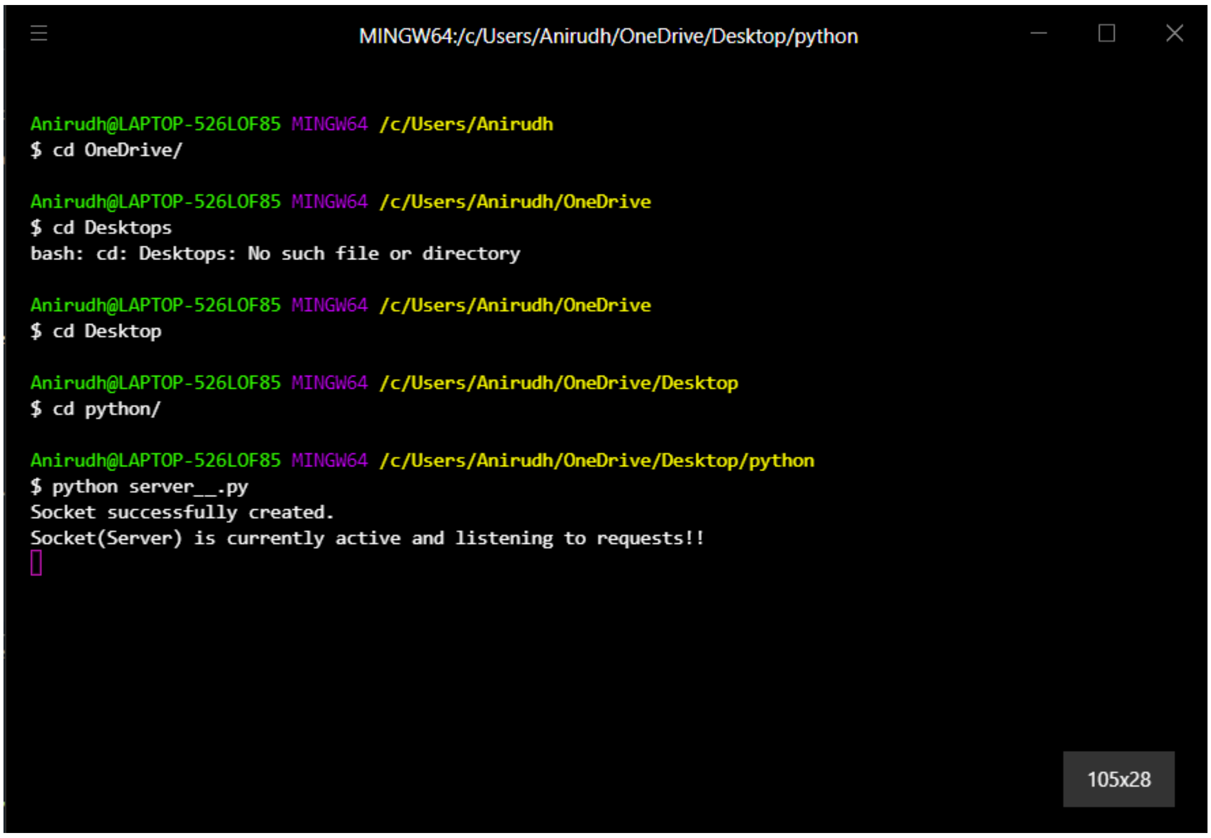
A code snippet for client.py:



```python
# Importing the socket module
import socket
# For distributing the messsages along all clients
import select
# When no message recieved or any other communication error
import errno
import sys
# For realtime updation of state
import threading

# AF_INET - IPv4 Connection
# SOCK_STREAM - TCP Connection
clientSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

# IPv4 to be used
# The port to which the client wants to connect
IP = "127.0.0.1"
port = 3000

# Defining the HeaderSize of each message to be recieved
HEADERSIZE = 10

# The client userName
my_userName = input("UserName : ")

# Connect to the server on this machine or locally
# socket.gethostname() to get the hostname of the server
clientSocket.connect((IP,port))
# No blocking the incoming messages
clientSocket.setblocking(False)

# Sending the username to the server
userName = my_userName.encode()
userNameHeader = f"{len(userName):<{HEADERSIZE}}".encode()
clientSocket.send(userNameHeader + userName)

```

In order to run our Application, we follow the following steps:

- ➔ **Open the Hyper terminal or Command Prompt**
- ➔ **Navigate onto your working file in our case server.py and client.py**
- ➔ **Write python filename to run a particular fine make sure python is installed beforehand.**
- ➔ **Now you can freely use the Chat Application**



```
Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh
$ cd OneDrive/

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive
$ cd Desktops
bash: cd: Desktops: No such file or directory

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive
$ cd Desktop

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop
$ cd python/

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python server__.py
Socket successfully created.
Socket(Server) is currently active and listening to requests!!
```

CODE:

Server_Reciever.py:

# Importing the socket module

import socket

# For distributing the messsages along all clients

import select

# For realtime updation of state

import threading

import time

# AF_INET - IPv4 Connection

# SOCK_STREAM - TCP Connection

serverSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

```
# For allowing reconnecting of clients
serverSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
print("Socket successfully created.")


# IPv4 to be used
# The Binding port no is reserved in my laptop
# Defining the HeaderSize of each message to be sent
IP = "127.0.0.1"
port = 3000
HEADERSIZE = 10


# Now we bind our host machine and port with the socket object we created
# The IPv4 address is given above
# The server is now listening for requests from other host machines also connected to the network
serverSocket.bind((IP,port))


#Listening to requests
serverSocket.listen()
print("ANIRUDH VADERA(20BCE2940)")
print("Socket(Server) is currently active and listening to requests!!")


# Stores all those sockets which are connected
socketsList = [serverSocket]
# Client conected
clients = {}


current_acknowledgement = 0
```

```
# A function to recieve messages from the clients connected over the network
def recieveMessage(clientSocket):
  try:
    # We add some extra header information to our msg in order to know the size
of the message we are sending
    # Getting the message header
    messageHeader = clientSocket.recv(HEADERSIZE)
    if not len(messageHeader):
      return False
    # Decoding the message length
    messageLength = int(messageHeader.decode().strip())
    # Returning the message and its header
    return {"Header" : messageHeader , "Data" : clientSocket.recv(messageLength)}
  except:
    return False


# Making a thread for every user connected to the server
def clientThread(notifiedSocket,current_acknowledgement,total_frames):
  while True:
    try:
      initial_message = recieveMessage(notifiedSocket)
      message = recieveMessage(notifiedSocket)
      # The part to do if a client leaves the connection
      if message is False:
        print(f"Closed Connection from {clients[notifiedSocket]['Data'].decode()}")
        socketsList.remove(notifiedSocket)
```

```
          del clients[notifiedSocket]

          break

        user = clients[notifiedSocket]

        print(f"Recieved frame from {user['Data'].decode()} :
{initial_message['Data'].decode()} :: ",f"Packet :
{(int(initial_message['Data'].decode()[-1]))%2}")

        print(f"Recieved message from {user['Data'].decode()} :
{message['Data'].decode()}")



        if(int(initial_message['Data'].decode()[-1]) == current_acknowledgement ):

          print("Correct Frame Recieved")

          current_acknowledgement = current_acknowledgement + 1

          ack_message = ("Ack" + str(current_acknowledgement)).encode()

          ackMessageHeader = f"{len(ack_message):<{HEADERSIZE}}".encode()


          userName = "Server".encode()

          userNameHeader = f"{len(userName):<{HEADERSIZE}}".encode()



          if(current_acknowledgement!=total_frames):

            notifiedSocket.send(ackMessageHeader + ack_message +
userNameHeader + userName)

          else:

            ack_message = ("error1").encode()

            ackMessageHeader = f"{len(ack_message):<{HEADERSIZE}}".encode()

            notifiedSocket.send(ackMessageHeader + ack_message +
userNameHeader + userName)

        else:
            if(int(initial_message['Data'].decode()[-1]) == current_acknowledgement - 1
):
                print("Discarding the Previous repeated frame")
                ack_message = ("Ack" + str(current_acknowledgement)).encode()
                ackMessageHeader = f"{len(ack_message):<{HEADERSIZE}}".encode()

                userName = "Server".encode()
                userNameHeader = f"{len(userName):<{HEADERSIZE}}".encode()

                notifiedSocket.send(ackMessageHeader + ack_message +
userNameHeader + userName)

    except:
        print(f"Closed Connection from {clients[notifiedSocket]['Data'].decode()}")
        if(current_acknowledgement==total_frames):
            print("All the frames were successfully recieved")
        socketsList.remove(notifiedSocket)
        del clients[notifiedSocket]
        break



# Listening to requests infinitely untill interupted
while True:
    # Accepting the user and storing its address in the below defined variables
    clientSocket, clientAddress = serverSocket.accept()

    # Getting the information user wants to send
    user = recieveMessage(clientSocket)
    if user is False:
        continue
    socketsList.append(clientSocket)
    clients[clientSocket] = user

    print(f"Connection from {clientAddress} has been established!! : UserName : {user['Data'].decode()}")



    total_frames_message = recieveMessage(clientSocket)
    total_frames = int(total_frames_message['Data'].decode())

    # We add some extra header information to our msg in order to know the size of the message we are sending
    # The message to be sent
    msg = "Welcome to the server,Thanks for connecting!!"
    # Adding the length of the message as the header information
    msg = f'{len(msg):<{HEADERSIZE}}' + msg

    # Sending information to client socket
    clientSocket.send(msg.encode())

    thread = threading.Thread(target = clientThread, args = (clientSocket,current_acknowledgement,total_frames))
    thread.start()
```

Client_Sender.py

```
# Importing the socket module
```

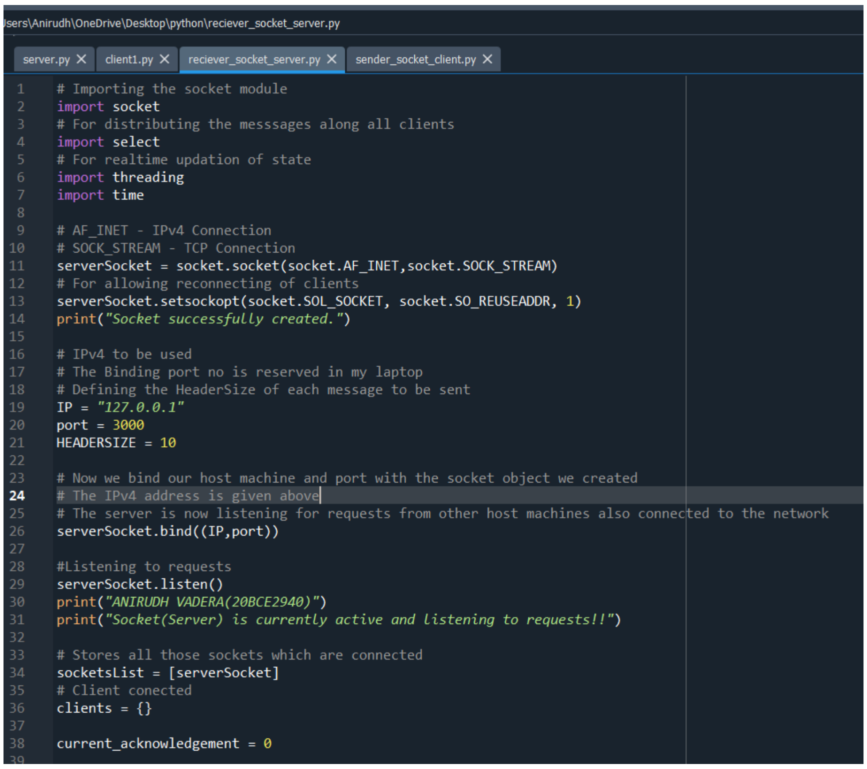
```
import socket
# For distributing the messsages along all clients
import select
# When no message recieved or any other communication error
import errno
import sys
# For realtime updation of state
import threading
import time

# AF_INET - IPv4 Connection
# SOCK_STREAM - TCP Connection
clientSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

# IPv4 to be used
# The port to which the client wants to connect
IP = "127.0.0.1"
port = 3000

# Defining the HeaderSize of each message to be recieved
HEADERSIZE = 10

# The client userName
my_userName = input("UserName : ")

# Connect to the server on this machine or locally
# socket.gethostname() to get the hostname of the server
```

```
clientSocket.connect((IP,port))
# No blocking the incoming messages
clientSocket.setblocking(False)

# Sending the username to the server
userName = my_userName.encode()
userNameHeader = f"{len(userName):<{HEADERSIZE}}".encode()
clientSocket.send(userNameHeader + userName)

window_size_total_frames = input("Enter the total number of frames to be sent : ")
# Sending the total frames to the server
window_size_total_frames = window_size_total_frames .encode()
totalFramesHeader = f"{len(window_size_total_frames):<{HEADERSIZE}}".encode()
clientSocket.send(totalFramesHeader + window_size_total_frames)

ack_flag = 1
total_frames = int(window_size_total_frames)
current_frame = 0

# recieving chunks of data from the server
def recieveData(ack_flag,current_frame,total_frames):
    flag = 0
    # Recieving things infinitely
    while (total_frames!=0):
        try:
            if(flag == 0):# For the initial informative message
                initHeader = clientSocket.recv(HEADERSIZE)
                initLength = int(initHeader.decode().strip())
                msg = clientSocket.recv(initLength).decode()
                print(f"Server > {msg}")
                flag = 1
            else:# For the subsequent messages


                message = input("")
                if message:
                    if(ack_flag!=0):
                        message = message.encode()
                        messageHeader = f"{len(message):<{HEADERSIZE}}".encode()
                        initial_message = ("Frame" + str(current_frame)).encode()
                        initialMessageHeader = 
f"{len(initial_message):<{HEADERSIZE}}".encode()
                        ack_flag=0
                        clientSocket.send(initialMessageHeader + initial_message + 
messageHeader + message)
                        temp = message
                    else:
                        time.sleep(5)
                        print("Waiting for the acknowledgement...")
                        print("Waited 5 seconds ... ")
                        print("The data is lost in between : Resending Data")
                        clientSocket.send(initialMessageHeader + initial_message + 
messageHeader + temp)



                ack_header = clientSocket.recv(HEADERSIZE)
                ack_message_length = int(ack_header.decode().strip())

                ack_message = clientSocket.recv(ack_message_length).decode()



                if(ack_message=="error1"):

                    time.sleep(5)

                    userNameHeader = clientSocket.recv(HEADERSIZE)

                    userNameLength = int(userNameHeader.decode().strip())

                    userName = clientSocket.recv(userNameLength).decode()

                    print("Waited 5 seconds ... Timeout Error")

                    print("No Acknowledgement Recieved : Resending the data")

                    clientSocket.send(initialMessageHeader + initial_message +
messageHeader + temp)

                    ack_header = clientSocket.recv(HEADERSIZE)

                    ack_message_length = int(ack_header.decode().strip())

                    ack_message = clientSocket.recv(ack_message_length).decode()

                    ack_recieved = int(ack_message[-1])

                    if(ack_recieved==(current_frame+1)):

                        if(ack_flag==0):

                          print("Correct Acknowledgement Recieved")

                          ack_flag = 1

                        current_frame = current_frame + 1

                        total_frames = total_frames - 1

                    else:

                        print("Wrong Acknowledgement...")


                    userNameHeader = clientSocket.recv(HEADERSIZE)

                    if not len(userNameHeader):
                        print("Connection closed by the Server")
                        sys.exit()

                    userNameLength = int(userNameHeader.decode().strip())
                    userName = clientSocket.recv(userNameLength).decode()

                    ack_number = int(ack_message[-1])%2

                    print(f"{userName} >> {ack_message} :: Acknowledgement for Packet
{ack_number}")
                else:
                    ack_recieved = int(ack_message[-1])

                    if(ack_recieved==(current_frame+1)):
                        if(ack_flag==0):
                          print("Correct Acknowledgement Recieved")
                          ack_flag = 1
                        current_frame = current_frame + 1
                        total_frames = total_frames - 1
                    else:
                        print("Wrong Acknowledgement...")

                    userNameHeader = clientSocket.recv(HEADERSIZE)
                    if not len(userNameHeader):
                        print("Connection closed by the Server")
                        sys.exit()

                    userNameLength = int(userNameHeader.decode().strip())
```

```
                userName = clientSocket.recv(userNameLength).decode()


                ack_number = int(ack_message[-1])%2


                print(f"{userName} >> {ack_message} :: Acknowledgement for Packet
{ack_number}")



        except IOError as e:
            # This is normal on non blocking connections - when there are no incoming
data, error is going to be raised
            # Some operating systems will indicate that using AGAIN, and some using
WOULDBLOCK error code
            # We are going to check for both - if one of them - that's expected, means no
incoming data, continue as normal
            # If we got different error code - something happened
            if(e.errno != errno.EAGAIN and e.errno != errno.EWOULDBLOCK):
                print("Reading Error",str(e))
                sys.exit()
            continue
        # except Exception as e:
        #     print("General error",str(e))
        #     sys.exit()
    else:
        print("All the frames were sent successfully")


recieveThread = threading.Thread(target = recieveData,
args=(ack_flag,current_frame,total_frames,))
recieveThread.start()
```

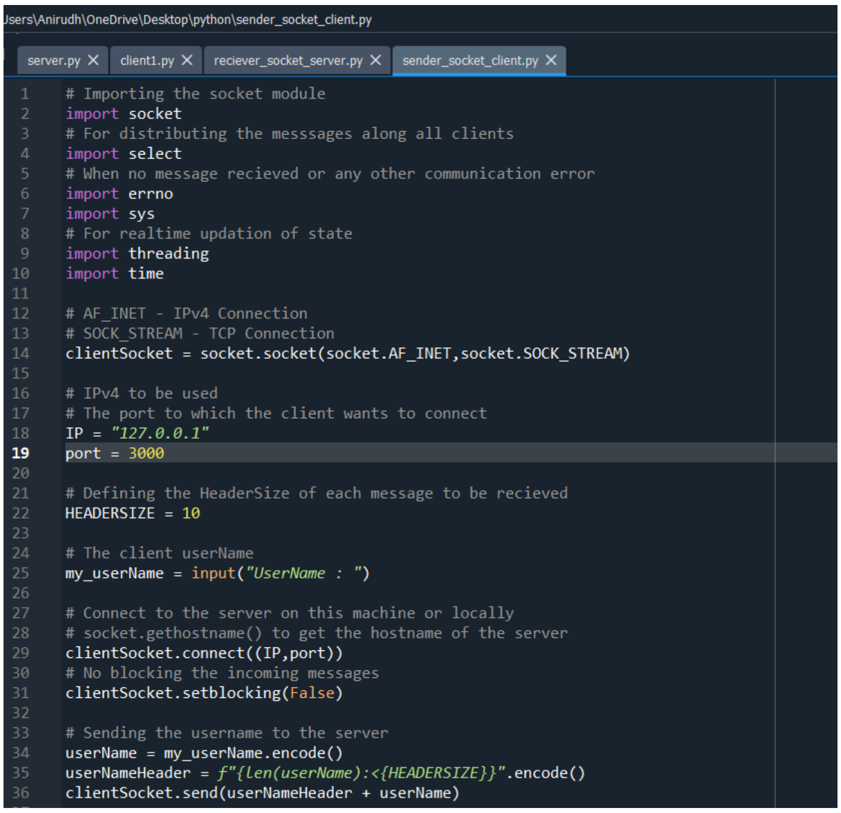
CODE SNIPPETS:

Server_Reciever.py:


Jsers\Anirudh\OneDrive\Desktop\python\reciever_socket_server.py
server.py
client1.py
reciever_socket_server.py
sender_socket_client.py

```python
# Importing the socket module
import socket
# For distributing the messsages along all clients
import select
# For realtime updation of state
import threading
import time

# AF_INET - IPv4 Connection
# SOCK_STREAM - TCP Connection
serverSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
# For allowing reconnecting of clients
serverSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
print("Socket successfully created.")

# IPv4 to be used
# The Binding port no is reserved in my laptop
# Defining the HeaderSize of each message to be sent
IP = "127.0.0.1"
port = 3000
HEADERSIZE = 10

# Now we bind our host machine and port with the socket object we created
# The IPv4 address is given above
# The server is now listening for requests from other host machines also connected to the network
serverSocket.bind((IP,port))

#Listening to requests
serverSocket.listen()
print("ANIRUDH VADERA(20BCE2940)")
print("Socket(Server) is currently active and listening to requests!!")

# Stores all those sockets which are connected
socketsList = [serverSocket]
# Client conected
clients = {}

current_acknowledgement = 0
```

Client_Sender.py

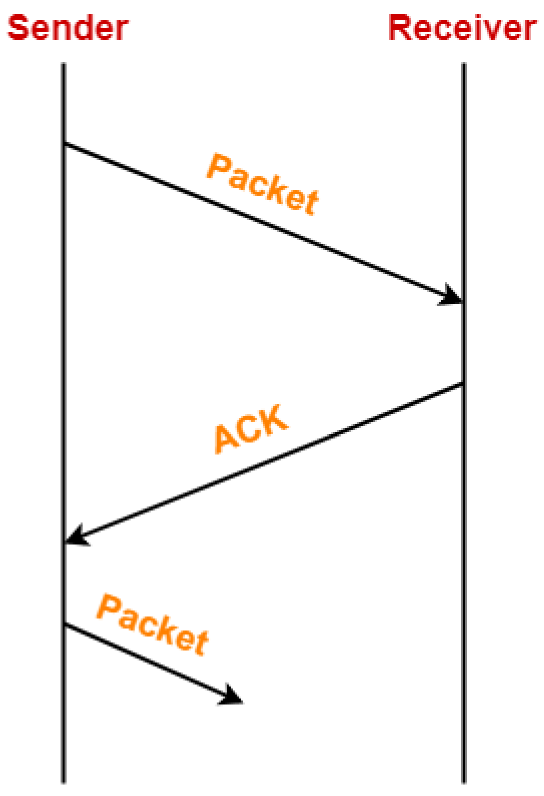

Jsers\Anirudh\OneDrive\Desktop\python\sender_socket_client.py
server.py
client1.py
reciever_socket_server.py
sender_socket_client.py

```python
# Importing the socket module
import socket
# For distributing the messsages along all clients
import select
# When no message recieved or any other communication error
import errno
import sys
# For realtime updation of state
import threading
import time

# AF_INET - IPv4 Connection
# SOCK_STREAM - TCP Connection
clientSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

# IPv4 to be used
# The port to which the client wants to connect
IP = "127.0.0.1"
port = 3000

# Defining the HeaderSize of each message to be recieved
HEADERSIZE = 10

# The client userName
my_userName = input("UserName : ")

# Connect to the server on this machine or locally
# socket.gethostname() to get the hostname of the server
clientSocket.connect((IP,port))
# No blocking the incoming messages
clientSocket.setblocking(False)

# Sending the username to the server
userName = my_userName.encode()
userNameHeader = f"{len(userName):<{HEADERSIZE}}".encode()
clientSocket.send(userNameHeader + userName)
```

OUTPUT:

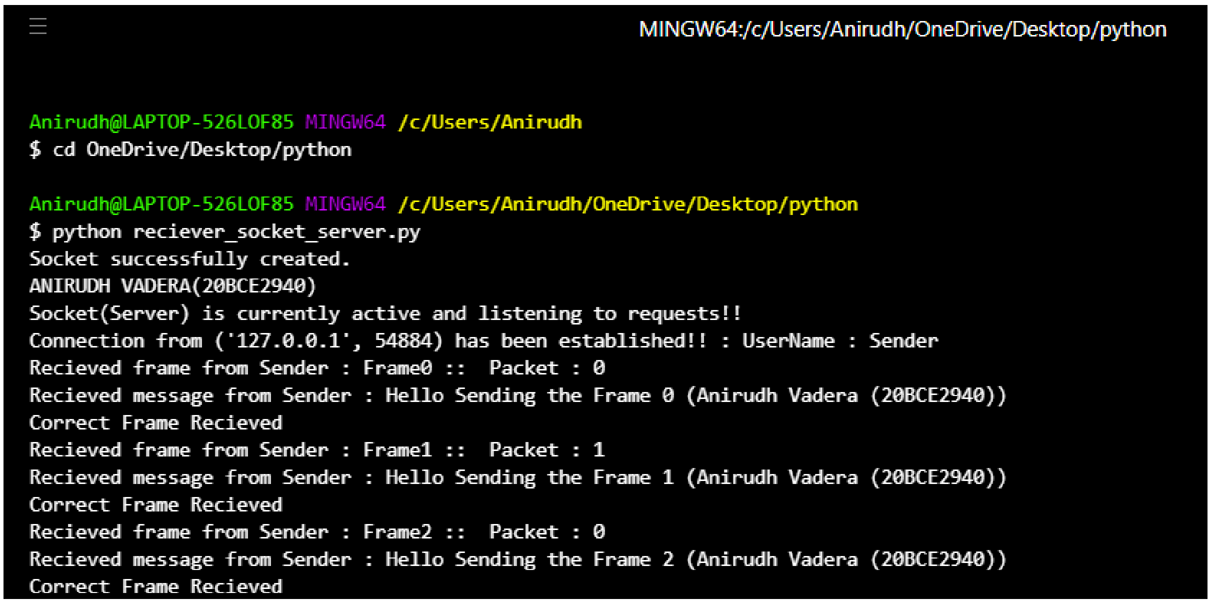
RECIEVER-SERVER :: SENDER-CLIENT

CASE 1: IDEAL CASE:



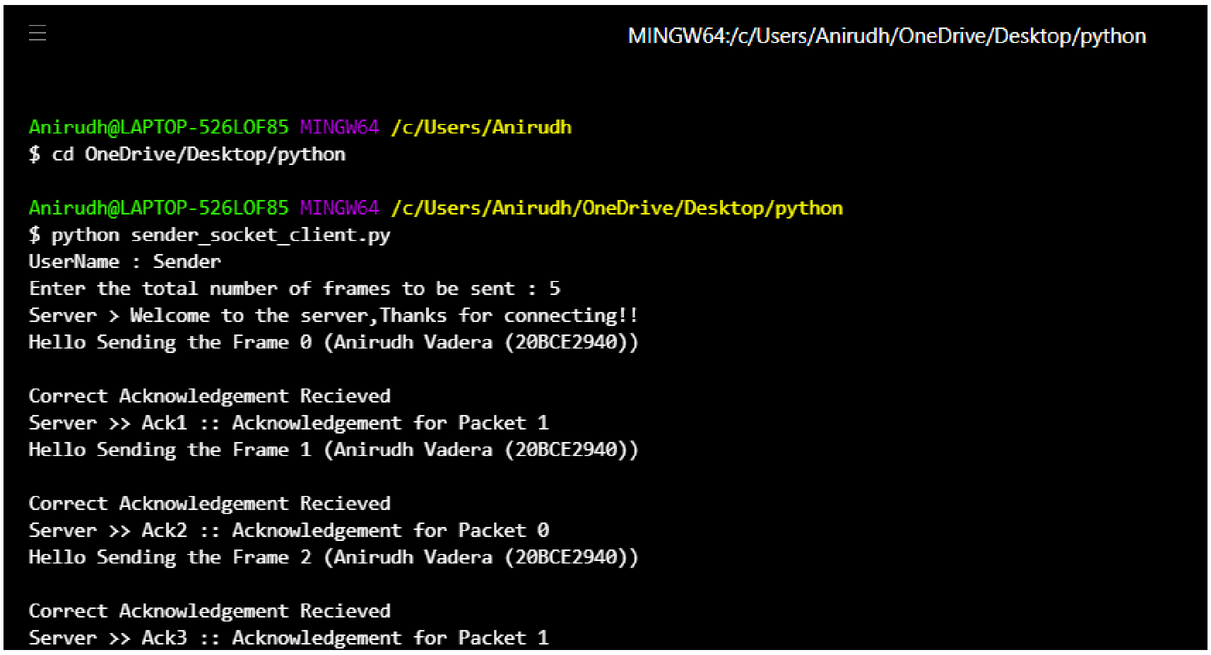
Stop and Wait Protocol

Reciever_Server.py



```
MINGW64:/c/Users/Anirudh/OneDrive/Desktop/python

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh
$ cd OneDrive/Desktop/python

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python reciever_socket_server.py
Socket successfully created.
ANIRUDH VADERA(20BCE2940)
Socket(Server) is currently active and listening to requests!!
Connection from ('127.0.0.1', 54884) has been established!! : UserName : Sender
Recieved frame from Sender : Frame0 ::  Packet : 0
Recieved message from Sender : Hello Sending the Frame 0 (Anirudh Vadera (20BCE2940))
Correct Frame Recieved
Recieved frame from Sender : Frame1 ::  Packet : 1
Recieved message from Sender : Hello Sending the Frame 1 (Anirudh Vadera (20BCE2940))
Correct Frame Recieved
Recieved frame from Sender : Frame2 ::  Packet : 0
Recieved message from Sender : Hello Sending the Frame 2 (Anirudh Vadera (20BCE2940))
Correct Frame Recieved
```

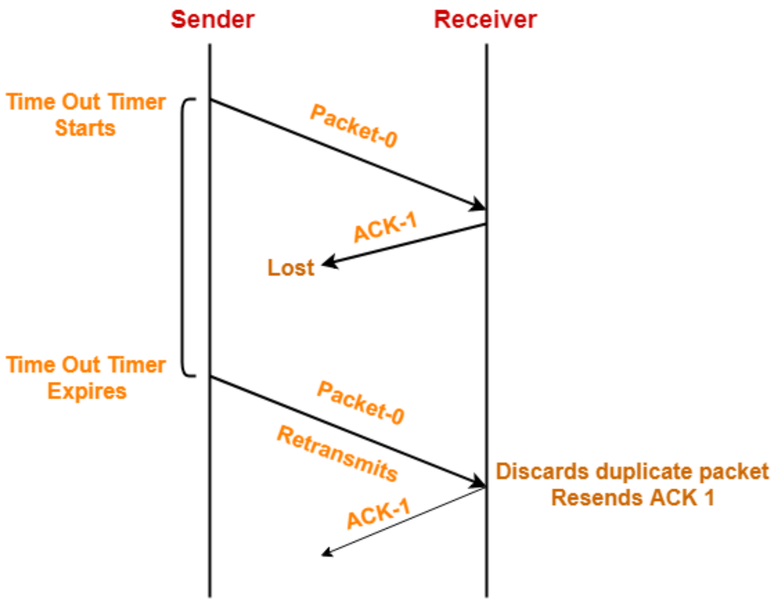
Sender_Client.py



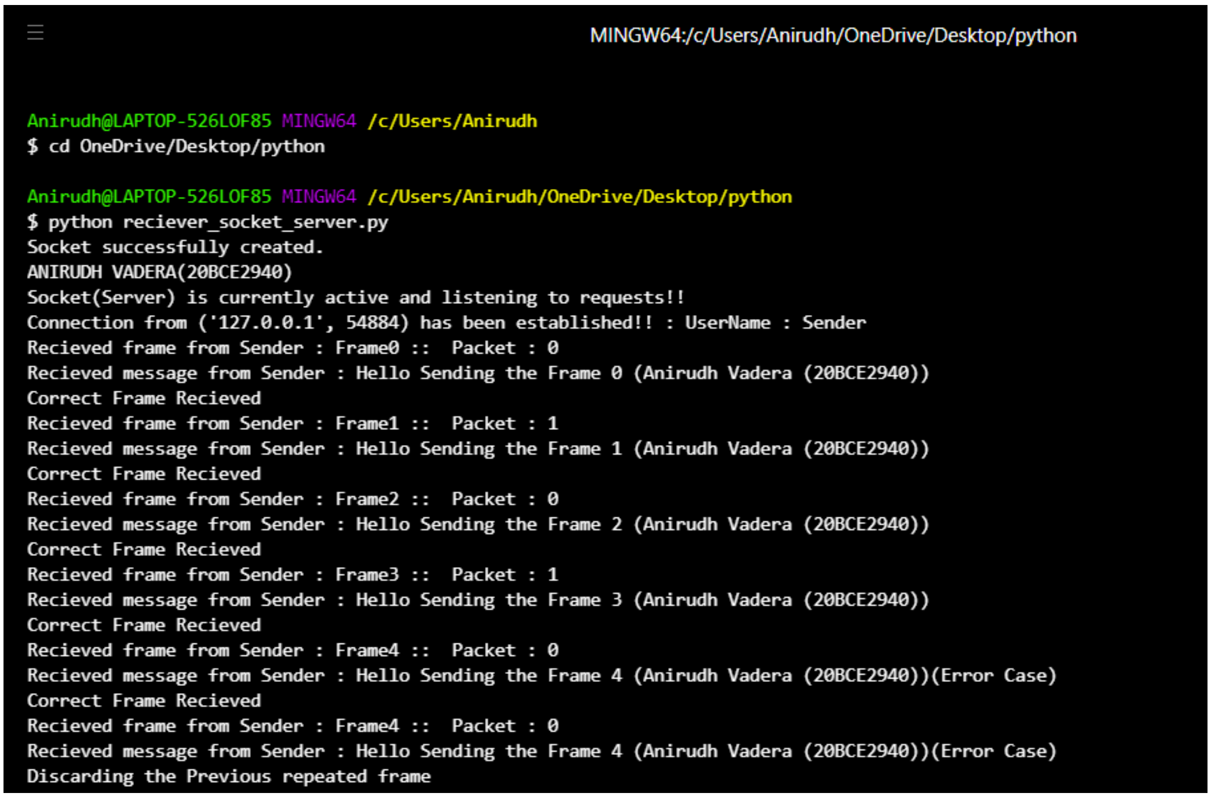
```
MINGW64:/c/Users/Anirudh/OneDrive/Desktop/python

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh
$ cd OneDrive/Desktop/python

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python sender_socket_client.py
UserName : Sender
Enter the total number of frames to be sent : 5
Server > Welcome to the server,Thanks for connecting!!
Hello Sending the Frame 0 (Anirudh Vadera (20BCE2940))

Correct Acknowledgement Recieved
Server >> Ack1 :: Acknowledgement for Packet 1
Hello Sending the Frame 1 (Anirudh Vadera (20BCE2940))

Correct Acknowledgement Recieved
Server >> Ack2 :: Acknowledgement for Packet 0
Hello Sending the Frame 2 (Anirudh Vadera (20BCE2940))

Correct Acknowledgement Recieved
Server >> Ack3 :: Acknowledgement for Packet 1
```

CASE 2: PROBLEM OF LOST ACKNOWLEDGEMENT:

- Sequence number on data packets help to solve the problem of delayed acknowledgement.
- Consider the acknowledgement sent by the receiver gets lost.
- Then, sender retransmits the same data packet after its timer goes off.
- This prevents the occurrence of deadlock.
- The sequence number on the data packet helps the receiver to identify the duplicate data packet.
- Receiver discards the duplicate packet and re-sends the same acknowledgement.



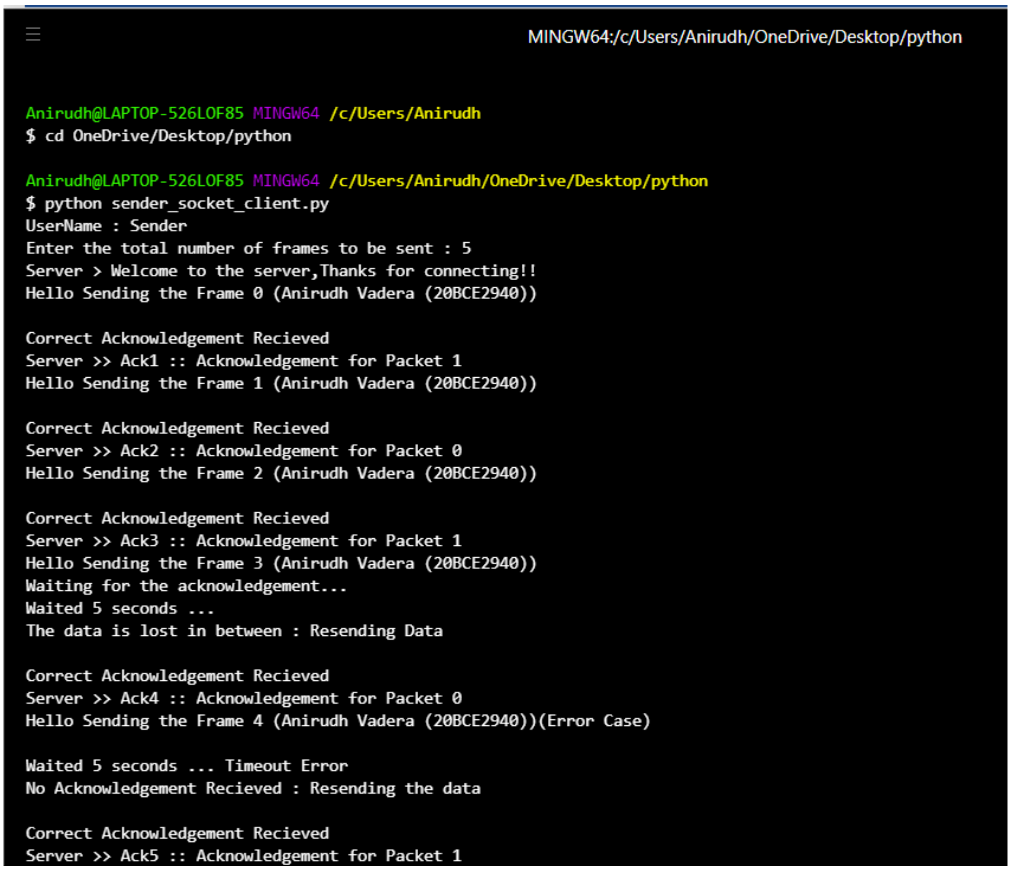
Reciever_Server.py



MINGW64:/c/Users/Anirudh/OneDrive/Desktop/python

```
Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh
$ cd OneDrive/Desktop/python

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python reciever_socket_server.py
Socket successfully created.
ANIRUDH VADERA(20BCE2940)
Socket(Server) is currently active and listening to requests!!
Connection from ('127.0.0.1', 54884) has been established!! : UserName : Sender
Recieved frame from Sender : Frame0 ::  Packet : 0
Recieved message from Sender : Hello Sending the Frame 0 (Anirudh Vadera (20BCE2940))
Correct Frame Recieved
Recieved frame from Sender : Frame1 ::  Packet : 1
Recieved message from Sender : Hello Sending the Frame 1 (Anirudh Vadera (20BCE2940))
Correct Frame Recieved
Recieved frame from Sender : Frame2 ::  Packet : 0
Recieved message from Sender : Hello Sending the Frame 2 (Anirudh Vadera (20BCE2940))
Correct Frame Recieved
Recieved frame from Sender : Frame3 ::  Packet : 1
Recieved message from Sender : Hello Sending the Frame 3 (Anirudh Vadera (20BCE2940))
Correct Frame Recieved
Recieved frame from Sender : Frame4 ::  Packet : 0
Recieved message from Sender : Hello Sending the Frame 4 (Anirudh Vadera (20BCE2940))(Error Case)
Correct Frame Recieved
Recieved frame from Sender : Frame4 ::  Packet : 0
Recieved message from Sender : Hello Sending the Frame 4 (Anirudh Vadera (20BCE2940))(Error Case)
Discarding the Previous repeated frame
```

Sender_Client.py

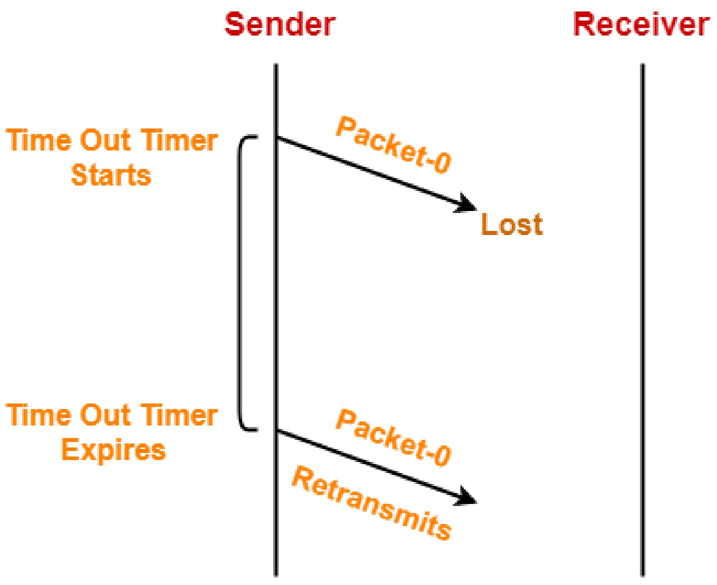


MINGW64:/c/Users/Anirudh/OneDrive/Desktop/python

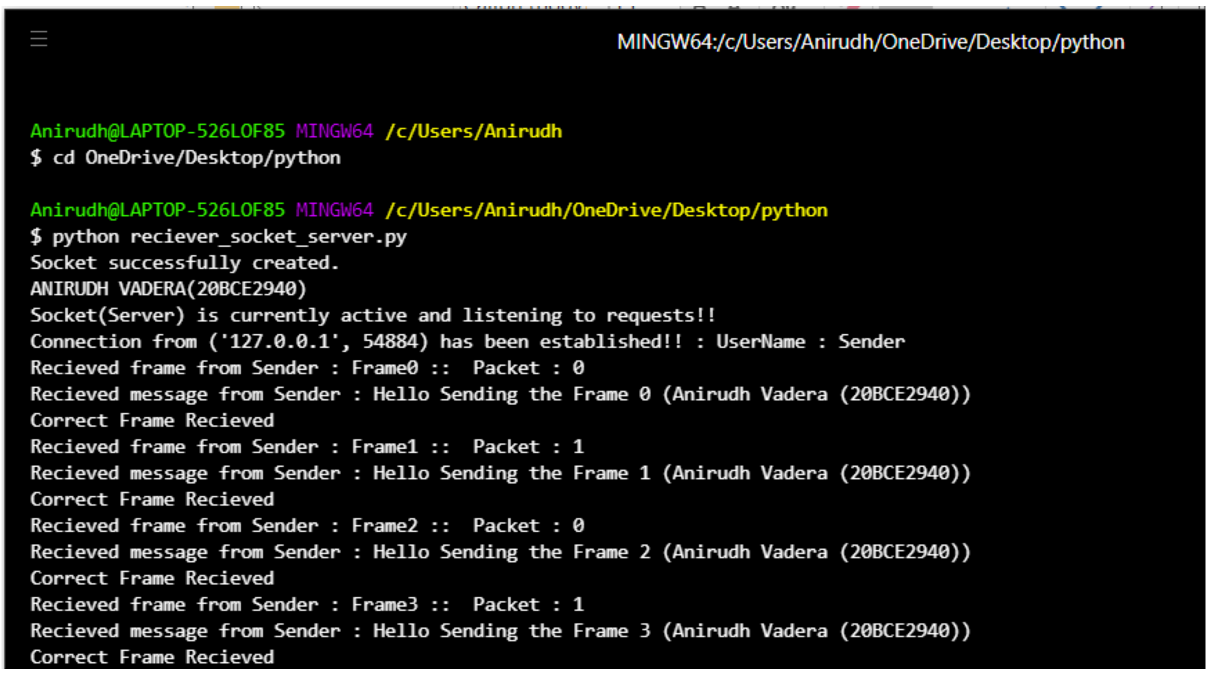
```
Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh
$ cd OneDrive/Desktop/python

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python sender_socket_client.py
UserName : Sender
Enter the total number of frames to be sent : 5
Server > Welcome to the server,Thanks for connecting!!
Hello Sending the Frame 0 (Anirudh Vadera (20BCE2940))

Correct Acknowledgement Recieved
Server >> Ack1 :: Acknowledgement for Packet 1
Hello Sending the Frame 1 (Anirudh Vadera (20BCE2940))

Correct Acknowledgement Recieved
Server >> Ack2 :: Acknowledgement for Packet 0
Hello Sending the Frame 2 (Anirudh Vadera (20BCE2940))

Correct Acknowledgement Recieved
Server >> Ack3 :: Acknowledgement for Packet 1
Hello Sending the Frame 3 (Anirudh Vadera (20BCE2940))
Waiting for the acknowledgement...
Waited 5 seconds ...
The data is lost in between : Resending Data

Correct Acknowledgement Recieved
Server >> Ack4 :: Acknowledgement for Packet 0
Hello Sending the Frame 4 (Anirudh Vadera (20BCE2940))(Error Case)

Waited 5 seconds ... Timeout Error
No Acknowledgement Recieved : Resending the data

Correct Acknowledgement Recieved
Server >> Ack5 :: Acknowledgement for Packet 1
```

CASE 3: PROBLEM OF LOST DATA PACKET:

- Time out timer helps to solve the problem of lost data packet.
- After sending a data packet to the receiver, sender starts the time out timer.
- If the data packet gets acknowledged before the timer expires, sender stops the time out timer.
- If the timer goes off before receiving the acknowledgement, sender retransmits the same data packet.
- After retransmission, sender resets the timer.
- This prevents the occurrence of deadlock.

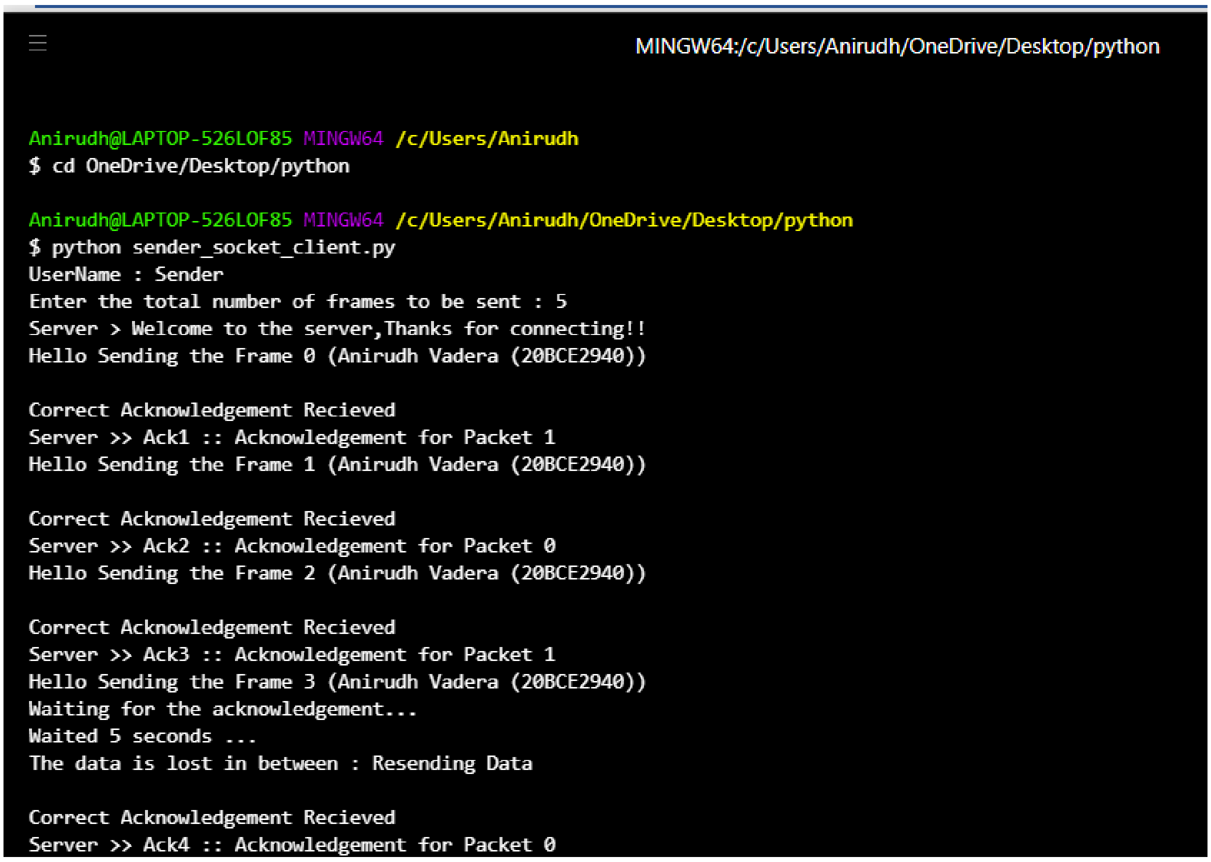


Reciever_Server.py



```
MINGW64:/c/Users/Anirudh/OneDrive/Desktop/python

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh
$ cd OneDrive/Desktop/python

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python reciever_socket_server.py
Socket successfully created.
ANIRUDH VADERA(20BCE2940)
Socket(Server) is currently active and listening to requests!!
Connection from ('127.0.0.1', 54884) has been established!! : UserName : Sender
Recieved frame from Sender : Frame0 ::  Packet : 0
Recieved message from Sender : Hello Sending the Frame 0 (Anirudh Vadera (20BCE2940))
Correct Frame Recieved
Recieved frame from Sender : Frame1 ::  Packet : 1
Recieved message from Sender : Hello Sending the Frame 1 (Anirudh Vadera (20BCE2940))
Correct Frame Recieved
Recieved frame from Sender : Frame2 ::  Packet : 0
Recieved message from Sender : Hello Sending the Frame 2 (Anirudh Vadera (20BCE2940))
Correct Frame Recieved
Recieved frame from Sender : Frame3 ::  Packet : 1
Recieved message from Sender : Hello Sending the Frame 3 (Anirudh Vadera (20BCE2940))
Correct Frame Recieved
```

Sender_Client.py

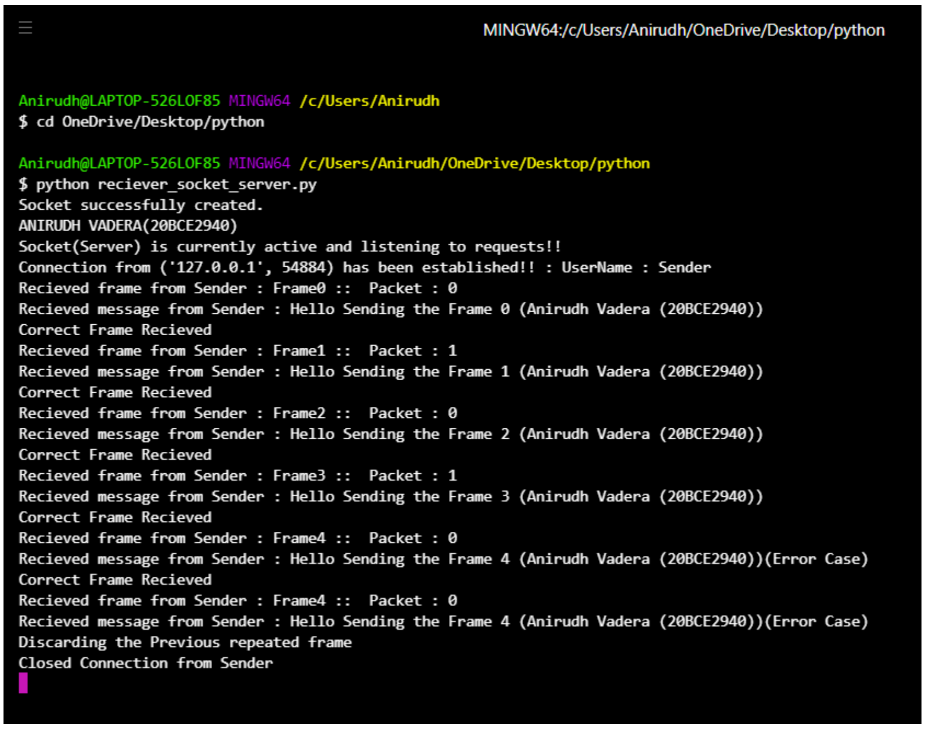


MINGW64:/c/Users/Anirudh/OneDrive/Desktop/python

```
Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh
$ cd OneDrive/Desktop/python

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python sender_socket_client.py
UserName : Sender
Enter the total number of frames to be sent : 5
Server > Welcome to the server,Thanks for connecting!!
Hello Sending the Frame 0 (Anirudh Vadera (20BCE2940))

Correct Acknowledgement Recieved
Server >> Ack1 :: Acknowledgement for Packet 1
Hello Sending the Frame 1 (Anirudh Vadera (20BCE2940))

Correct Acknowledgement Recieved
Server >> Ack2 :: Acknowledgement for Packet 0
Hello Sending the Frame 2 (Anirudh Vadera (20BCE2940))

Correct Acknowledgement Recieved
Server >> Ack3 :: Acknowledgement for Packet 1
Hello Sending the Frame 3 (Anirudh Vadera (20BCE2940))
Waiting for the acknowledgement...
Waited 5 seconds ...
The data is lost in between : Resending Data

Correct Acknowledgement Recieved
Server >> Ack4 :: Acknowledgement for Packet 0
```

After Sending all the frames:

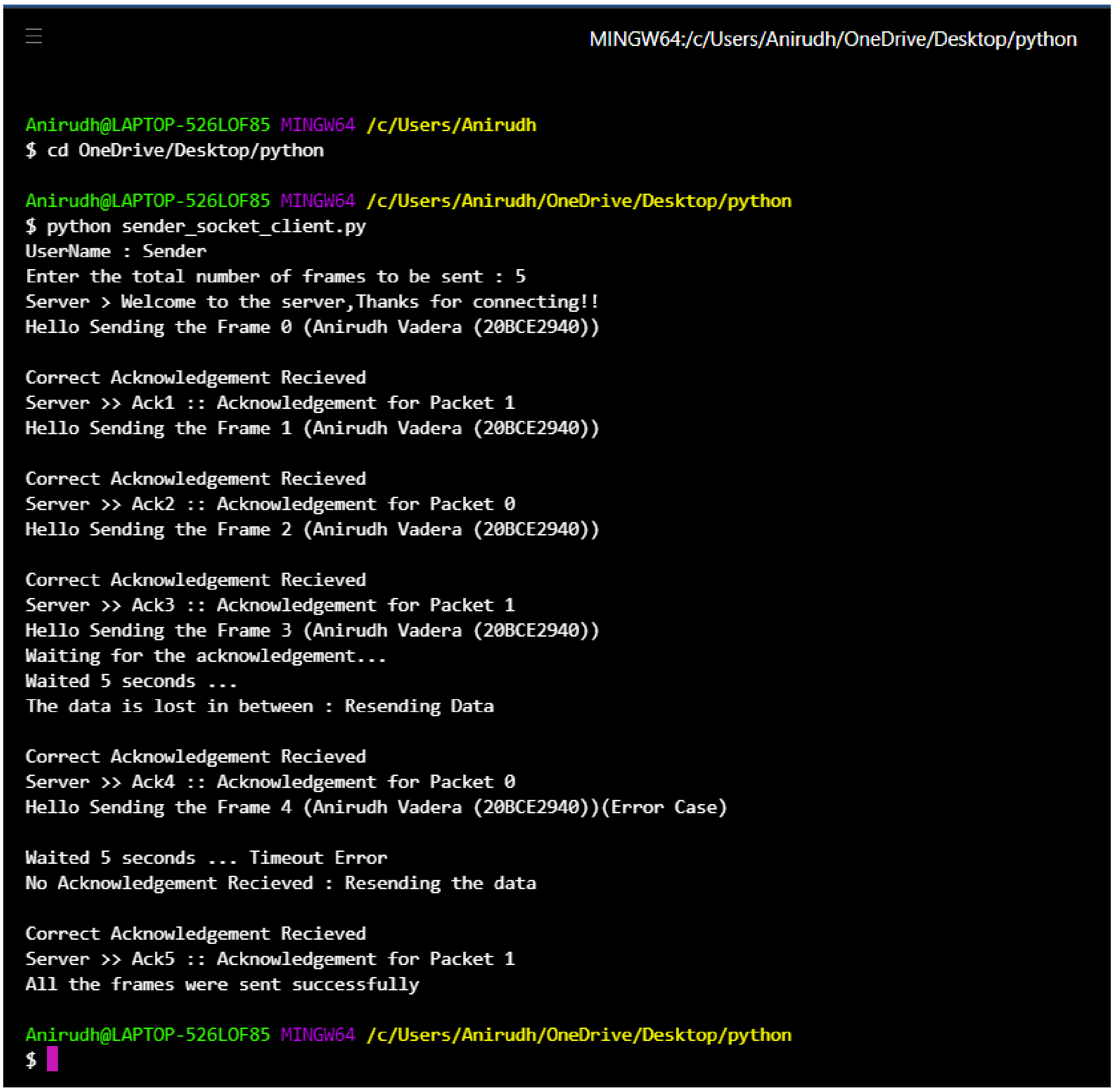
Reciever_Server.py



MINGW64:/c/Users/Anirudh/OneDrive/Desktop/python

```
Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh
$ cd OneDrive/Desktop/python

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python reciever_socket_server.py
Socket successfully created.
ANIRUDH VADERA(20BCE2940)
Socket(Server) is currently active and listening to requests!!
Connection from ('127.0.0.1', 54884) has been established!! : UserName : Sender
Recieved frame from Sender : Frame0 ::  Packet : 0
Recieved message from Sender : Hello Sending the Frame 0 (Anirudh Vadera (20BCE2940))
Correct Frame Recieved
Recieved frame from Sender : Frame1 ::  Packet : 1
Recieved message from Sender : Hello Sending the Frame 1 (Anirudh Vadera (20BCE2940))
Correct Frame Recieved
Recieved frame from Sender : Frame2 ::  Packet : 0
Recieved message from Sender : Hello Sending the Frame 2 (Anirudh Vadera (20BCE2940))
Correct Frame Recieved
Recieved frame from Sender : Frame3 ::  Packet : 1
Recieved message from Sender : Hello Sending the Frame 3 (Anirudh Vadera (20BCE2940))
Correct Frame Recieved
Recieved frame from Sender : Frame4 ::  Packet : 0
Recieved message from Sender : Hello Sending the Frame 4 (Anirudh Vadera (20BCE2940))(Error Case)
Correct Frame Recieved
Recieved frame from Sender : Frame4 ::  Packet : 0
Recieved message from Sender : Hello Sending the Frame 4 (Anirudh Vadera (20BCE2940))(Error Case)
Discarding the Previous repeated frame
Closed Connection from Sender
```

Sender_Client.py



```
Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh
$ cd OneDrive/Desktop/python

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python sender_socket_client.py
UserName : Sender
Enter the total number of frames to be sent : 5
Server > Welcome to the server,Thanks for connecting!!
Hello Sending the Frame 0 (Anirudh Vadera (20BCE2940))

Correct Acknowledgement Recieved
Server >> Ack1 :: Acknowledgement for Packet 1
Hello Sending the Frame 1 (Anirudh Vadera (20BCE2940))

Correct Acknowledgement Recieved
Server >> Ack2 :: Acknowledgement for Packet 0
Hello Sending the Frame 2 (Anirudh Vadera (20BCE2940))

Correct Acknowledgement Recieved
Server >> Ack3 :: Acknowledgement for Packet 1
Hello Sending the Frame 3 (Anirudh Vadera (20BCE2940))
Waiting for the acknowledgement...
Waited 5 seconds ...
The data is lost in between : Resending Data

Correct Acknowledgement Recieved
Server >> Ack4 :: Acknowledgement for Packet 0
Hello Sending the Frame 4 (Anirudh Vadera (20BCE2940))(Error Case)

Waited 5 seconds ... Timeout Error
No Acknowledgement Recieved : Resending the data

Correct Acknowledgement Recieved
Server >> Ack5 :: Acknowledgement for Packet 1
All the frames were sent successfully

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$
```