GO_BACK N AND SELECTIVE REPEAT PROTOCOL USING SOCKETS (FLOW CONTROL)

CSE1004(NETWORK AND COMMUNICATION)LAB:L53-L54

FEBURARY 26, 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.

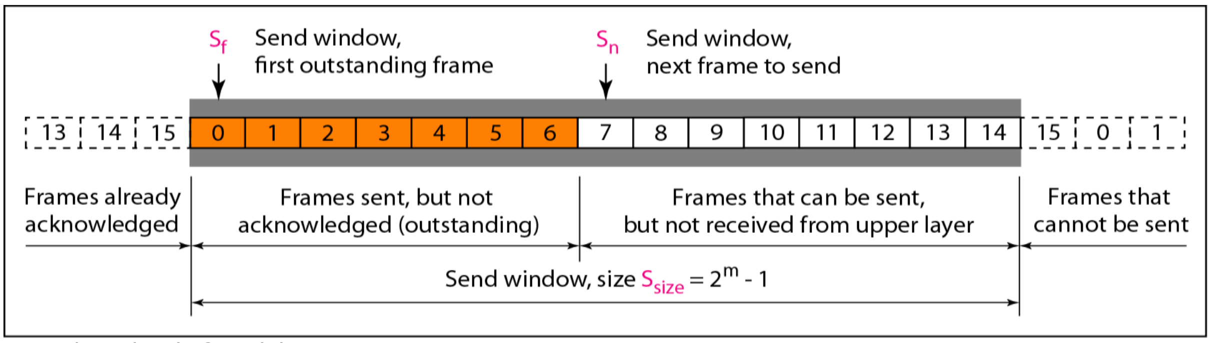
Code to be Implemented : Go_Back_N mechanism

Selective Repeat mechanism

DESCRIPTION:

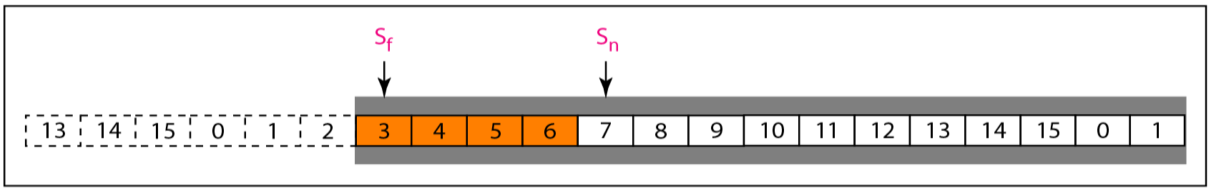
Go Back-N ARQ:

Before understanding the working of Go-Back-N ARQ, we first look at the sliding window protocol. As we know that the sliding window protocol is different from the stop-and-wait protocol. In the stop-and-wait protocol, the sender can send only one frame at a time and cannot send the next frame without receiving the acknowledgment of the previously sent frame, whereas, in the case of sliding window protocol, the multiple frames can be sent at a time. The variations of sliding window protocol are Go-Back-N ARQ and Selective Repeat ARQ. Let's understand 'what is Go-Back-N ARQ'.

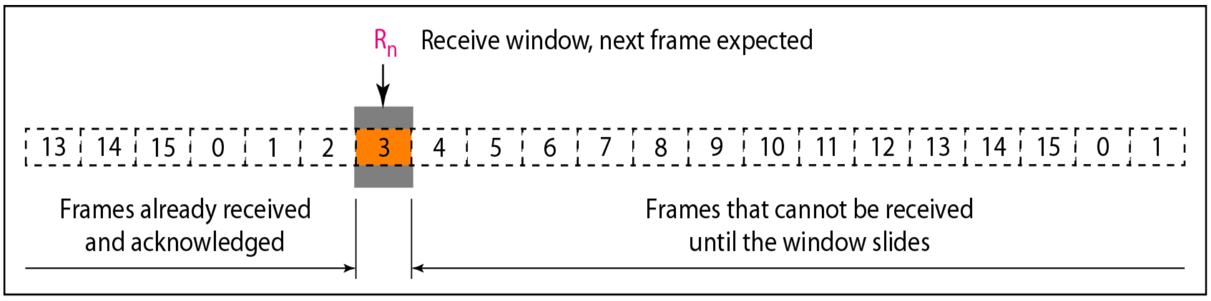


S_f Send window, first outstanding frame
S_n Send window, next frame to send
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already acknowledged
Frames sent, but not acknowledged (outstanding)
Frames that can be sent, but not received from upper layer
Frames that cannot be sent
Send window, size $S_{size} = 2^m - 1$

a. Send window before sliding

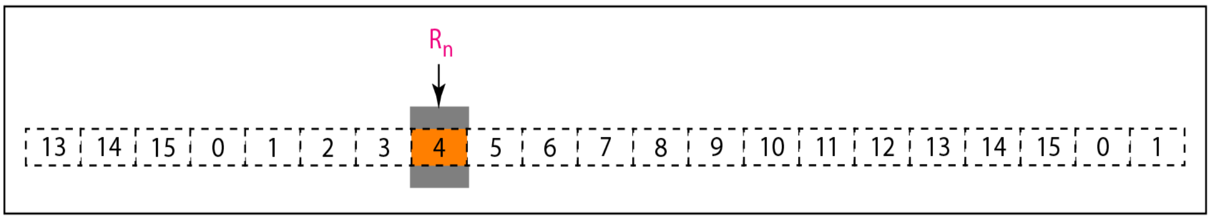


b. Send window after sliding

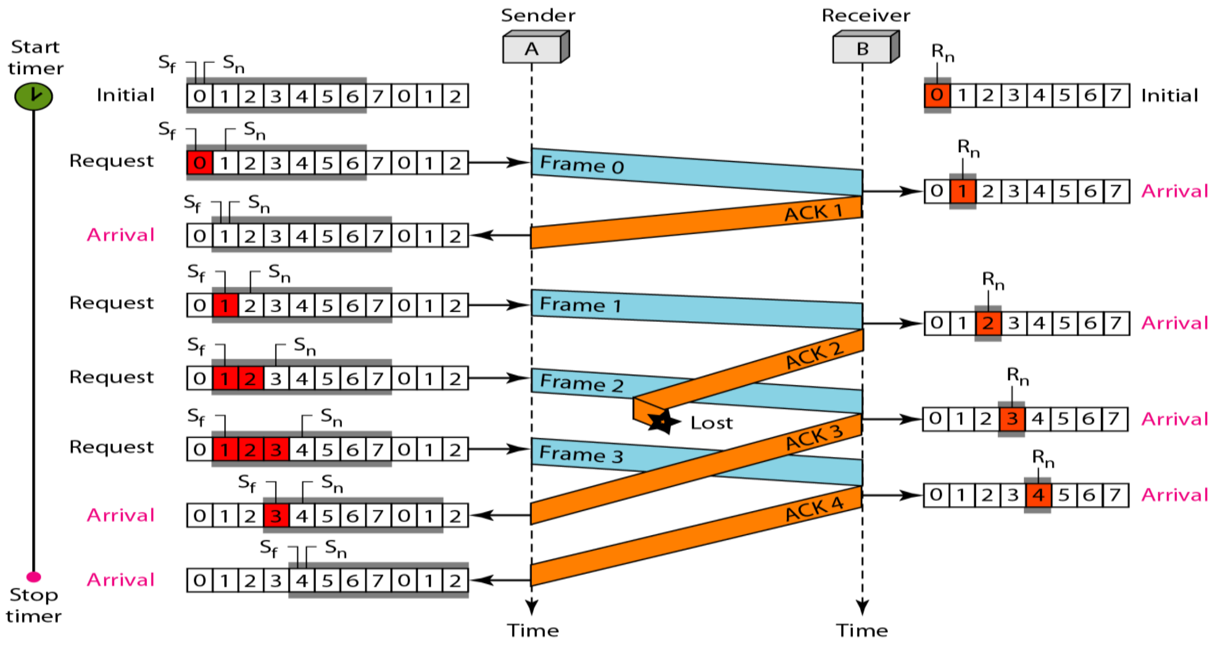


R_n Receive window, next frame expected
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already received and acknowledged
Frames that cannot be received until the window slides

a. Receive window



b. Window after sliding



Working:

In Go-Back-N ARQ, **N** is the sender's window size. Suppose we say that Go-Back-3, which means that the three frames can be sent at a time before expecting the acknowledgment from the receiver.

It uses the principle of protocol pipelining in which the multiple frames can be sent before receiving the acknowledgment of the first frame. If we have five frames and the concept is Go-Back-3, which means that the three frames can be sent, i.e., frame no 1, frame no 2, frame no 3 can be sent before expecting the acknowledgment of frame no 1.

In Go-Back-N ARQ, the frames are numbered sequentially as Go-Back-N ARQ sends the multiple frames at a time that requires the numbering approach to distinguish the frame from another frame, and these numbers are known as the sequential numbers.

The number of frames that can be sent at a time totally depends on the size of the sender's window. So, we can say that 'N' is the number of frames that can be sent at a time before receiving the acknowledgment from the receiver.

If the acknowledgment of a frame is not received within an agreed-upon time period, then all the frames available in the current window will be retransmitted. Suppose we have sent the frame no 5, but we didn't receive the acknowledgment of frame no 5, and the current window is holding three frames, then these three frames will be retransmitted.

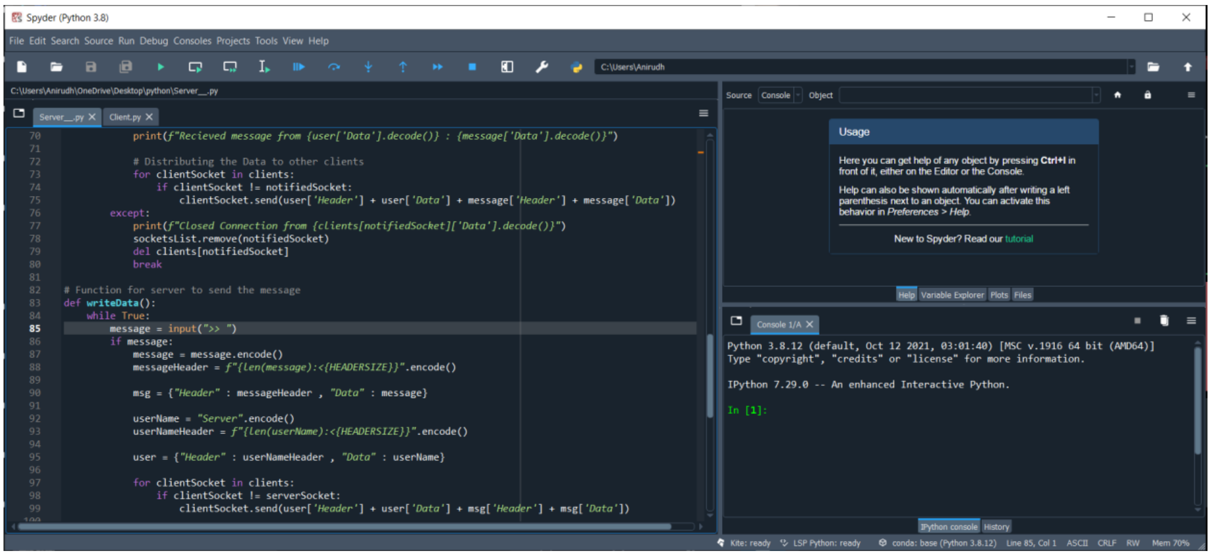
The sequence number of the outbound frames depends upon the size of the sender's window. Suppose the sender's window size is 2, and we have ten frames to send, then the sequence numbers will not be 1,2,3,4,5,6,7,8,9,10. Let's understand through an example.

- N is the sender's window size.
- If the size of the sender's window is 4 then the sequence number will be 0,1,2,3,0,1,2,3,0,1,2, and so on.

The number of bits in the sequence number is 2 to generate the binary sequence 00,01,10,11.

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.
  - ➔ **First Open your python ide**
  - ➔ **I will be using anaconda distribution and a spyder IDE**



```python
                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
Kite: ready LSP Python: ready conda: base (Python 3.8.12) Line 85, Col 1 ASCII CRLF RW Mem 70%

- ➔ We will be using 2 files for our purpose
- ➔ A server file
- ➔ A client file

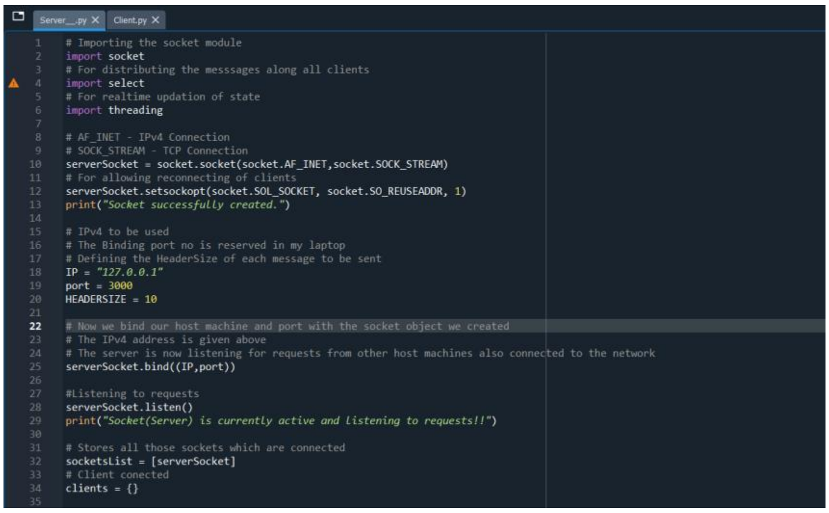
There are some common steps to be followed explained below

- ➔ 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:



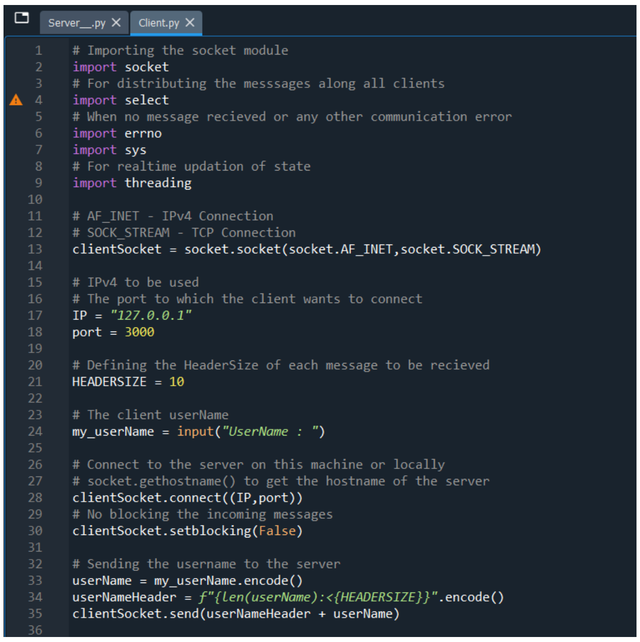
Server__.py Client.py

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

# 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:

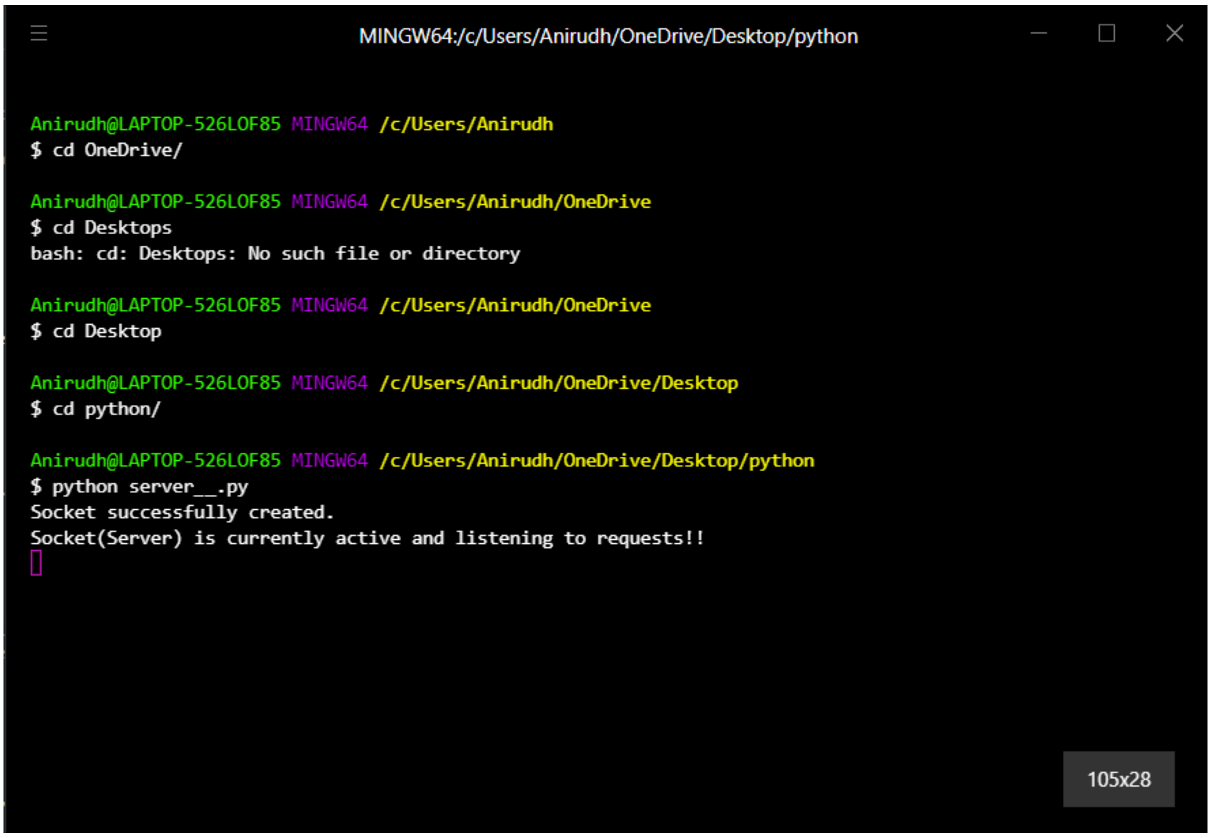


Server__.py Client.py

```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)
```

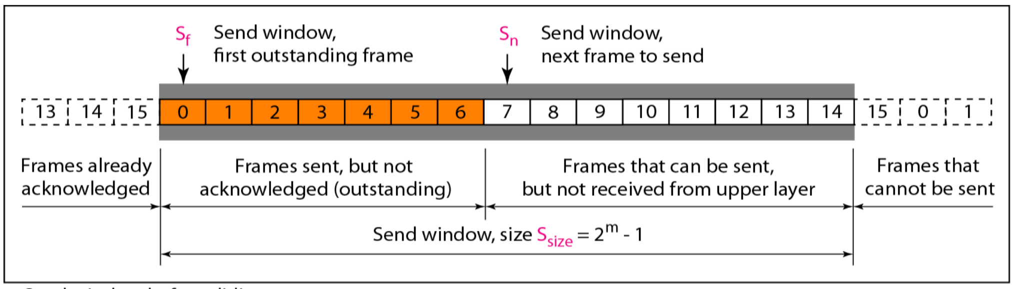
In order to run our Application, we follow the following steps:

- ➔ **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**



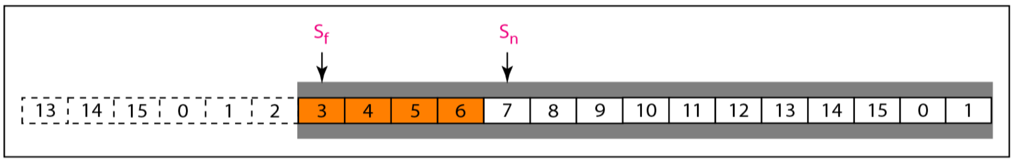
```
MINGW64:/c/Users/Anirudh/OneDrive/Desktop/python

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!!
```

DATA STRUCTURES USED:



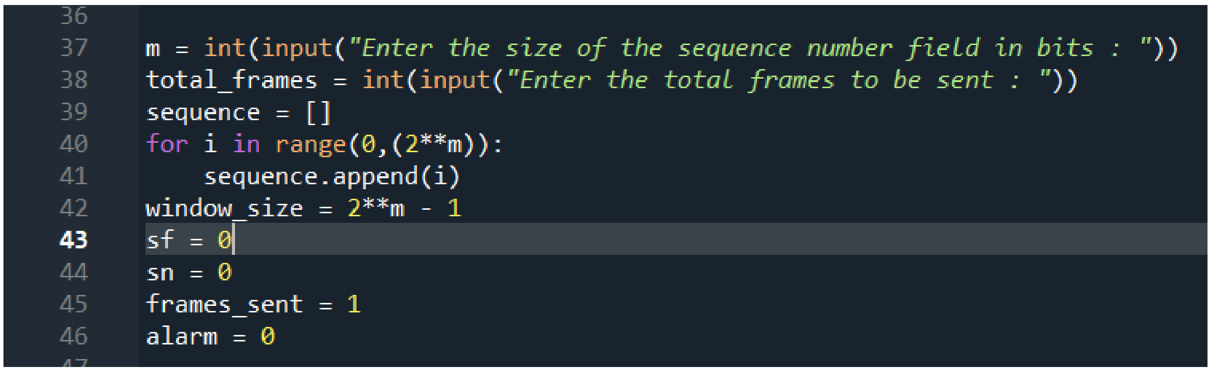
S_f
Send window, first outstanding frame
S_n
Send window, next frame to send
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already acknowledged
Frames sent, but not acknowledged (outstanding)
Frames that can be sent, but not received from upper layer
Frames that cannot be sent
Send window, size $S_{size} = 2^m - 1$

a. Send window before sliding

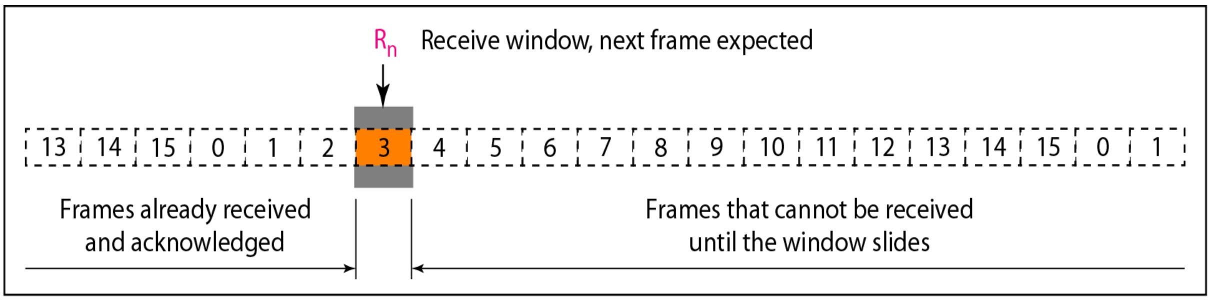


b. Send window after sliding

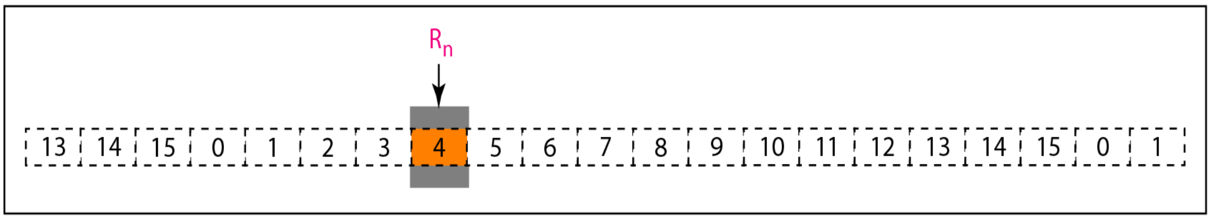
Sender Side:



```
m = int(input("Enter the size of the sequence number field in bits : "))
total_frames = int(input("Enter the total frames to be sent : "))
sequence = []
for i in range(0,(2**m)):
    sequence.append(i)
window_size = 2**m - 1
sf = 0
sn = 0
frames_sent = 1
alarm = 0
```

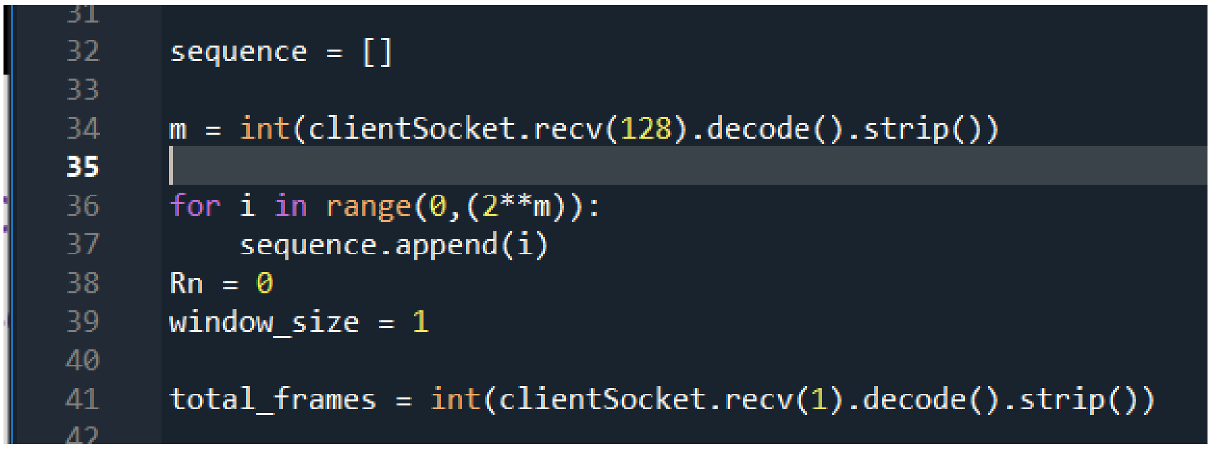

R_n
Receive window, next frame expected
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already received and acknowledged
Frames that cannot be received until the window slides

a. Receive window



b. Window after sliding

Receiver Side:



```
sequence = []

m = int(clientSocket.recv(128).decode().strip())

for i in range(0,(2**m)):
    sequence.append(i)
Rn = 0
window_size = 1

total_frames = int(clientSocket.recv(1).decode().strip())
```

CODE:

Go_Back_N_Reciever.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



# 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))
# Sending the username to the server
userName = my_userName.encode()
clientSocket.send(userName)
```

```
sequence = []

m = int(clientSocket.recv(128).decode().strip())

for i in range(0,(2**m)):
   sequence.append(i)
Rn = 0
window_size = 1

total_frames = int(clientSocket.recv(1).decode().strip())

# recieving chunks of data from the server
def recieveData():
   flag = 0
   global Rn,total_frames
   # Recieving things infinitely
   while (total_frames!=0):
      # try:
         if(flag == 0):# For the initial informative message
            msg = clientSocket.recv(128).decode()
            print(f"Server > {msg}")
            flag = 1
         else:# For the subsequent messages
            message = clientSocket.recv(9).decode()
            if(int(message[-1])==Rn):
               if(message):
                  total_frames = total_frames - 1
```

```
                Rn = Rn + 1
                if(Rn>=len(sequence)):
                    Rn = 0
                if(total_frames!=0):
                    if(Rn!=3):
                        ack_message = "Ack : " + str(sequence[Rn])
                        clientSocket.send(ack_message.encode())
                    else:
                        print("Acknowledgement Lost")
                print(f"Recieved frame from Server: {message}")
            else:
                print("No Action",end="   :    ")
                print("Frame : ",message[-1],"discarded")
    else:
        print("All the frames were recieved successfully")


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

Go_Back_N_Sender.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

# 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
```

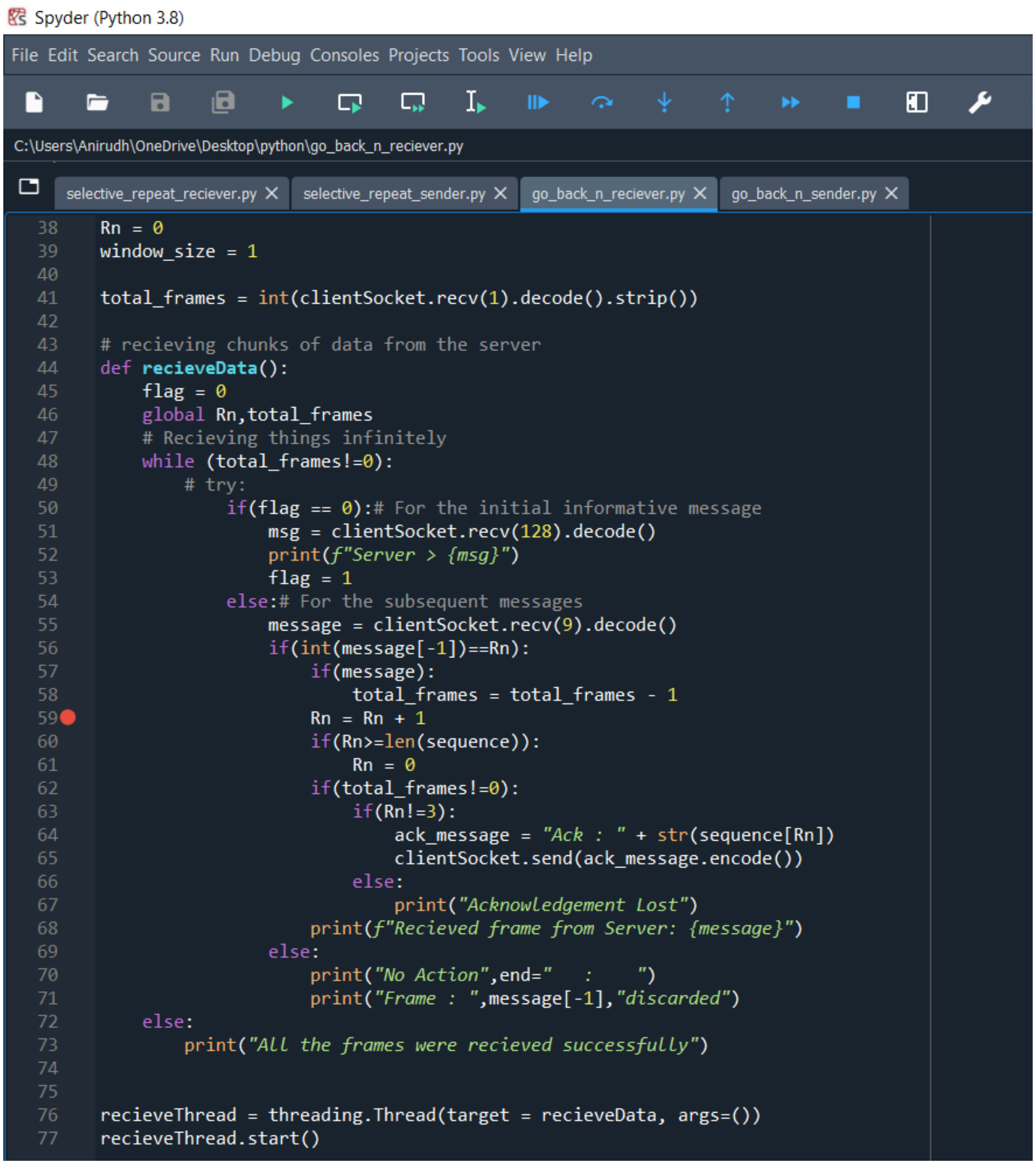
```
clients = {}

m = int(input("Enter the size of the sequence number field in bits : "))
total_frames = int(input("Enter the total frames to be sent : "))
sequence = []
for i in range(0,(2**m)):
   sequence.append(i)
window_size = 2**m - 1
sf = 0
sn = 0
frames_sent = 1
alarm = 0

# A function to recieve messages from the clients connected over the network
def recieveMessage(clientSocket):
   try:
      return {"Data" : clientSocket.recv(7)}
   except:
      return False

def recieve_ack(notifiedSocket):
   global sf,sn,frames_sent
   while True:
      if(sf<=sn):
         ack_message = recieveMessage(notifiedSocket)
         if(ack_message['Data'].decode()):
            user = clients[notifiedSocket]
            print(f"{user['Data'].decode()} >> {ack_message['Data'].decode()}")
```

```
        if(int(ack_message['Data'].decode()[-1])>=(sf+1) or
int(ack_message['Data'].decode()[-1])==0):
            print("Correct Acknowledgement Recieved")
            difference = (int(ack_message['Data'].decode()[-1])-(sf))
            if(difference>=0):
                sf = sf + difference
                frames_sent = frames_sent + difference
                if(sf>=(2**m-1)):
                    sf = sf - (2**m)
            else:
                difference = difference + (2**m)
                sf = sf + difference
                frames_sent = frames_sent + difference
                if(sf>=(2**m-1)):
                    sf = sf - (2**m)
            if(difference > 1):
                print("Number of Jumped acknowledgement : ",difference-1)

# def timer():
#     global alarm,sf,sn
#     while True:
#         time.sleep(1.5)
#         if(sf!=sn):
#             alarm = (alarm + 1) % 2

# thread3 = threading.Thread(target = timer, args = ())
# thread3.start()
# Making a thread for every user connected to the server
def clientThread(notifiedSocket):
    global sf,sn,frames_sent,total_frames,alarm
    temp_flag = 0
    while (frames_sent!=total_frames):
        # if(alarm==1):
        #     sn = sf
        #     alarm = 0
        if((sn-sf)<=(2**m-2) and (sn<=(2**m-1))):
            time.sleep(1)
            message = "Frame : " + str(sequence[sn])
            # if(sn==1 and temp_flag==0):
            #     print("Not sending First Frame for the first time :")
            #     temp_flag = 1
            # else:
            notifiedSocket.send(message.encode())
            sn = sn + 1
            if(sn>=(2**m)):
                sn = sn - (2**m)
            # The part to do if a client leaves the connection
            if message is False:
                print("The message is False")
                print(f"Closed Connection from {clients[notifiedSocket]['Data'].decode()}")
                socketsList.remove(notifiedSocket)
                del clients[notifiedSocket]
                break
    else:
        sf = sf + 1
        print("All the frames were sent successfully")

# 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()}")

    clientSocket.send(str(m).encode())

    # Sending the count of total frames
    clientSocket.send(str(total_frames).encode())

    msg = "Welcome to the server,Thanks for connecting!!"

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

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

thread2= threading.Thread(target = recieve_ack, args = (clientSocket,))
thread2.start()
```

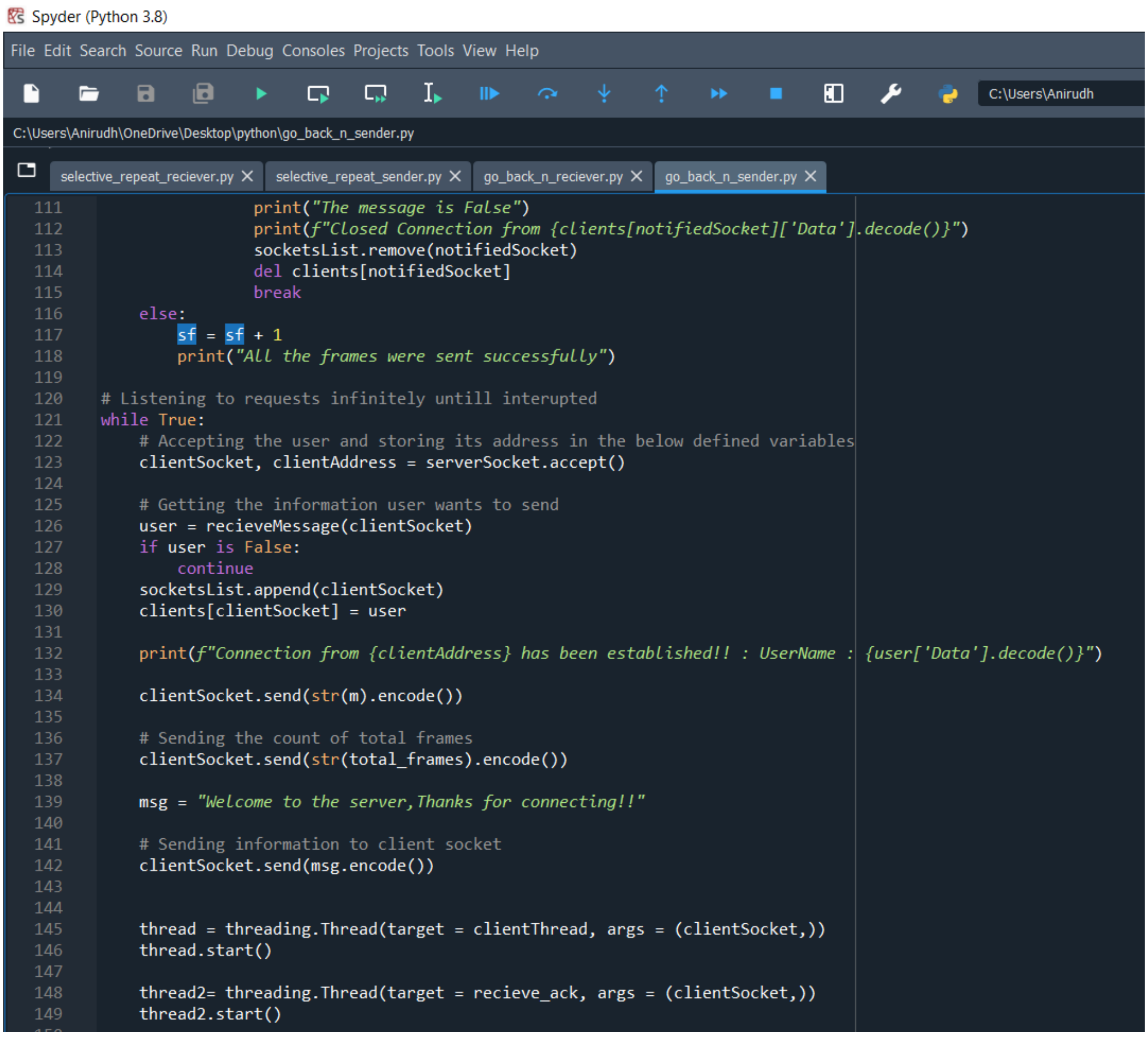
CODE SNIPPETS:

Go_Back_N_Receiver.py:


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Anirudh\OneDrive\Desktop\python\go_back_n_reciever.py
selective_repeat_reciever.py
selective_repeat_sender.py
go_back_n_reciever.py
go_back_n_sender.py

```python
Rn = 0
window_size = 1

total_frames = int(clientSocket.recv(1).decode().strip())

# recieving chunks of data from the server
def recieveData():
    flag = 0
    global Rn,total_frames
    # Recieving things infinitely
    while (total_frames!=0):
        # try:
            if(flag == 0):# For the initial informative message
                msg = clientSocket.recv(128).decode()
                print(f"Server > {msg}")
                flag = 1
            else:# For the subsequent messages
                message = clientSocket.recv(9).decode()
                if(int(message[-1])==Rn):
                    if(message):
                        total_frames = total_frames - 1
                    Rn = Rn + 1
                    if(Rn>=len(sequence)):
                        Rn = 0
                    if(total_frames!=0):
                        if(Rn!=3):
                            ack_message = "Ack : " + str(sequence[Rn])
                            clientSocket.send(ack_message.encode())
                        else:
                            print("Acknowledgement Lost")
                    print(f"Recieved frame from Server: {message}")
                else:
                    print("No Action",end="    :     ")
                    print("Frame : ",message[-1],"discarded")
    else:
        print("All the frames were recieved successfully")


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

Go_Back_N_Sender.py:


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Anirudh
C:\Users\Anirudh\OneDrive\Desktop\python\go_back_n_sender.py
selective_repeat_reciever.py
selective_repeat_sender.py
go_back_n_reciever.py
go_back_n_sender.py

```python
                print("The message is False")
                print(f"Closed Connection from {clients[notifiedSocket]['Data'].decode()}")
                socketsList.remove(notifiedSocket)
                del clients[notifiedSocket]
                break
    else:
        sf = sf + 1
        print("All the frames were sent successfully")

# 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()}")

    clientSocket.send(str(m).encode())

    # Sending the count of total frames
    clientSocket.send(str(total_frames).encode())

    msg = "Welcome to the server,Thanks for connecting!!"

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


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

    thread2= threading.Thread(target = recieve_ack, args = (clientSocket,))
    thread2.start()
```

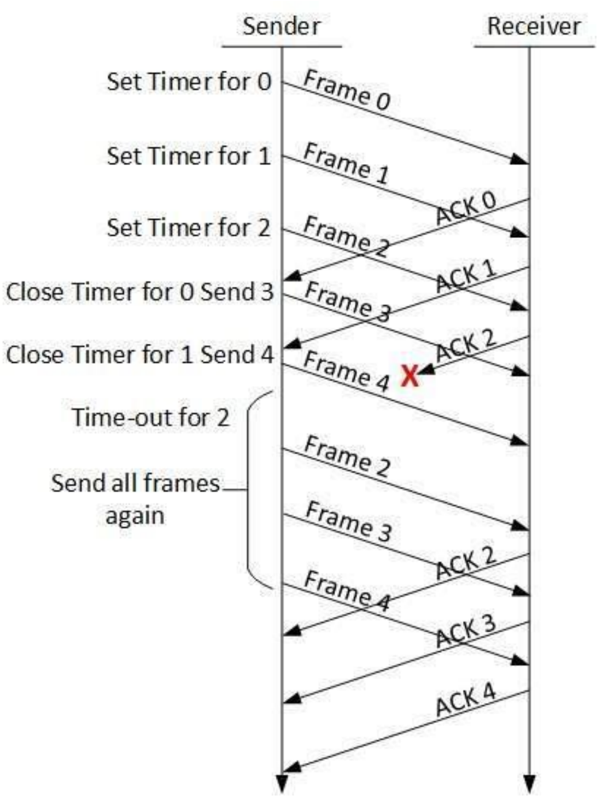
OUTPUT:

RECIEVER-SERVER :: SENDER-CLIENT

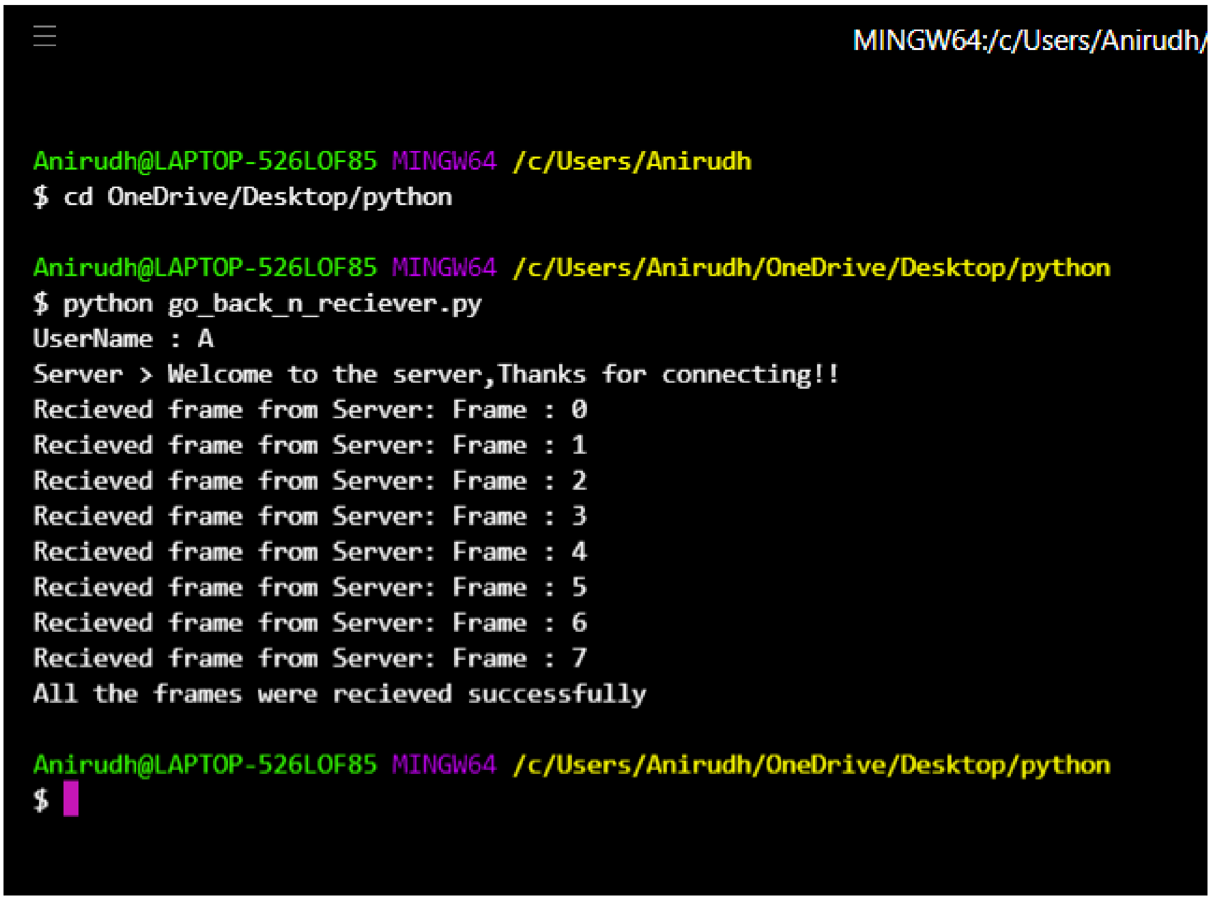
CASE 1: IDEAL CASE:

Number of Bits = 3

Total Frames = 8

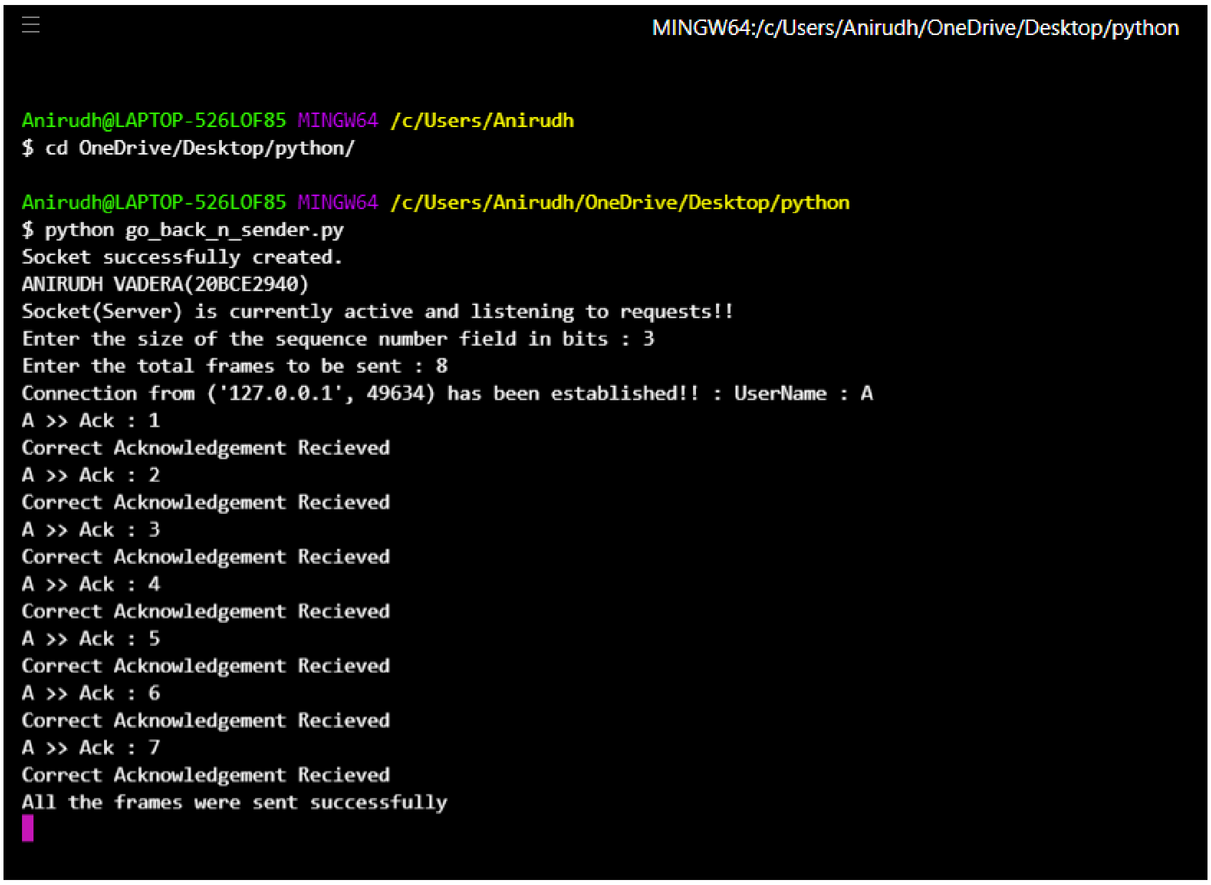


Reciever_Server.py


MINGW64:/c/Users/Anirudh/

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

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python go_back_n_reciever.py
UserName : A
Server > Welcome to the server,Thanks for connecting!!
Recieved frame from Server: Frame : 0
Recieved frame from Server: Frame : 1
Recieved frame from Server: Frame : 2
Recieved frame from Server: Frame : 3
Recieved frame from Server: Frame : 4
Recieved frame from Server: Frame : 5
Recieved frame from Server: Frame : 6
Recieved frame from Server: Frame : 7
All the frames were recieved successfully

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

Sender_Client.py

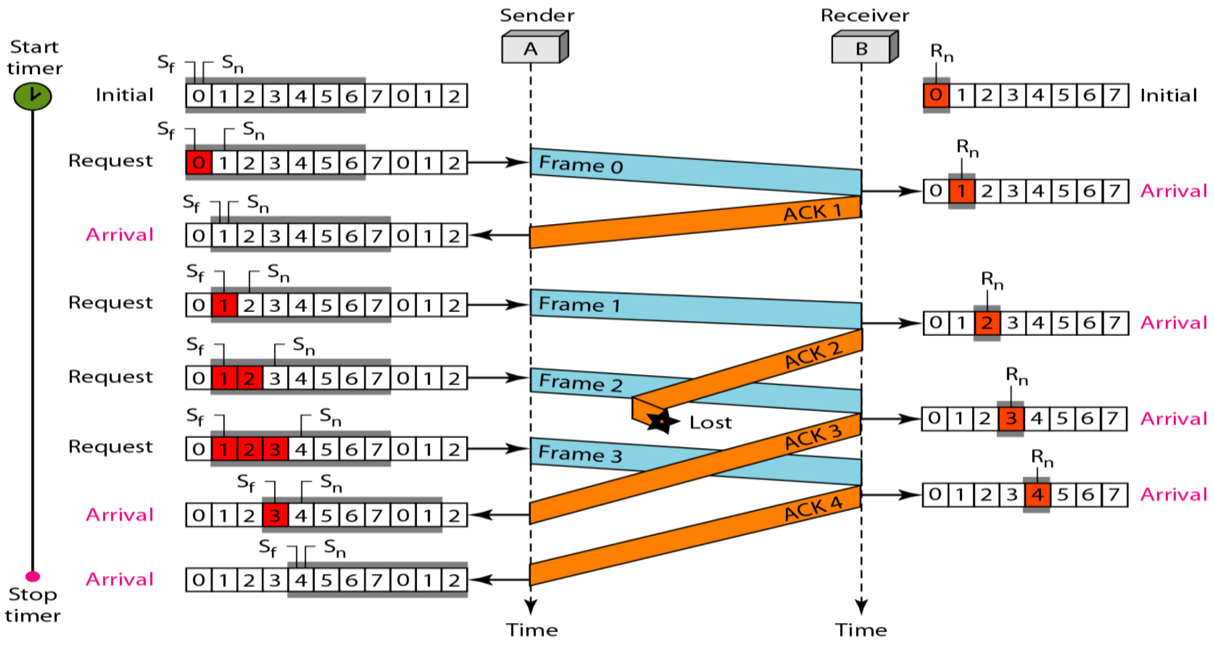


MINGW64:/c/Users/Anirudh/OneDrive/Desktop/python

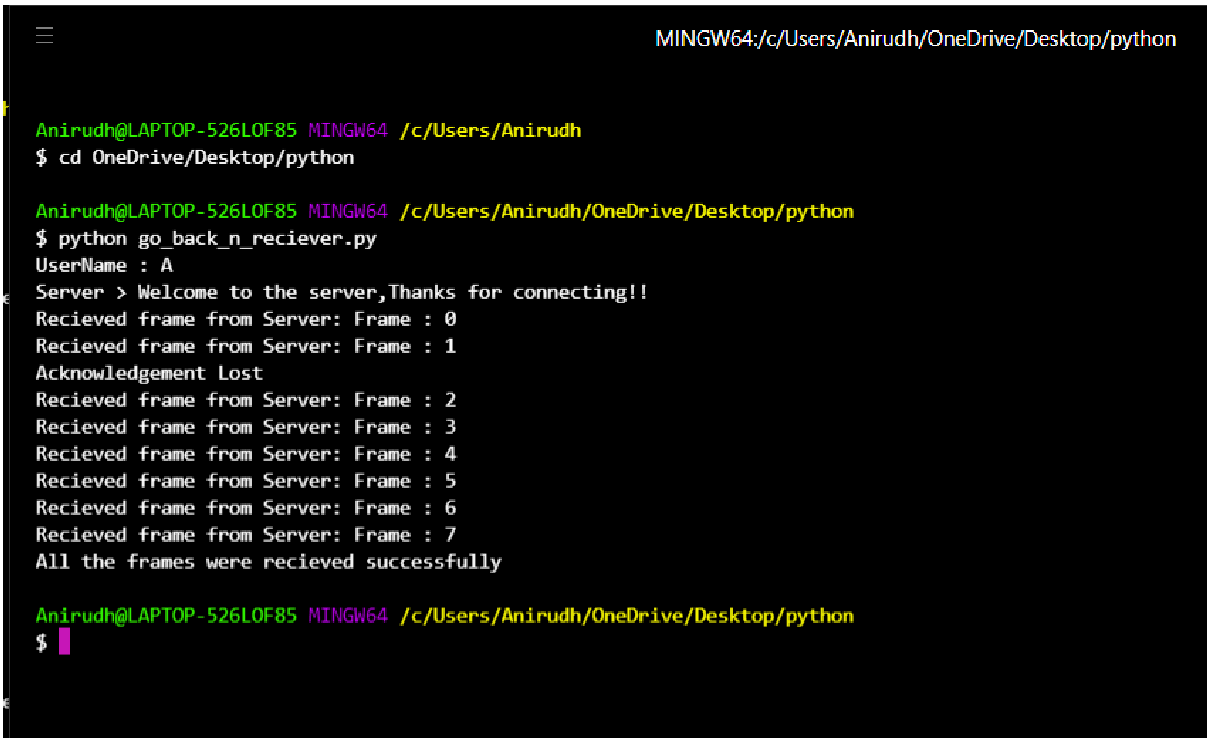
```
Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh
$ cd OneDrive/Desktop/python/

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python go_back_n_sender.py
Socket successfully created.
ANIRUDH VADERA(20BCE2940)
Socket(Server) is currently active and listening to requests!!
Enter the size of the sequence number field in bits : 3
Enter the total frames to be sent : 8
Connection from ('127.0.0.1', 49634) has been established!! : UserName : A
A >> Ack : 1
Correct Acknowledgement Recieved
A >> Ack : 2
Correct Acknowledgement Recieved
A >> Ack : 3
Correct Acknowledgement Recieved
A >> Ack : 4
Correct Acknowledgement Recieved
A >> Ack : 5
Correct Acknowledgement Recieved
A >> Ack : 6
Correct Acknowledgement Recieved
A >> Ack : 7
Correct Acknowledgement Recieved
All the frames were sent successfully
```

CASE 2: PROBLEM OF LOST ACKNOWLEDGEMENT:

Figure 6 shows an example of Go-Back-N. This is an example of a case where the forward channel is reliable, but the reverse is not. No data frames are lost, but some ACKs are delayed and one is lost. The example also shows how cumulative acknowledgments can help if acknowledgments are delayed or lost. After initialization, there are seven sender events. Request events are triggered by data from the network layer; arrival events are triggered by acknowledgments from the physical layer. There is no time-out event here because all outstanding frames are acknowledged before the timer expires. Note that although ACK 2 is lost, ACK 3 serves as both ACK 2 and ACK 3.

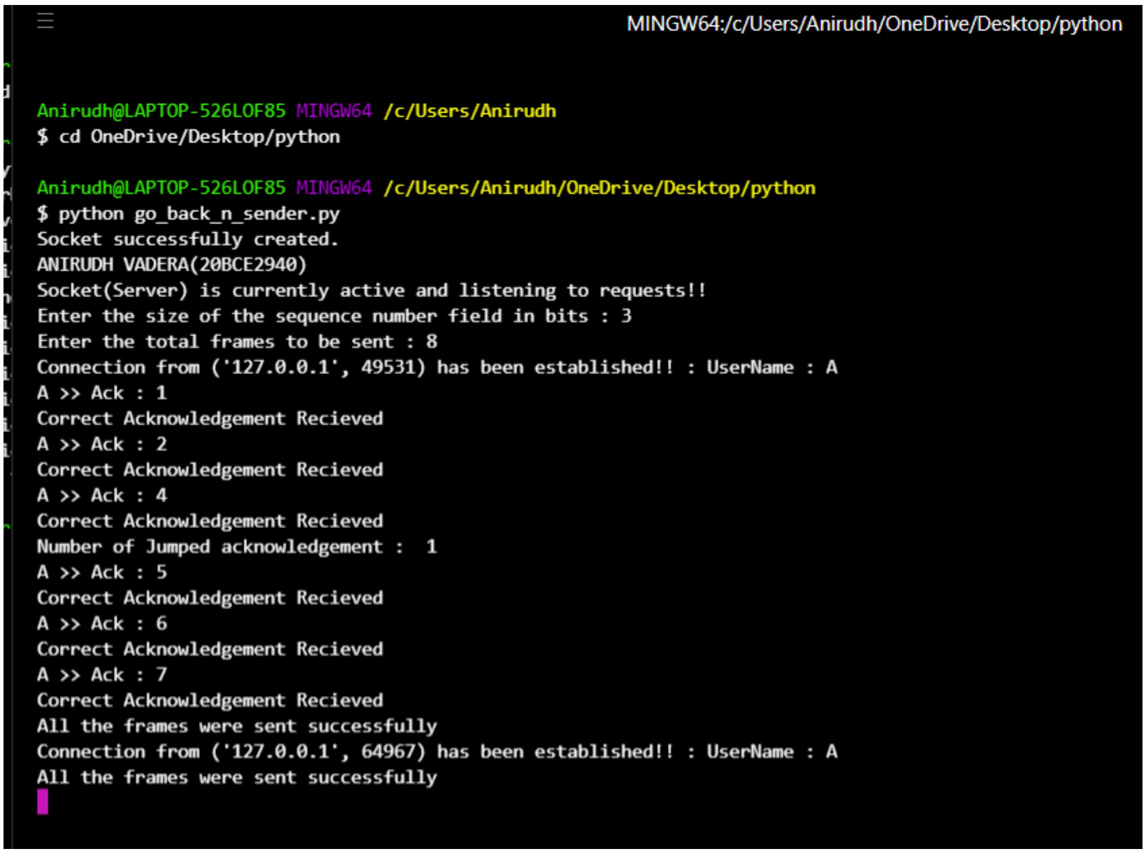


Reciever_Server.py



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

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python go_back_n_reciever.py
UserName : A
Server > Welcome to the server,Thanks for connecting!!
Recieved frame from Server: Frame : 0
Recieved frame from Server: Frame : 1
Acknowledgement Lost
Recieved frame from Server: Frame : 2
Recieved frame from Server: Frame : 3
Recieved frame from Server: Frame : 4
Recieved frame from Server: Frame : 5
Recieved frame from Server: Frame : 6
Recieved frame from Server: Frame : 7
All the frames were recieved successfully

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

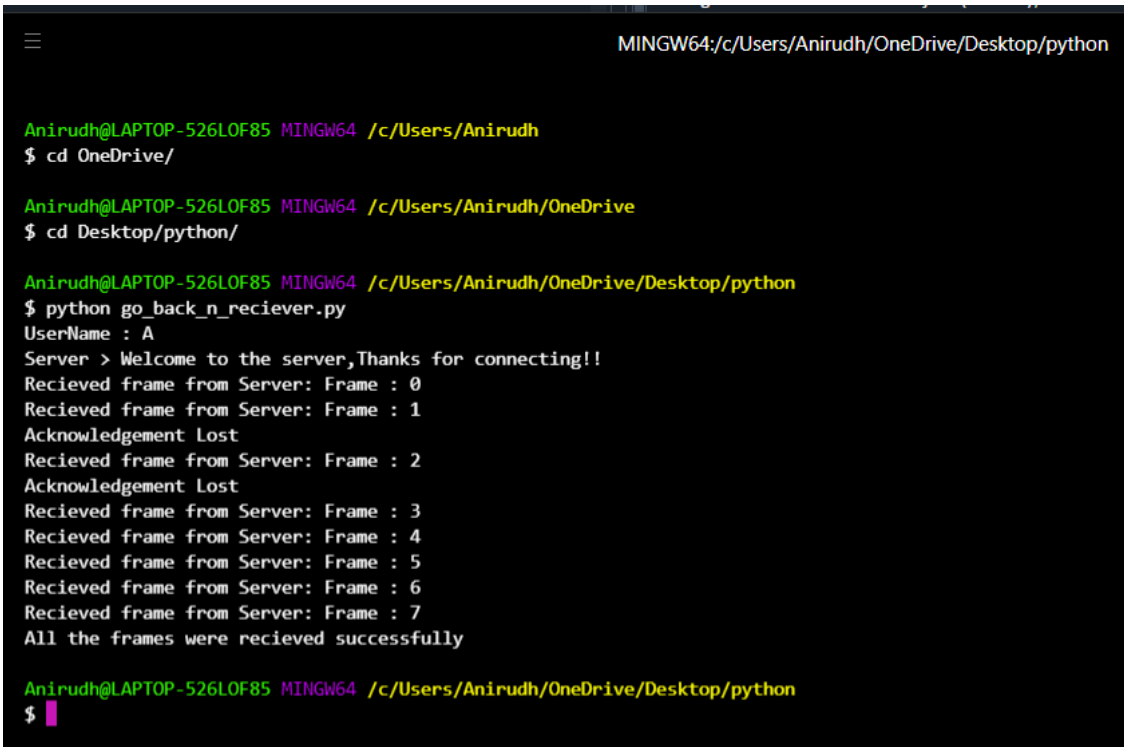
Sender_Client.py



```
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 go_back_n_sender.py
Socket successfully created.
ANIRUDH VADERA(20BCE2940)
Socket(Server) is currently active and listening to requests!!
Enter the size of the sequence number field in bits : 3
Enter the total frames to be sent : 8
Connection from ('127.0.0.1', 49531) has been established!! : UserName : A
A >> Ack : 1
Correct Acknowledgement Recieved
A >> Ack : 2
Correct Acknowledgement Recieved
A >> Ack : 4
Correct Acknowledgement Recieved
Number of Jumped acknowledgement :  1
A >> Ack : 5
Correct Acknowledgement Recieved
A >> Ack : 6
Correct Acknowledgement Recieved
A >> Ack : 7
Correct Acknowledgement Recieved
All the frames were sent successfully
Connection from ('127.0.0.1', 64967) has been established!! : UserName : A
All the frames were sent successfully
```

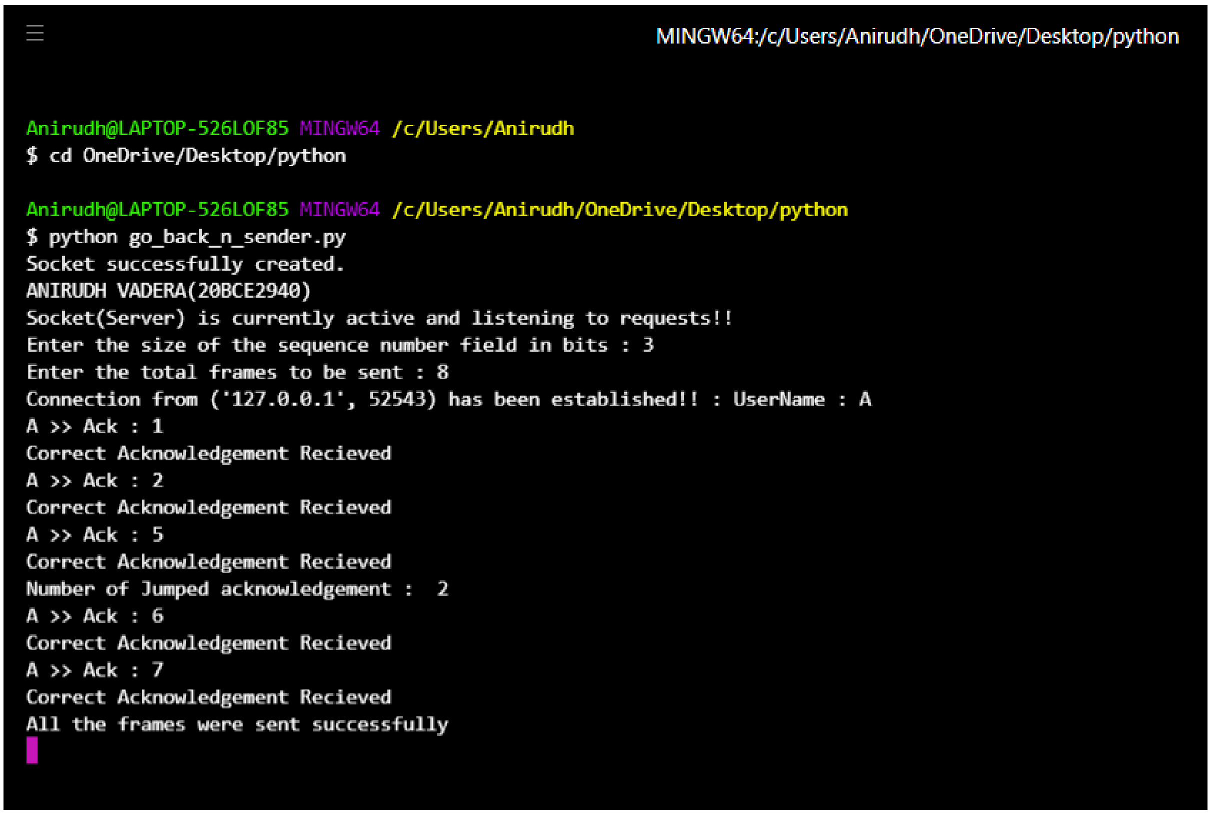
If Ack 3 and 4 are skipped:

Reciever_Server.py



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

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

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

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python go_back_n_reciever.py
UserName : A
Server > Welcome to the server,Thanks for connecting!!
Recieved frame from Server: Frame : 0
Recieved frame from Server: Frame : 1
Acknowledgement Lost
Recieved frame from Server: Frame : 2
Acknowledgement Lost
Recieved frame from Server: Frame : 3
Recieved frame from Server: Frame : 4
Recieved frame from Server: Frame : 5
Recieved frame from Server: Frame : 6
Recieved frame from Server: Frame : 7
All the frames were recieved successfully

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

Sender_Client.py

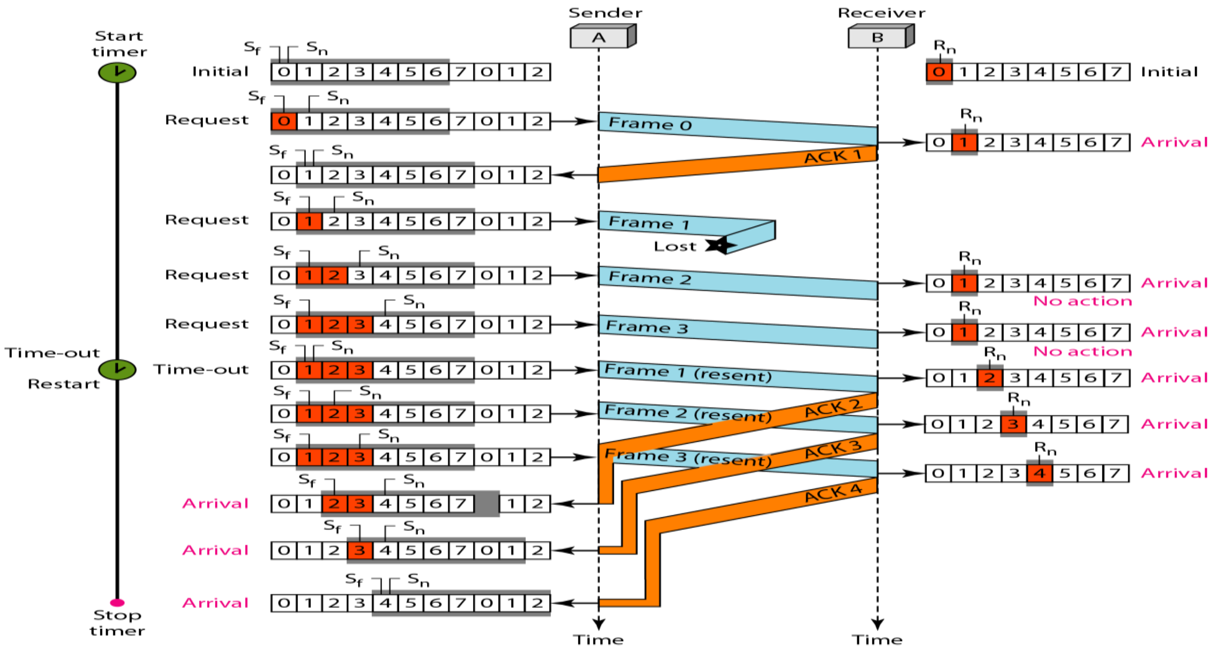

MINGW64:/c/Users/Anirudh/OneDrive/Desktop/python

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

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python go_back_n_sender.py
Socket successfully created.
ANIRUDH VADERA(20BCE2940)
Socket(Server) is currently active and listening to requests!!
Enter the size of the sequence number field in bits : 3
Enter the total frames to be sent : 8
Connection from ('127.0.0.1', 52543) has been established!! : UserName : A
A >> Ack : 1
Correct Acknowledgement Recieved
A >> Ack : 2
Correct Acknowledgement Recieved
A >> Ack : 5
Correct Acknowledgement Recieved
Number of Jumped acknowledgement :  2
A >> Ack : 6
Correct Acknowledgement Recieved
A >> Ack : 7
Correct Acknowledgement Recieved
All the frames were sent successfully
```

CASE 3: PROBLEM OF LOST DATA PACKET:

Figure 11.17 shows what happens when a frame is lost. Frames 0, 1, 2, and 3 are sent. However, frame 1 is lost. The receiver receives frames 2 and 3, but they are discarded because they are received out of order. The sender receives no acknowledgment about frames 1, 2, or 3. Its timer finally expires. The sender sends all outstanding frames (1, 2, and 3) because it does not know what is wrong. Note that the resending of frames 1, 2, and 3 is the response to one single event. When the sender is responding to this event, it cannot accept the triggering of other events. This means that when ACK 2 arrives, the sender is still busy with sending frame 3.

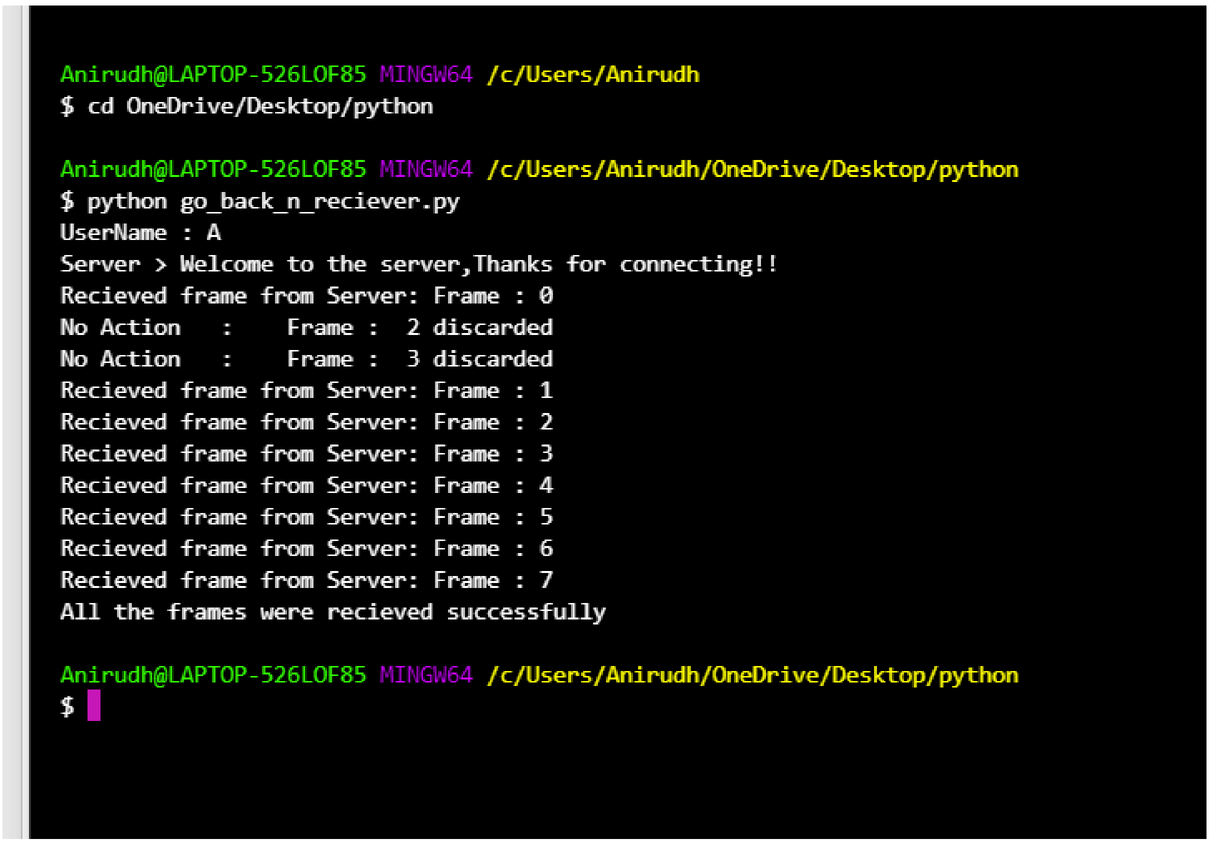
The physical layer must wait until this event is completed and the data link layer goes back to its sleeping state. We have shown a vertical line to indicate the delay. It is the same story with ACK 3; but when ACK 3 arrives, the sender is busy responding to ACK 2. It happens again when ACK 4 arrives. Note that before the second timer expires, all outstanding frames have been sent and the timer is stopped.



We add an extra thread for timer

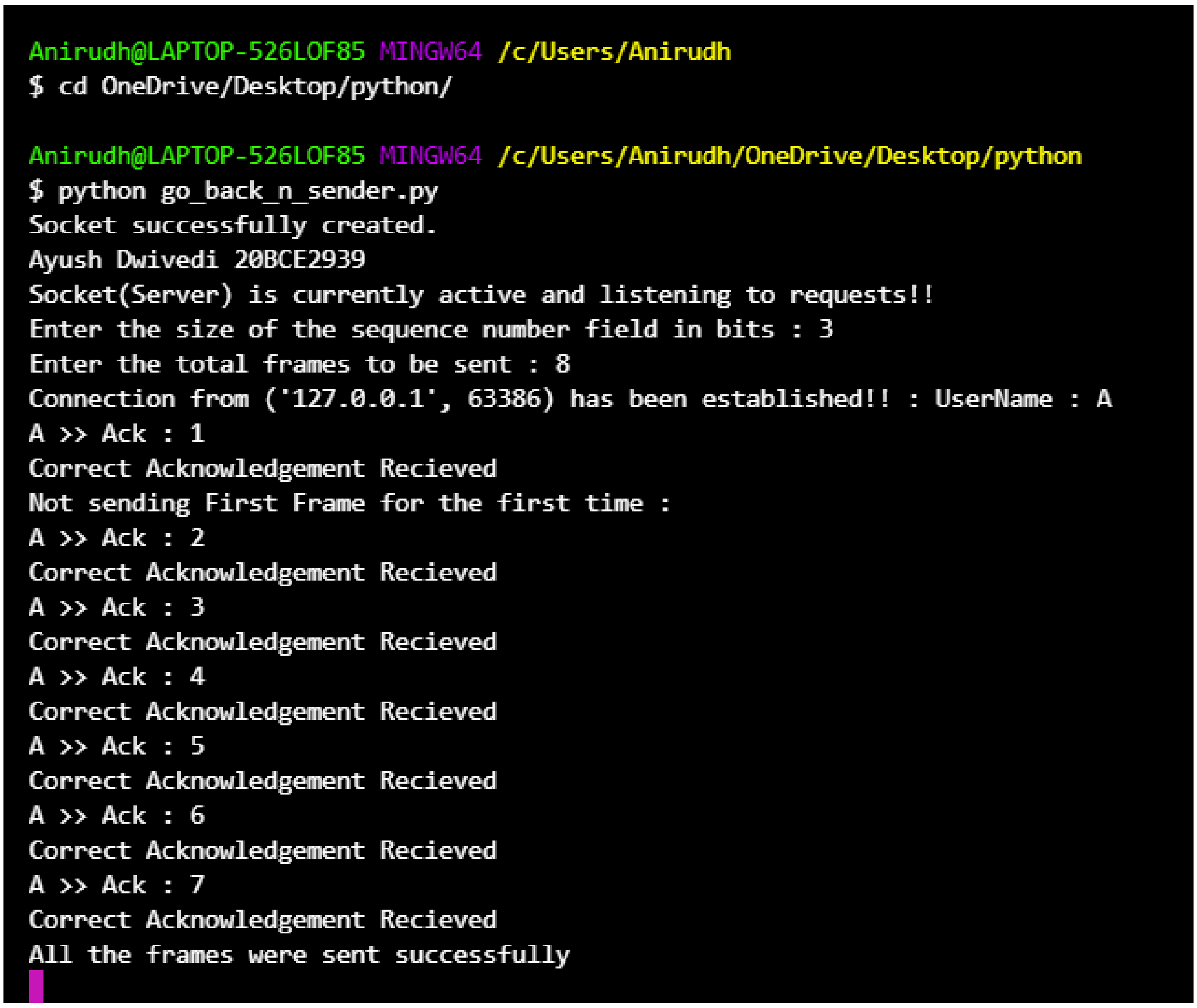
When the timer expires we resend the frame

Reciever_Server.py:



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

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python go_back_n_reciever.py
UserName : A
Server > Welcome to the server,Thanks for connecting!!
Recieved frame from Server: Frame : 0
No Action    :    Frame :  2 discarded
No Action    :    Frame :  3 discarded
Recieved frame from Server: Frame : 1
Recieved frame from Server: Frame : 2
Recieved frame from Server: Frame : 3
Recieved frame from Server: Frame : 4
Recieved frame from Server: Frame : 5
Recieved frame from Server: Frame : 6
Recieved frame from Server: Frame : 7
All the frames were recieved successfully

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

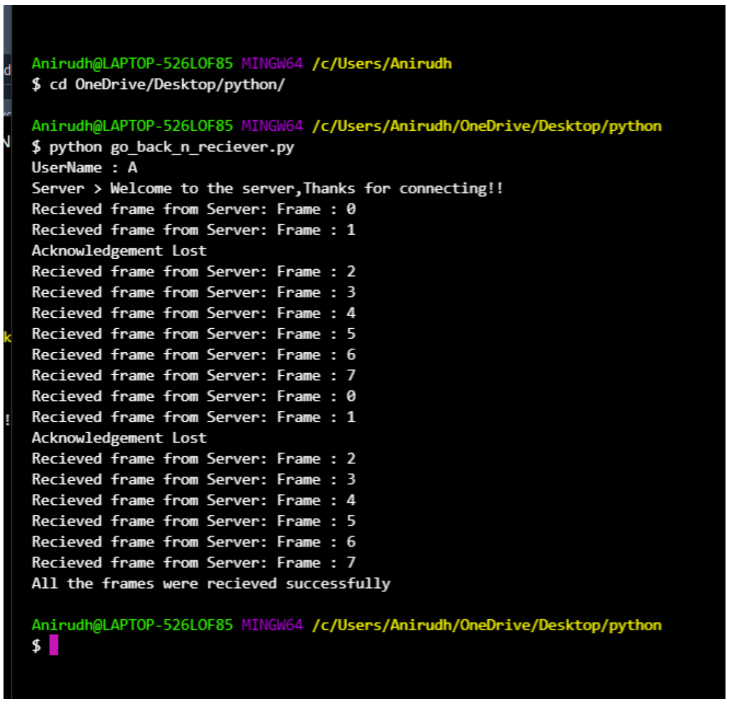
Sender_Client.py:



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

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python go_back_n_sender.py
Socket successfully created.
Ayush Dwivedi 20BCE2939
Socket(Server) is currently active and listening to requests!!
Enter the size of the sequence number field in bits : 3
Enter the total frames to be sent : 8
Connection from ('127.0.0.1', 63386) has been established!! : UserName : A
A >> Ack : 1
Correct Acknowledgement Recieved
Not sending First Frame for the first time :
A >> Ack : 2
Correct Acknowledgement Recieved
A >> Ack : 3
Correct Acknowledgement Recieved
A >> Ack : 4
Correct Acknowledgement Recieved
A >> Ack : 5
Correct Acknowledgement Recieved
A >> Ack : 6
Correct Acknowledgement Recieved
A >> Ack : 7
Correct Acknowledgement Recieved
All the frames were sent successfully
```

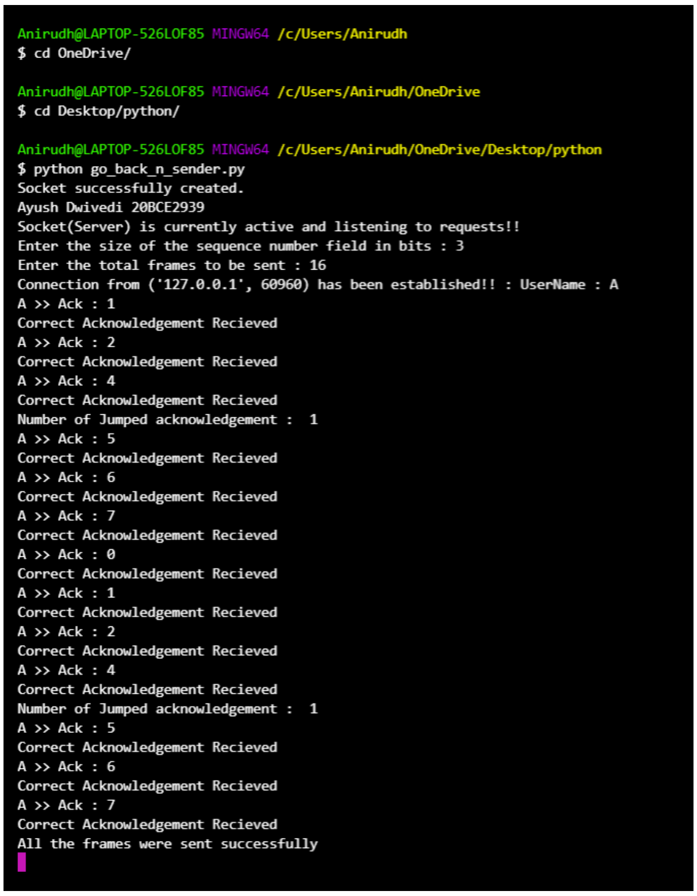
Using 16-bit:

Reciever_Server.py:



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

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python go_back_n_reciever.py
UserName : A
Server > Welcome to the server,Thanks for connecting!!
Recieved frame from Server: Frame : 0
Recieved frame from Server: Frame : 1
Acknowledgement Lost
Recieved frame from Server: Frame : 2
Recieved frame from Server: Frame : 3
Recieved frame from Server: Frame : 4
Recieved frame from Server: Frame : 5
Recieved frame from Server: Frame : 6
Recieved frame from Server: Frame : 7
Recieved frame from Server: Frame : 0
Recieved frame from Server: Frame : 1
Acknowledgement Lost
Recieved frame from Server: Frame : 2
Recieved frame from Server: Frame : 3
Recieved frame from Server: Frame : 4
Recieved frame from Server: Frame : 5
Recieved frame from Server: Frame : 6
Recieved frame from Server: Frame : 7
All the frames were recieved successfully

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

Sender_Client.py:

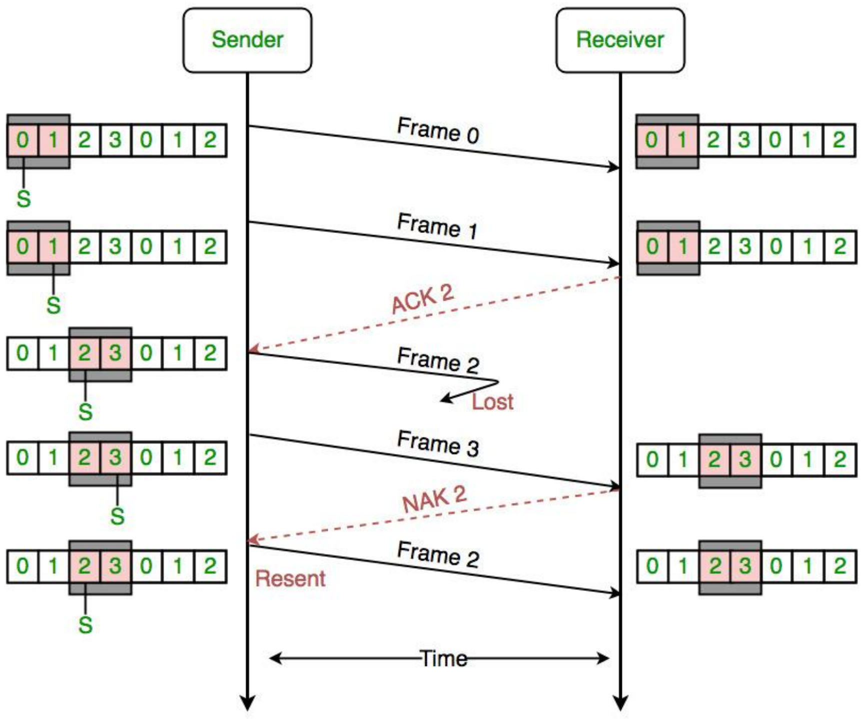


```
Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh
$ cd OneDrive/

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

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python go_back_n_sender.py
Socket successfully created.
Ayush Dwivedi 20BCE2939
Socket(Server) is currently active and listening to requests!!
Enter the size of the sequence number field in bits : 3
Enter the total frames to be sent : 16
Connection from ('127.0.0.1', 60960) has been established!! : UserName : A
A >> Ack : 1
Correct Acknowledgement Recieved
A >> Ack : 2
Correct Acknowledgement Recieved
A >> Ack : 4
Correct Acknowledgement Recieved
Number of Jumped acknowledgement :  1
A >> Ack : 5
Correct Acknowledgement Recieved
A >> Ack : 6
Correct Acknowledgement Recieved
A >> Ack : 7
Correct Acknowledgement Recieved
A >> Ack : 0
Correct Acknowledgement Recieved
A >> Ack : 1
Correct Acknowledgement Recieved
A >> Ack : 2
Correct Acknowledgement Recieved
A >> Ack : 4
Correct Acknowledgement Recieved
Number of Jumped acknowledgement :  1
A >> Ack : 5
Correct Acknowledgement Recieved
A >> Ack : 6
Correct Acknowledgement Recieved
A >> Ack : 7
Correct Acknowledgement Recieved
All the frames were sent successfully
```

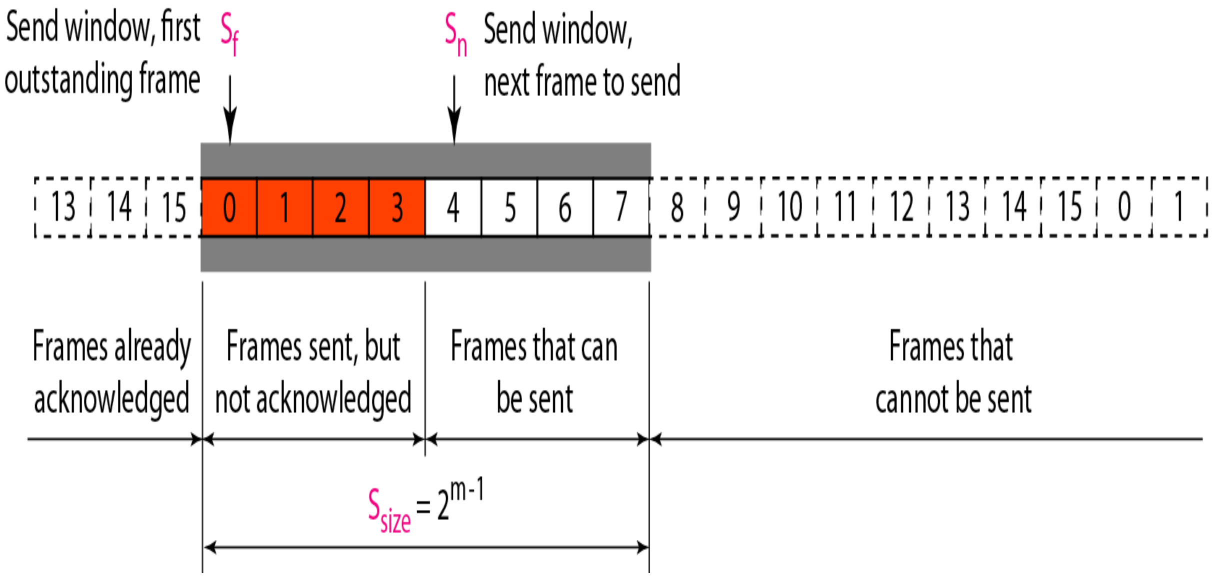
Selective Repeat :

This protocol(SRP) is mostly identical to GBN protocol, except that buffers are used and the receiver, and the sender, each maintains a window of size. SRP works better when the link is very unreliable. Because in this case, retransmission tends to happen more frequently, selectively retransmitting frames is more efficient than retransmitting all of them. SRP also requires full-duplex link. backward acknowledgments are also in progress.

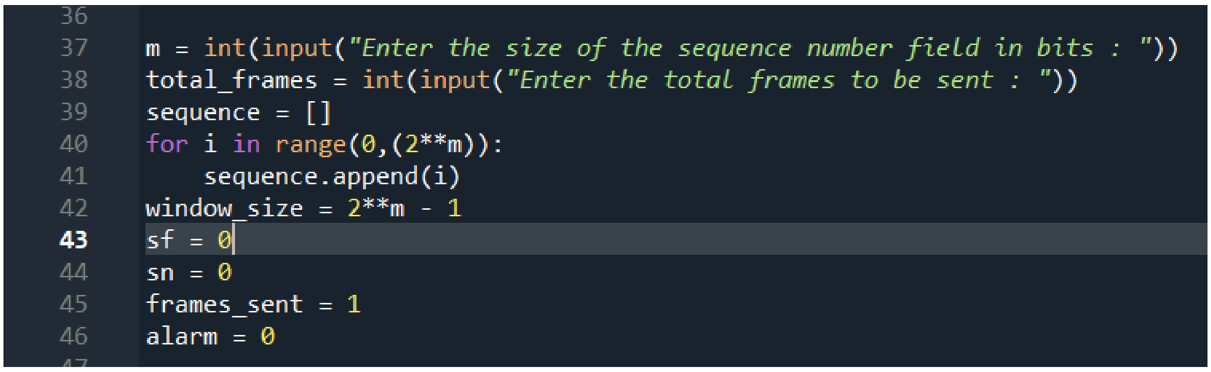
- Sender's Windows (Ws) = Receiver's Windows (Wr).
- Window size should be less than or equal to half the sequence number in SR protocol. This is to avoid packets being recognized incorrectly. If the size of the window is greater than half the sequence number space, then if an ACK is lost, the sender may send new packets that the receiver believes are retransmissions.
- Sender can transmit new packets as long as their number is with W of all unACKed packets.
- Sender retransmit un-ACKed packets after a timeout – Or upon a NAK if NAK is employed.
- Receiver ACKs all correct packets.
- Receiver stores correct packets until they can be delivered in order to the higher layer.
- In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of 2^m.



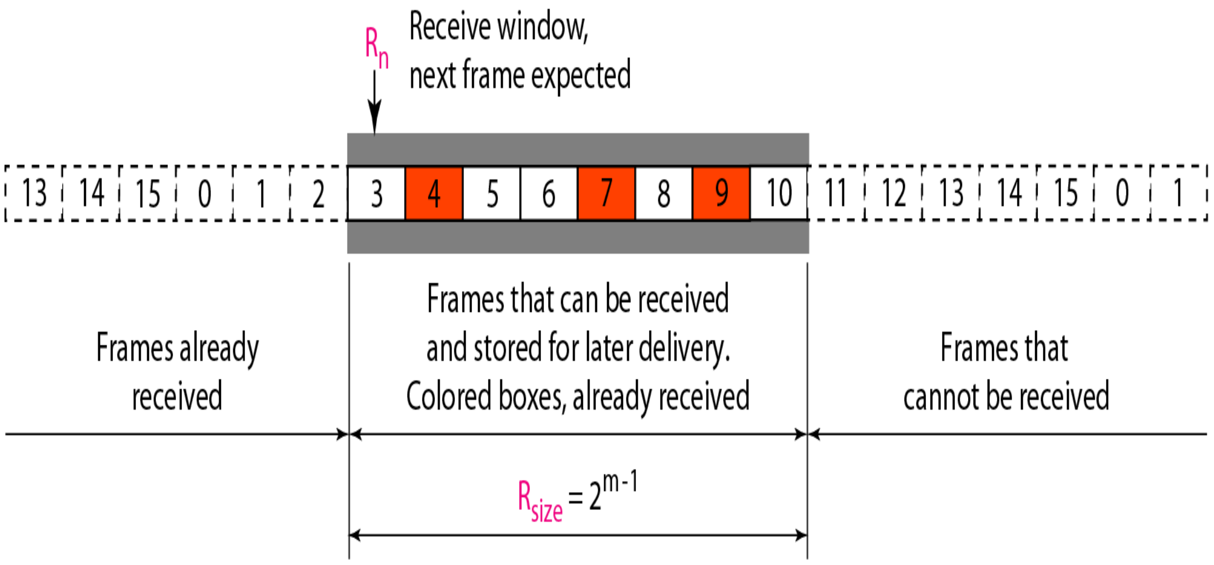
DATA STRUCTURES USED:


Send window, first outstanding frame
S_f
S_n
Send window, next frame to send
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already acknowledged
Frames sent, but not acknowledged
Frames that can be sent
Frames that cannot be sent
$S_{size} = 2^{m-1}$

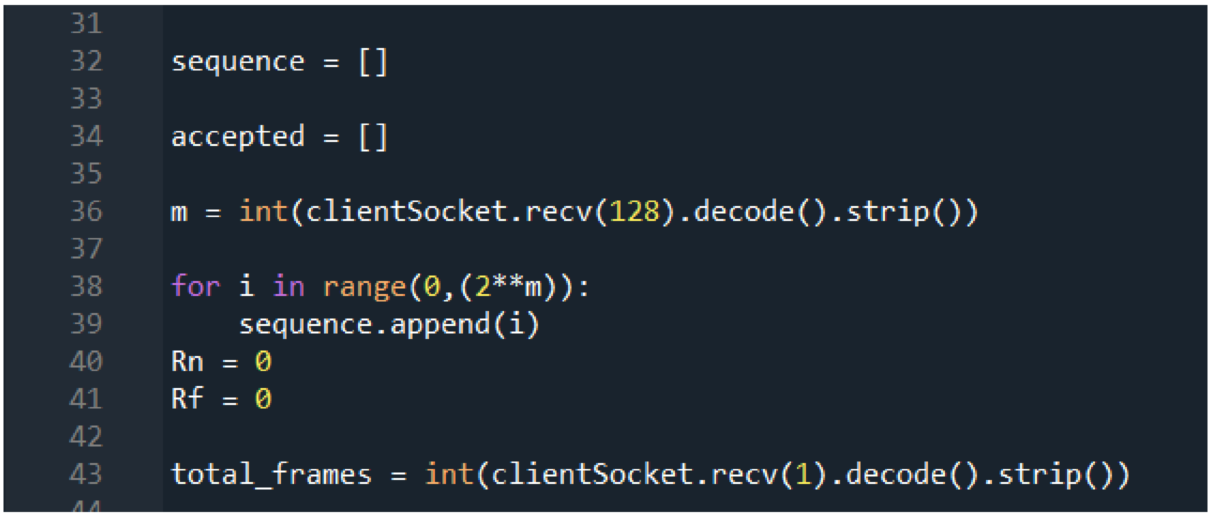
Sender Side:



```
m = int(input("Enter the size of the sequence number field in bits : "))
total_frames = int(input("Enter the total frames to be sent : "))
sequence = []
for i in range(0,(2**m)):
    sequence.append(i)
window_size = 2**m - 1
sf = 0
sn = 0
frames_sent = 1
alarm = 0
```


R_n
Receive window, next frame expected
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already received
Frames that can be received and stored for later delivery. Colored boxes, already received
Frames that cannot be received
$R_{size} = 2^{m-1}$

Receiver Side:


```
sequence = []

accepted = []

m = int(clientSocket.recv(128).decode().strip())

for i in range(0,(2**m)):
    sequence.append(i)
Rn = 0
Rf = 0

total_frames = int(clientSocket.recv(1).decode().strip())
```

CODE:

Go_Back_N_Reciever.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



# 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))
# Sending the username to the server
userName = my_userName.encode()
clientSocket.send(userName)

sequence = []

accepted = []

m = int(clientSocket.recv(128).decode().strip())

for i in range(0,(2**m)):
    sequence.append(i)
Rn = 0
Rf = 0

total_frames = int(clientSocket.recv(1).decode().strip())
```

```
for i in range(total_frames):
    accepted.append(0)

# recieving chunks of data from the server
def recieveData():
    flag = 0
    global Rn,Rf,total_frames,accepted
    # Recieving things infinitely
    while (total_frames!=0):
        if(flag == 0):# For the initial informative message
            msg = clientSocket.recv(128).decode()
            print(f"Server > {msg}")
            flag = 1
        else:# For the subsequent messages
            message = clientSocket.recv(9).decode()
            if(accepted[int(message[-1])]==1):
                print("Frame discarded : Frame ",(message[-1]))
            else:
                if((Rf-Rn)<=(2**(m-1)-1) and (Rf<=(2**m-1))):
                    if(int(message[-1])==Rn):
                        if(Rf>Rn):
                            accepted[Rn] = 1
                            temp = 0
                            for i in range(Rn,Rn+(2**(m-1))+1):
                                if(accepted[i] == 0):
                                    temp = i
                                    break
                            if(message):
```

```
            total_frames = total_frames - temp + Rn + 1
        print(f"Recieved frame from Server: {message}")
        print("Number of Jumped frame_acknowledgement : ",temp-Rn)
        Rn = temp
        if(int(message[-1])<(2**m-1)):
            ack_message = "Ack : " + str(sequence[Rn])
            clientSocket.send(ack_message.encode())
    else:
        if(message):
            total_frames = total_frames - 1
        accepted[Rf] = 1
        Rf = Rf + 1
        Rn = Rn + 1
        temp_flag_resent = 0
        if(total_frames!=0):
            if(int(message[-1])<(2**m-1)):
                # if(Rn!=2 and Rn!=3):
                # if(temp_flag_resent==0 and Rn==2):
                    # temp_flag_resent = 1
                ack_message = "Ack : " + str(sequence[Rn])
                clientSocket.send(ack_message.encode())
            print(f"Recieved frame from Server: {message}")
else:
    if(int(message[-1])>Rn):
        accepted[int(message[-1])] = 1
        if(int(message[-1])>Rf):
            Rf = int(message[-1])
        if(message):
```

```
                    total_frames = total_frames - 1
                print(f"Recieved frame from Server: {message}")
                if(int(message[-1])<(2**m-1)):
                    ack_message = "Nak : " + str(sequence[Rn])
                    clientSocket.send(ack_message.encode())
                print("Negative Acknowldgement Sent of frame : ",sequence[Rn])

    else:
        print("All the frames were recieved successfully")



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

Selective_Repeat_Sender.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)
```

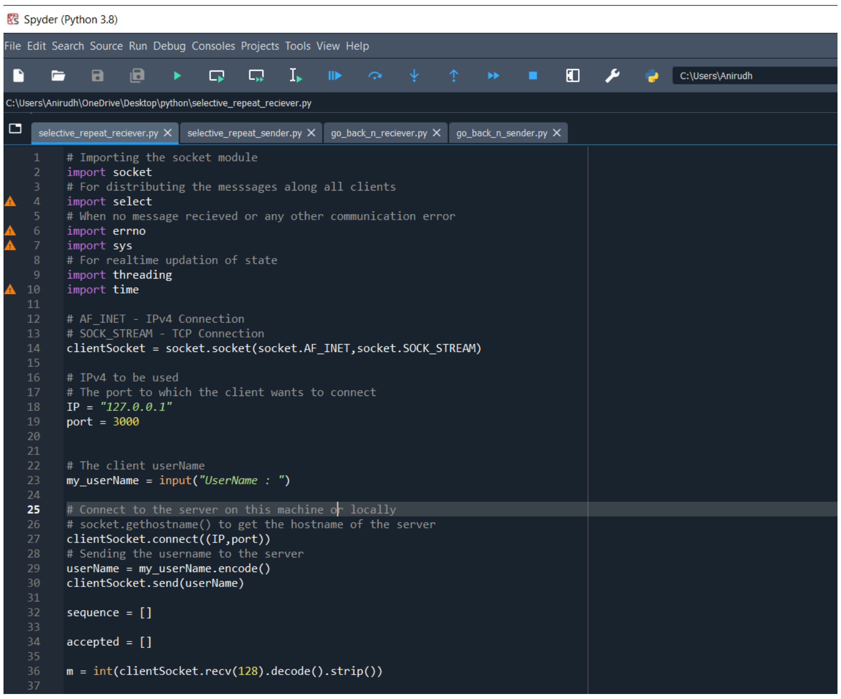
```
print("Socket successfully created.")


# 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


# 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 = {}


m = int(input("Enter the size of the sequence number field in bits : "))

total_frames = int(input("Enter the total frames to be sent : "))

sequence = []

for i in range(0,(2**m)):
    sequence.append(i)
sf = 0
sn = 0
frames_sent = 1
alarm = 0

sent = []
for i in range(total_frames):
    sent.append(0)

# A function to recieve messages from the clients connected over the network
def recieveMessage(clientSocket):
    try:
        return {"Data" : clientSocket.recv(7)}
    except:
        return False

def recieve_ack(notifiedSocket):
    global sf,sn,frames_sent
    while True:
        if(sf<=sn):
            ack_message = recieveMessage(notifiedSocket)
            if(ack_message['Data'].decode()):
                if(ack_message['Data'].decode()[0]=='A'):
                    user = clients[notifiedSocket]
                    print(f"{user['Data'].decode()} >> {ack_message['Data'].decode()}")
                    if(int(ack_message['Data'].decode()[-1])>=(sf+1) or
int(ack_message['Data'].decode()[-1])==0):
```

```
                sent[int(ack_message['Data'].decode()[-1]) - 1] = 1
                print("Correct Acknowledgement Recieved")
                difference = (int(ack_message['Data'].decode()[-1])-(sf))
                if(difference>=0):
                    sf = sf + difference
                    frames_sent = frames_sent + difference
                    if(sf>=(2**m-1)):
                        sf = sf - (2**m)
                else:
                    difference = difference + (2**m)
                    sf = sf + difference
                    frames_sent = frames_sent + difference
                    if(sf>=(2**m-1)):
                        sf = sf - (2**m)
                if(difference > 1):
                    print("Number of Jumped acknowledgement : ",difference-1)
        else:
            user = clients[notifiedSocket]
            print(f"{user['Data'].decode()} >> {ack_message['Data'].decode()}")
            message = "Frame : " + str(sequence[int(ack_message['Data'].decode()[-
1])])
            notifiedSocket.send(message.encode())


def timer():
    global alarm,sf,sn
    while True:
        time.sleep(1.5)
        if(sf!=sn):
            alarm = (alarm + 1) % 2

thread3 = threading.Thread(target = timer, args = ())
thread3.start()

# Making a thread for every user connected to the server
def clientThread(notifiedSocket):
    global sf,sn,frames_sent,total_frames,alarm
    temp_flag = 0
    while (frames_sent!=total_frames):
        if(alarm==1):
            print("Timer Expired")
            print("Resending Frames")
            for i in range(sf,sf+2**(m-1)):
                if(sent[i] != 1):
                    message = "Frame : " + str(sequence[sn])
                    notifiedSocket.send(message.encode())
            alarm = 0
        if((sn-sf)<=(2**(m-1)-1) and (sn<=(2**m-1))):
            time.sleep(1)
            message = "Frame : " + str(sequence[sn])
            if(sn==3 and temp_flag==0):
                print("Not sending Third Frame for the first time :")
                temp_flag = 1
            else:
                notifiedSocket.send(message.encode())
            sn = sn + 1
            if(sn>=(2**m)):
                sn = sn - (2**m)
            # The part to do if a client leaves the connection
            if message is False:
                print("The message is False")
                print(f"Closed Connection from {clients[notifiedSocket]['Data'].decode()}")
                socketsList.remove(notifiedSocket)
                del clients[notifiedSocket]
                break
    else:
        sf = sf + 1
        print("All the frames were sent successfully")

# 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()}")

    clientSocket.send(str(m).encode())
```

```
# Sending the count of total frames
clientSocket.send(str(total_frames).encode())

msg = "Welcome to the server,Thanks for connecting!!"

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


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

thread2= threading.Thread(target = recieve_ack, args = (clientSocket,))
thread2.start()
```

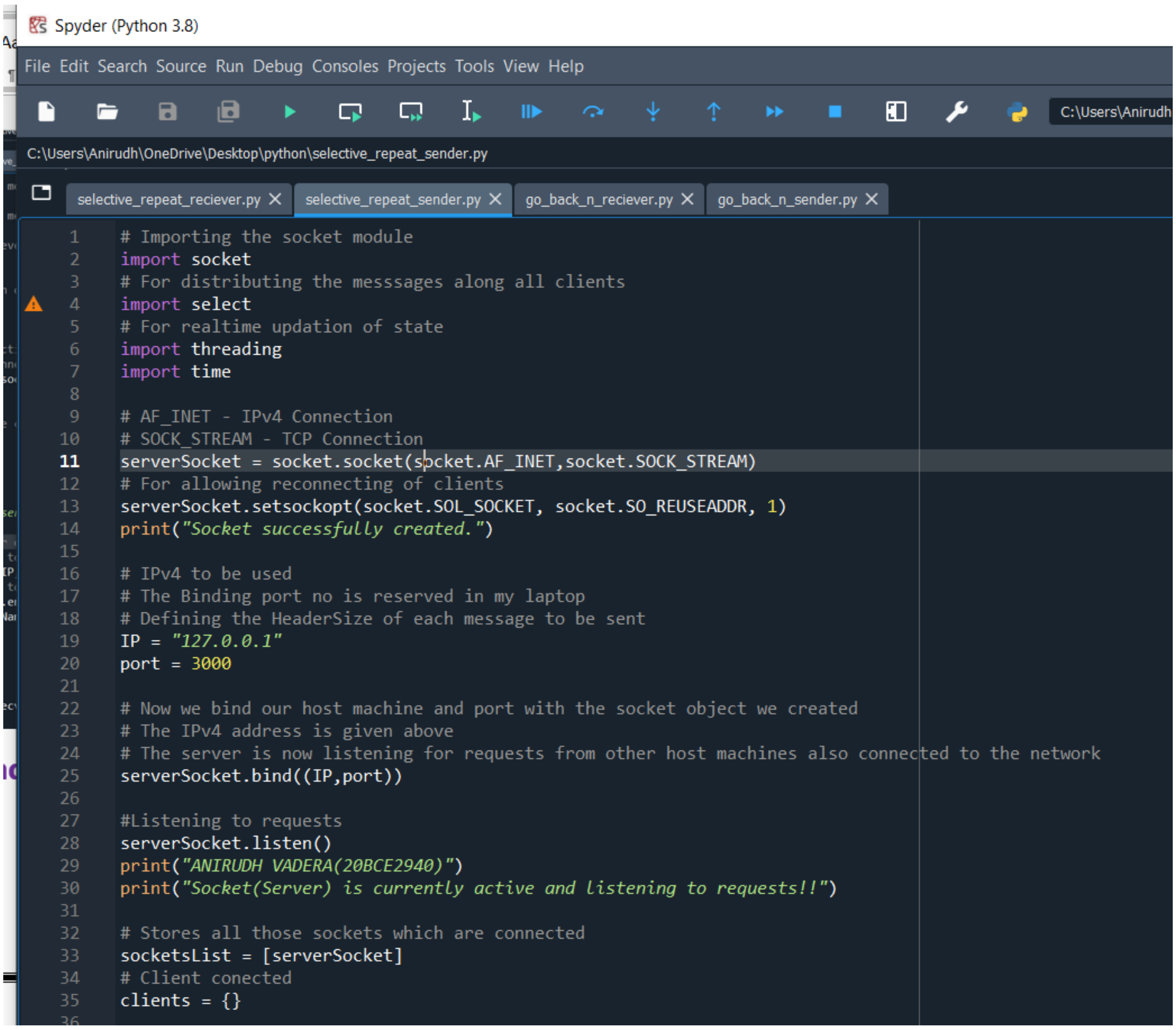
CODE SNIPPETS:

Selective_Repeat_Receiver.py:


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Anirudh
C:\Users\Anirudh\OneDrive\Desktop\python\selective_repeat_reciever.py
selective_repeat_reciever.py
selective_repeat_sender.py
go_back_n_reciever.py
go_back_n_sender.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


# 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))
# Sending the username to the server
userName = my_userName.encode()
clientSocket.send(userName)

sequence = []

accepted = []

m = int(clientSocket.recv(128).decode().strip())

```

Selective_Repeat_Sender.py:


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Anirudh\OneDrive\Desktop\python\selective_repeat_sender.py
selective_repeat_reciever.py
selective_repeat_sender.py
go_back_n_reciever.py
go_back_n_sender.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

# 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 = {}
```

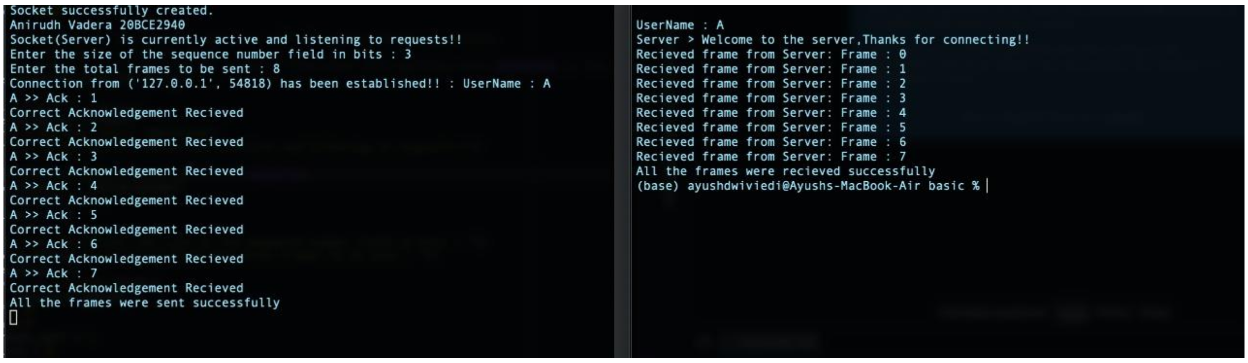
OUTPUT:

RECIEVER-SERVER :: SENDER-CLIENT

CASE 1: IDEAL CASE:

Number of Bits = 3

Total Frames = 8

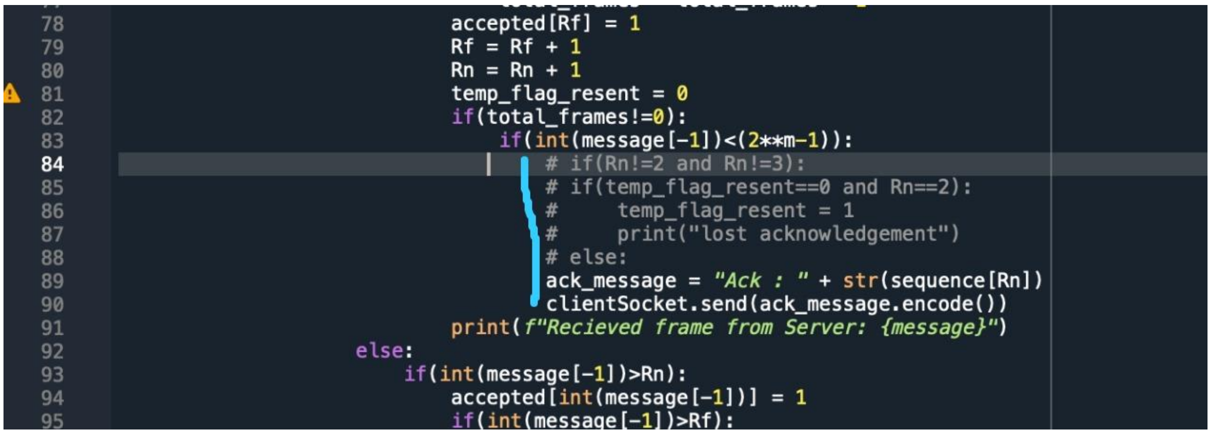


```
Socket successfully created.
Anirudh Vadera 20BCE2940
Socket(Server) is currently active and listening to requests!!
Enter the size of the sequence number field in bits : 3
Enter the total frames to be sent : 8
Connection from ('127.0.0.1', 54818) has been established!! : UserName : A
A >> Ack : 1
Correct Acknowledgement Recieved
A >> Ack : 2
Correct Acknowledgement Recieved
A >> Ack : 3
Correct Acknowledgement Recieved
A >> Ack : 4
Correct Acknowledgement Recieved
A >> Ack : 5
Correct Acknowledgement Recieved
A >> Ack : 6
Correct Acknowledgement Recieved
A >> Ack : 7
Correct Acknowledgement Recieved
All the frames were sent successfully
```

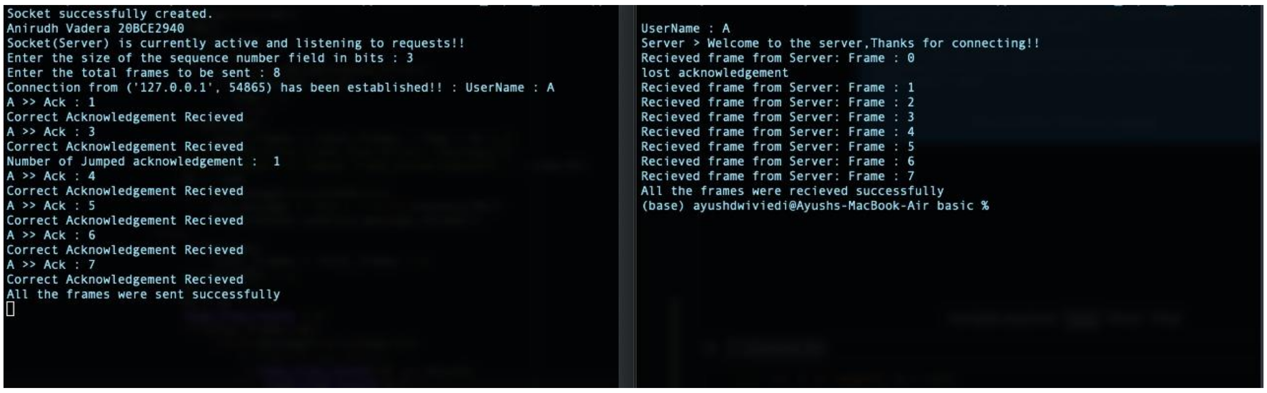
```
UserName : A
Server > Welcome to the server,Thanks for connecting!!
Recieved frame from Server: Frame : 0
Recieved frame from Server: Frame : 1
Recieved frame from Server: Frame : 2
Recieved frame from Server: Frame : 3
Recieved frame from Server: Frame : 4
Recieved frame from Server: Frame : 5
Recieved frame from Server: Frame : 6
Recieved frame from Server: Frame : 7
All the frames were recieved successfully
(base) ayushdwiviedi@Ayushs-MacBook-Air basic % |
```

Case 2: Acknowledgement Lost

Code:



```python
                        accepted[Rf] = 1
                        Rf = Rf + 1
                        Rn = Rn + 1
                        temp_flag_resent = 0
                        if(total_frames!=0):
                            if(int(message[-1])<(2**m-1)):
                                # if(Rn!=2 and Rn!=3):
                                # if(temp_flag_resent==0 and Rn==2):
                                #     temp_flag_resent = 1
                                #     print("lost acknowledgement")
                                # else:
                                ack_message = "Ack : " + str(sequence[Rn])
                                clientSocket.send(ack_message.encode())
                        print(f"Recieved frame from Server: {message}")
                else:
                    if(int(message[-1])>Rn):
                        accepted[int(message[-1])] = 1
                        if(int(message[-1])>Rf):
```

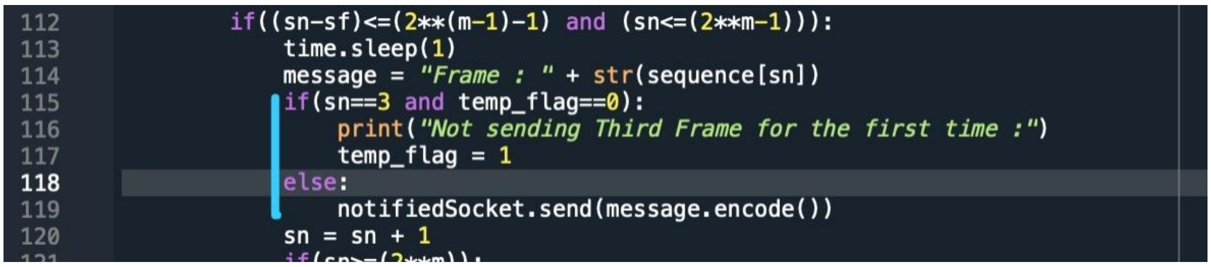


```
Socket successfully created.
Anirudh Vadera 20BCE2940
Socket(Server) is currently active and listening to requests!!
Enter the size of the sequence number field in bits : 3
Enter the total frames to be sent : 8
Connection from ('127.0.0.1', 54865) has been established!! : UserName : A
A >> Ack : 1
Correct Acknowledgement Recieved
A >> Ack : 3
Correct Acknowledgement Recieved
Number of Jumped acknowledgement :  1
A >> Ack : 4
Correct Acknowledgement Recieved
A >> Ack : 5
Correct Acknowledgement Recieved
A >> Ack : 6
Correct Acknowledgement Recieved
A >> Ack : 7
Correct Acknowledgement Recieved
All the frames were sent successfully
```

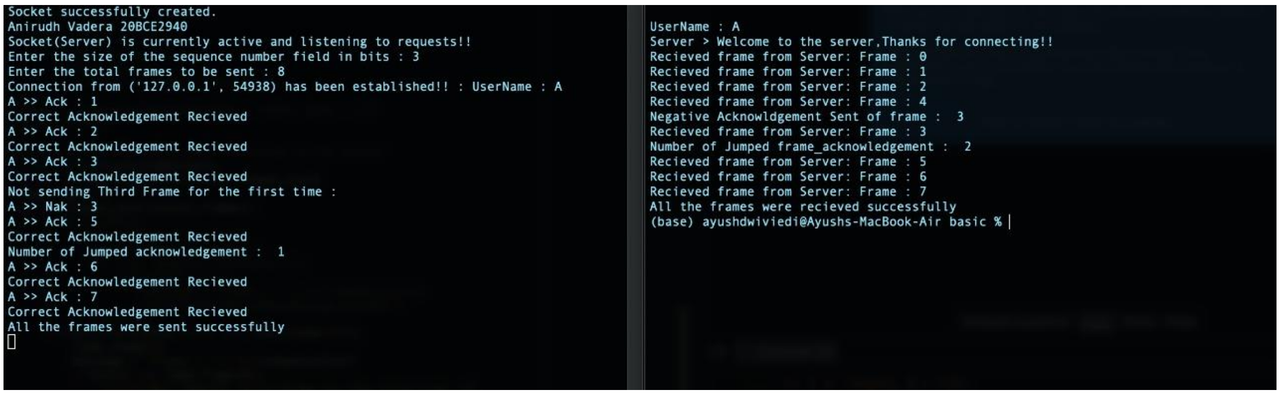
```
UserName : A
Server > Welcome to the server,Thanks for connecting!!
Recieved frame from Server: Frame : 0
lost acknowledgement
Recieved frame from Server: Frame : 1
Recieved frame from Server: Frame : 2
Recieved frame from Server: Frame : 3
Recieved frame from Server: Frame : 4
Recieved frame from Server: Frame : 5
Recieved frame from Server: Frame : 6
Recieved frame from Server: Frame : 7
All the frames were recieved successfully
(base) ayushdwiviedi@Ayushs-MacBook-Air basic %
```

Case 3: Frame Lost

Code



```python
        if((sn-sf)<=(2**(m-1)-1) and (sn<=(2**m-1))):
            time.sleep(1)
            message = "Frame : " + str(sequence[sn])
            if(sn==3 and temp_flag==0):
                print("Not sending Third Frame for the first time :")
                temp_flag = 1
            else:
                notifiedSocket.send(message.encode())
            sn = sn + 1
            if(sn>=(2**m)):
```



```
Socket successfully created.
Anirudh Vadera 20BCE2940
Socket(Server) is currently active and listening to requests!!
Enter the size of the sequence number field in bits : 3
Enter the total frames to be sent : 8
Connection from ('127.0.0.1', 54938) has been established!! : UserName : A
A >> Ack : 1
Correct Acknowledgement Recieved
A >> Ack : 2
Correct Acknowledgement Recieved
A >> Ack : 3
Correct Acknowledgement Recieved
Not sending Third Frame for the first time :
A >> Nak : 3
A >> Ack : 5
Correct Acknowledgement Recieved
Number of Jumped acknowledgement :  1
A >> Ack : 6
Correct Acknowledgement Recieved
A >> Ack : 7
Correct Acknowledgement Recieved
All the frames were sent successfully
```

```
UserName : A
Server > Welcome to the server,Thanks for connecting!!
Recieved frame from Server: Frame : 0
Recieved frame from Server: Frame : 1
Recieved frame from Server: Frame : 2
Recieved frame from Server: Frame : 4
Negative Acknowldgement Sent of frame :  3
Recieved frame from Server: Frame : 3
Number of Jumped frame_acknowledgement :  2
Recieved frame from Server: Frame : 5
Recieved frame from Server: Frame : 6
Recieved frame from Server: Frame : 7
All the frames were recieved successfully
(base) ayushdwiviedi@Ayushs-MacBook-Air basic %
```