Name: Tanmoy Sarkar
Roll No: 002010501020
Class: BCSE III
Assignment No: 2
Subject: Computer Network
Group: A1

Problem Statement – Implement three data link layer protocols, Stop and Wait, Go Back N Sliding Window and Selective Repeat Sliding Window for flow control.

Sender, Receiver and Channel all are independent processes. There may be multiple Transmitter and Receiver processes, but only one Channel process. The channel process introduces random delay and/or bit error while transferring frames. Define your own frame format or you may use IEEE 802.3 Ethernet frame format.
Hints: Some points you may consider in your design.
Following functions may be required in Sender.
Send: This function, invoked every time slot at the sender, decides if the sender should (1) do nothing, (2) retransmit the previous data frame due to a timeout, or (3) send a new data frame. Also, you have to consider current network time measure in time slots.
Recv_Ack: This function is invoked whenever an ACK packet is received. Need to consider network time when the ACK was received, ack_num and timestamp are the sender's sequence number and timestamp that were echoed in the ACK. This function must call the timeout function.
Timeout: This function should be called by ACK method to compute the most recent data packet's round-trip time and then re-compute the value of timeout.
Following functions may be required in Receiver.
Recv: This function at the receiver is invoked upon receiving a data frame from the sender.
Send_Ack: This function is required to build the ACK and transmit.
Sliding window: The sliding window protocols (Go-Back-N and Selective Repeat) extend the stop-and-wait protocol by allowing the sender to have multiple frames outstanding (i.e., unacknowledged) at any given time. The maximum number of unacknowledged frames at the sender cannot exceed its "window size". Upon receiving a frame, the receiver sends an ACK for the frame's sequence number. The receiver then buffers the received frames and delivers them in sequence number order to the application.
Performance metrics: Receiver Throughput (packets per time slot), RTT, bandwidth-delay product, utilization percentage.

PRE-REQUISITES

As this protocol required communication between the client and server and also required a channel to simulate injecting error like real world, we need to build a program based on socket to make that easy to implement the protocols given in the assignment.

SocketServer - *Capable of handling multiple clients and support messages passing to one client to another by maintaining a connection pool*

```python
import socket
from _thread import *

# Purpose
# 1. Manage connection pool
# 2. Manage client requests
# 3. Bi-directional communication
# -> Sender can send data to Receiver
# -> Receiver can send data to Sender

class SocketServer:

    clientConnectionInstances = {}
    senderReceiver = {}
    # map of senderIp:senderPort to receiverIp:receiverPort
    # map of receiverIp:receiverPort to senderIp:senderPort

    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.socket = socket.socket()

    def __repr__(self) -> str:
        return f"SocketServer({self.host}, {self.port})"

    def start(self):
        try:
            self.socket.bind((self.host, self.port))
        except socket.error as e:
            print(str(e))
        print('Server is online ...')
        self.socket.listen(5)

    def startAcceptConnections(self):
        while True:
            client, address = self.socket.accept()
            # Store client connection instance in dictionary
            SocketServer.clientConnectionInstances[f"{address[0]}:{address[1]}"] =
client
            print('Connected to: ' + address[0] + ':' + str(address[1]))
            # Start a new thread to handle client requests
            start_new_thread(SocketServer.handleClient, (address[0],
str(address[1])))

    @staticmethod
    def handleClient(clientIp, clientPort):
        address = f"{clientIp}:{clientPort}"
        # Get client connection instance from dictionary
        client = SocketServer.clientConnectionInstances[address]

        # Start listening for client requests
        while True:
            try:
                data = client.recv(1024)
                # Check whether client sent data
                if not data:
                    break
                data = data.decode('utf-8')
                # Check for special command to connect to receiver
                if str(data).startswith("connect:"):
                    splittedData = data.split(":")
                    if(len(splittedData) == 3):
                        receiverAddress = f"{splittedData[1]}:{splittedData[2]}"
                        # Check whether receiver address found
                        if receiverAddress in SocketServer.clientConnectionInstances:
                            # Check if receiver address is found in senderReceiver
map
                            if receiverAddress in SocketServer.senderReceiver or
address in SocketServer.senderReceiver:
                                print("Already connected to receiver")
                            else:
                                # Store in sender receiver map
                                SocketServer.senderReceiver[address] =
receiverAddress
                                SocketServer.senderReceiver[receiverAddress] =
address
                                print(f"Setup done ! Sender {address} is connected to
receiver {receiverAddress}")
                        else:
                            print(f"Receiver {receiverAddress} not found")
                    else:
                        continue
                # Else consider as normal data
                else:
                    # Get receiver address from senderReceiver map
                    receiverAddress = None
                    if address in SocketServer.senderReceiver:
                        receiverAddress = SocketServer.senderReceiver[address]
                    else:
                        print(f"No receiver found for sender {address}")
                        continue
                    # Check whether receiver instance found
                    receiverInstance =
SocketServer.clientConnectionInstances[receiverAddress]
                    if receiverInstance is None:
                        print(f"Receiver {receiverAddress} not found")
                        continue
                    # Send data to receiver

receiverInstance.sendall(str.encode(SocketServer.modifyData(data)))
                    print(f"{address} ---> {receiverAddress} --> {data}")

            except (BrokenPipeError, ConnectionResetError, OSError):
                    print(f"Receiver {receiverAddress} is offline")
                    break
            except Exception as e:
                print(str(e))
                break

        client.close()

        if address in SocketServer.clientConnectionInstances:
            del SocketServer.clientConnectionInstances[address]

        if address in SocketServer.senderReceiver:
            # Get receiver address from senderReceiver map
            receiverAddress = SocketServer.senderReceiver[address]
            # Get receiver instance from clientConnectionInstances map
            if receiverAddress in  SocketServer.clientConnectionInstances:
                receiverInstance =
SocketServer.clientConnectionInstances[receiverAddress]
                receiverInstance.send(str.encode(f"disconnect:"))
                # Close receiver instance
                receiverInstance.close()
                del SocketServer.clientConnectionInstances[receiverAddress]
                del SocketServer.senderReceiver[receiverAddress]
            # Delete own address from senderReceiver map
            del SocketServer.senderReceiver[address]

        print(f"Client {address} has disconnected")
```

```
    @staticmethod
    def modifyData(data):
        # An method to be overridden by child class
        return data

    def closeAllConnections(self):
        self.socket.close()
        print("Server is offline")

# Example Code
# tmp = SocketServer(host='127.0.0.1', port=8081)
# tmp.start()
# tmp.startAcceptConnections()
```

Client - Base class to implement a client that can communicate with the server easily and send and receive messages to/from another client. It has some abstract methods which we need to implement that's different for each protocol.

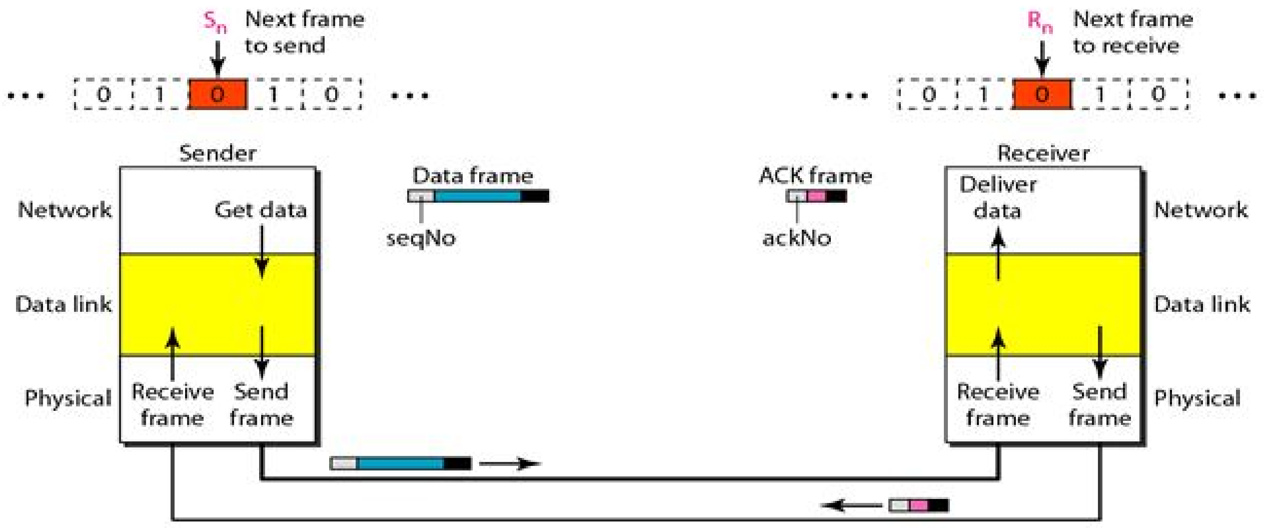
```
from abc import abstractmethod
import socket
import threading
import time


class Client:
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.killed = False
        self.condition = threading.Condition()

    def connectToServer(self):
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            self.sock.connect((self.host, self.port))
            print('Connected to server')
        except socket.error as e:
            print(str(e))

    def startProcess(self):
        self.setupBeforeProcess()
        self.senderThread = threading.Thread(target=self.sendData)
        self.receiverThread = threading.Thread(target=self.receiveData)
        self.receiverThread.start()
        self.senderThread.start()

    @abstractmethod
    def sendData(self):
        raise NotImplementedError

    @abstractmethod
    def receiveData(self):
        raise NotImplementedError

    @abstractmethod
    def setupBeforeProcess(self):
        raise NotImplementedError

    def closeConnection(self):
        if self.killed : return
        self.killed = True
        print('Closing connection')
        self.condition.acquire()
        self.condition.notifyAll()
        self.condition.release()

        try:
            self.sock.close()
        except socket.error as e:
            print(str(e))
        print('Connection closed')


# client = Client('127.0.0.1',8081)
# client.connectToServer()
# client.closeConnection()
```

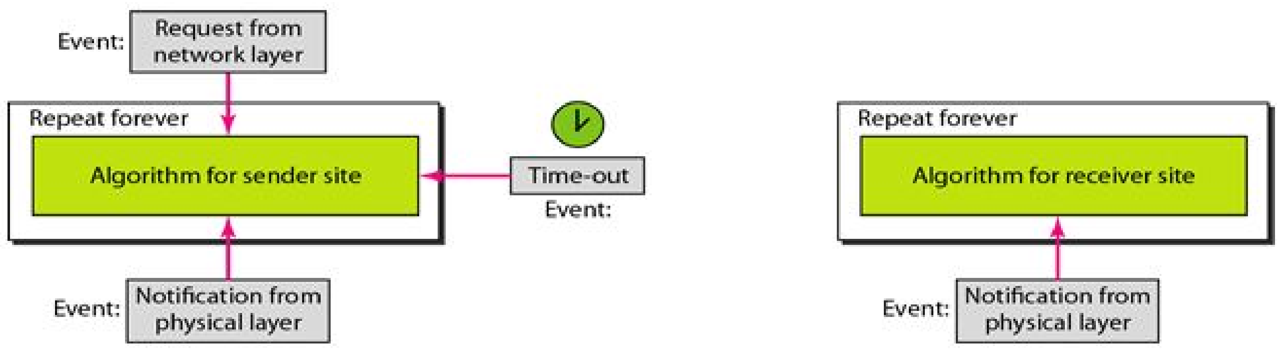
Channel - *it inherited the socket server class and added the error injecting and random delay logics in it.*

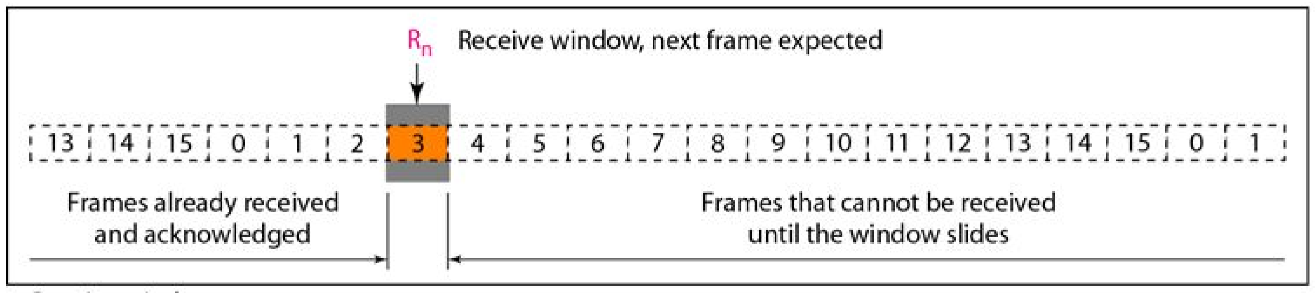
```
import time
from clientServer.socketServer import SocketServer
import random

class Channel(SocketServer):
    errorCount = 0
    def __init__(self, host, port):
        super().__init__(host, port)
        # Update modifyData static method of  SocketServer with  Channel's
        SocketServer.modifyData = Channel.modifyData

    @staticmethod
    def modifyData(data):
        # return data
        # return data
        if data in ["disconnect:"]:
            return data
        return Channel.injectErrorInData(data, loopC=1)

    @staticmethod
    def injectErrorInData(data, loopC=5):
        # time.sleep(random.random())
        olddata = data
        for _ in range(loopC):
            random_bit_location = random.randint(0, len(data)-1)
            should_inject_error = Channel.errorCount % 17 == 0
            Channel.errorCount += 1
            if should_inject_error:
                data = data[:random_bit_location] +  ("0" if
data[random_bit_location] == "1" else "1") + data[random_bit_location+1:]
        if olddata != data:
            print("Injected Error : "+olddata+" ---> "+data)
        return data




if __name__ == "__main__":
    server = Channel(host='127.0.0.1', port=8081)
    try:
        print("Socket Server[Channel] is starting....")
        server.start()
        server.startAcceptConnections()
    except KeyboardInterrupt:
        server.closeAllConnections()
        exit()
    except Exception as e:
        print(str(e))
        server.closeAllConnections()
        exit()
```

Error Detection Modules: We are using the error modules developed in the previous assignment for this assignment, for that reason we are going into detail about those.

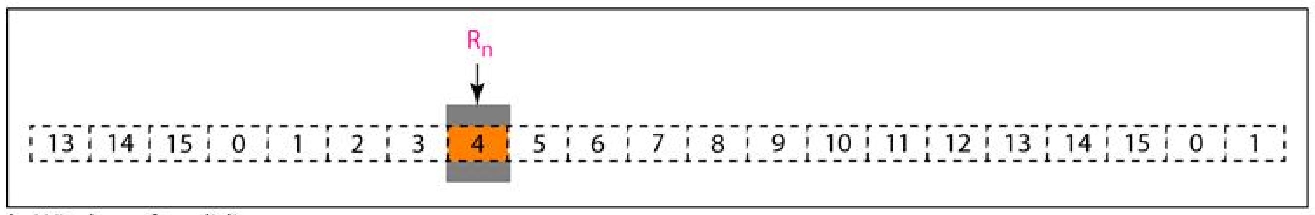
DESIGN OF STOP AND WAIT PROTOCOL




Event: Request from network layer
Repeat forever
Algorithm for sender site
Time-out Event:
Event: Notification from physical layer
Repeat forever
Algorithm for receiver site
Event: Notification from physical layer


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

Code -

Sender side

```
import socket
import threading
```

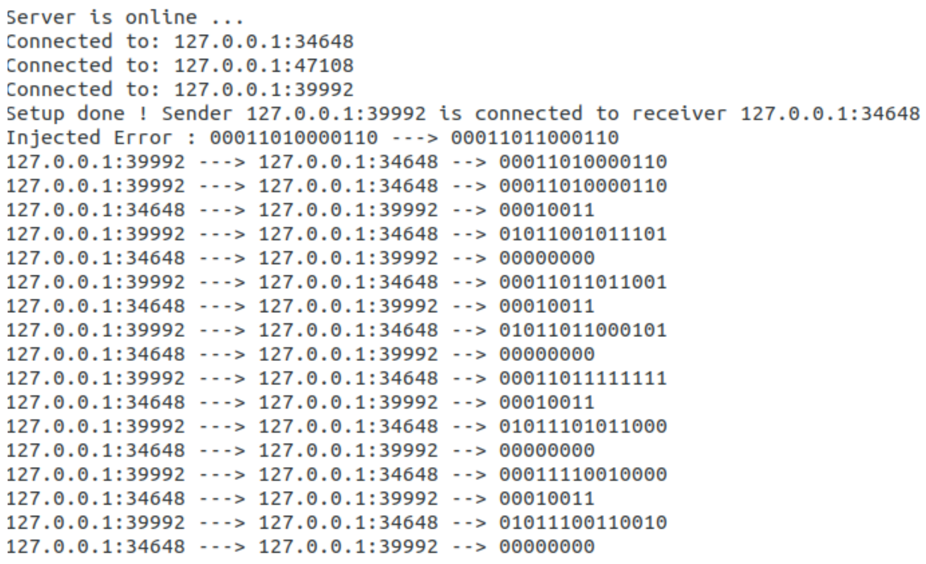
```
from time import sleep

from helpers import buildFrames, decodeData, encodeData


# FRAME FORMAT:
# For a frame with length of 8, the format is:
# [sn][data][parity]
# [2 ][  8 ][  4   ]

# Ack frame format:
# [sn][parity]
# [ 4 ][  4   ]

class Sender:
    def __init__(self, host, port, timeout):
        self.host = host
        self.port = port
        self.timeout = timeout
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((self.host, self.port))
        self.ackReceivedEvent = threading.Event()
        self.ackReceivedEvent.clear()

        self.data = []
        self.frameIndex = 0
        self.sn = 0
        self.snmax = 1

        # Configure the receiver
        receiver_ip = input("Enter receiver ip: ")
        if len(receiver_ip.strip()) == 0:
            receiver_ip = "127.0.0.1"
        receiver_port = input("Enter receiver port: ")

self.sock.sendall(str.encode(f'connect:{receiver_ip}:{receiver_port}'))
        sleep(1)

    def startProcess(self):
        self.senderThread = threading.Thread(target=self.send)
        self.receiverACKThread = threading.Thread(target=self.recvAck)
        self.receiverACKThread.start()
        self.senderThread.start()
    def send(self):
        while True:
            # Check frames
            if self.frameIndex >= len(self.data):
                print("No more frames to send")
                break
            # Prepare fram e-- by taking the sn and the element of window
            frame =
encodeData(str(bin(self.sn)[2:]).zfill(2)+self.data[self.frameIndex])
            print("[SEND] new frame ", frame, " with sn ", self.sn)
            # Increase seq no
            self.increaseSn()
            # Icrease frame index
            self.frameIndex += 1
            # Send a frame
            self.sock.sendall(str.encode(frame))
            # Wait for ack with a timeout
            self.ackReceivedEvent.clear()
            isNotified = self.ackReceivedEvent.wait(timeout=self.timeout)
            if not isNotified:
                print("[RESEND] resending frame ", frame, " with sn ",
self.sn)
                self.frameIndex -= 1
                self.decreaseSn()

            # If timeout, resend the frame -- run the loop again


    def recvAck(self):
        while True:
            # Listen for data
            data = self.sock.recv(1024)
            data = data.decode()
            # If ack is valid
            decodedData = decodeData(data)
            if decodedData[0]:
                seqNo = int(decodedData[1], 2)
                # If ack sn == current sn, just emit ackreceived event
                if seqNo != self.sn:
                    # Else ack sn != current sn, set sn to ack sn , emit
ackreceived event
                    self.sn = seqNo
                    # Set frame index to previous frame index
                    self.frameIndex = max(self.frameIndex-1, 0)
                    print("[ACK] ack received with sn ", seqNo)
                else:
                    print("[ACK] ack discarded as rn not macthed")
                # Emit ackreceived event
                self.ackReceivedEvent.set()
            else:
                print("[DISCARD] discarding ACK due to error")


    def increaseSn(self):
        self.sn += 1
        if self.sn > self.snmax:
            self.sn = 0

    def decreaseSn(self):
        self.sn -= 1
        self.sn = max(self.sn, 0)

    def pushData(self, data):
        frames = buildFrames(data, frameSize=8)
        for i in frames:
            self.data.append(i)



sender = Sender('127.0.0.1', 8081, 4)
sender.pushData("01101000011001010110110101101100011011110111010101110010111
0011")

sender.startProcess()
```

Receiver Side-

```
import socket
import threading

from helpers import decodeData, generateACK


class Receiver:
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((self.host, self.port))
        self.frameReceived = threading.Event()
        self.frameReceived.clear()

        self.data = []
        self.rn = 0
        self.rnmax = 1


    def startProcess(self):
        self.sendACKThread = threading.Thread(target=self.sendACK)
        self.receiverThread = threading.Thread(target=self.recv)
        self.receiverThread.start()
        self.sendACKThread.start()


    def recv(self):
        while True:
            # Receieve data
            data = self.sock.recv(1024)
            if data == "disconnect:" : break
            data = data.decode()
            # Decode data
            decodedData = decodeData(data)
            # Check if valid
            if decodedData[0]:
                # Extract frame and seq no
                data = decodedData[1]
                seqNo = int(data[:2], 2)
                frame = data[2:]
                # If seq no == rn, save data and send ack for next frame
                if seqNo == self.rn:
                    self.data.append(frame)
                    self.increaseRn()
                    self.frameReceived.set()
                    self.printData()
                    print("[ACCEPT] Frame received ")
                # If seq no != rn, discard and send ack fagain
                else:
                    print("[DISCARD] seqNo not matched to rn")
                    self.frameReceived.set()
            else:
                print("[DISCARD] frame due to error")
    def sendACK(self):
        while True:
            # Wait for frame received event
            self.frameReceived.wait()
            print("[ACK] Sending ACK for frame ", self.rn)
            # Send ack for the frame
            self.sock.sendall(str.encode(generateACK(self.rn,
with_parity=True)))
            # Clear frame received event
            self.frameReceived.clear()

    def increaseRn(self):
        self.rn += 1
        if self.rn > self.rnmax:
            self.rn = 0

    def printData(self):
        print("Data : ", end="", flush=True)
        for i in self.data:
            print(i, end="", flush=True)
        print()

receiver = Receiver("127.0.0.1", 8081)
receiver.startProcess()
```

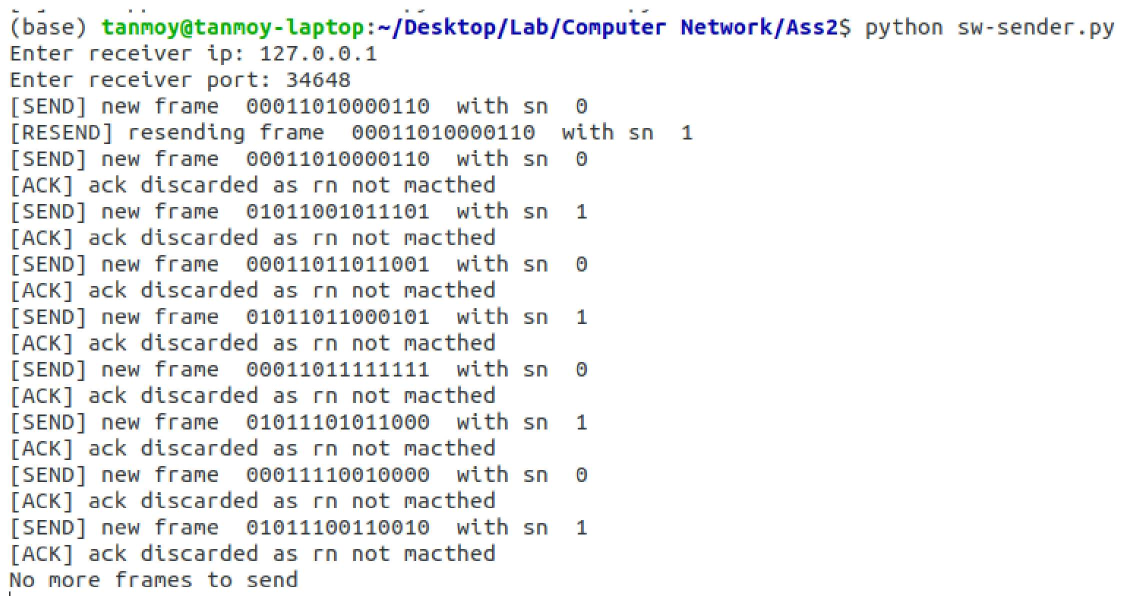
Output -

Channel -



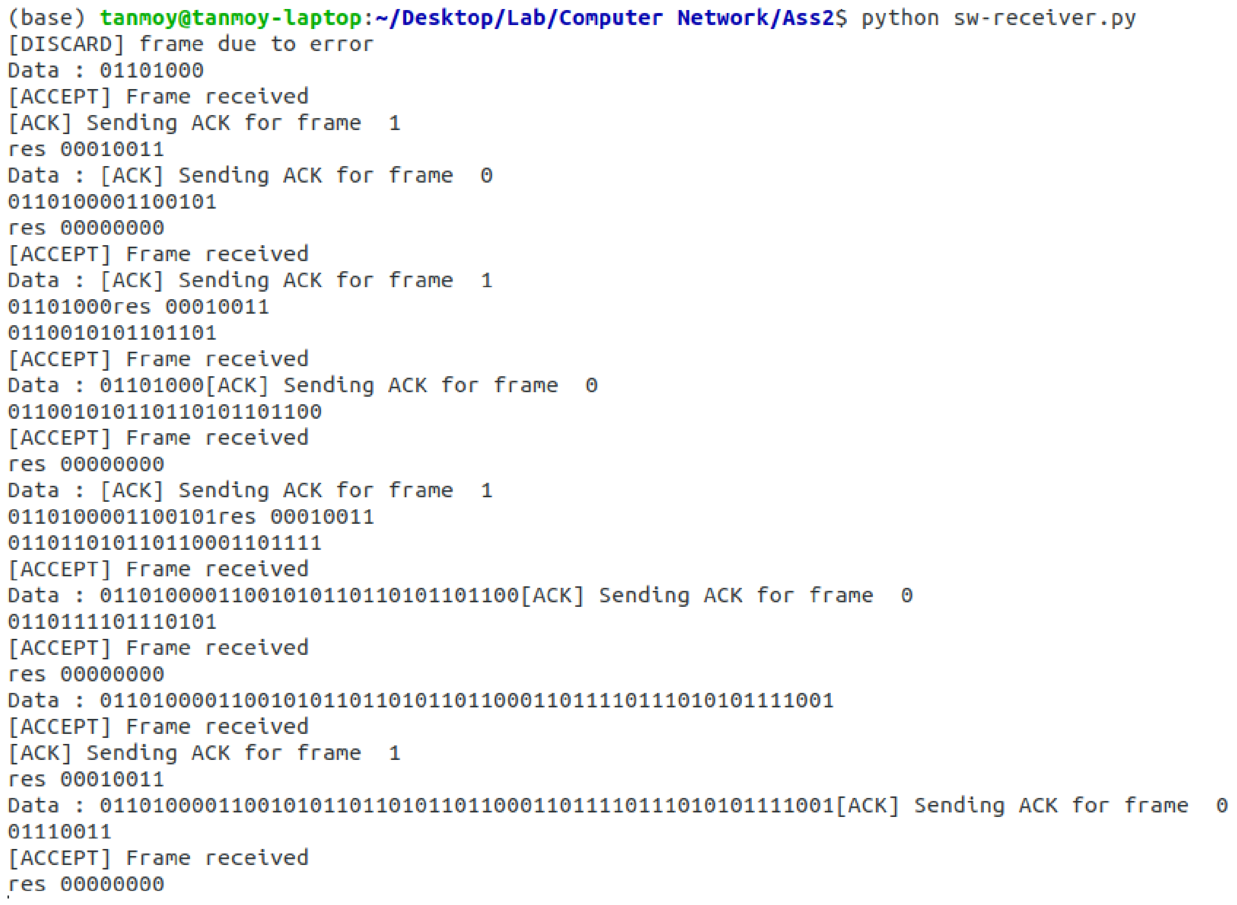
```
Server is online ...
Connected to: 127.0.0.1:34648
Connected to: 127.0.0.1:47108
Connected to: 127.0.0.1:39992
Setup done ! Sender 127.0.0.1:39992 is connected to receiver 127.0.0.1:34648
Injected Error : 00011010000110 ---> 00011011000110
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 00011010000110
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 00011010000110
127.0.0.1:34648 ---> 127.0.0.1:39992 --> 00010011
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 01011001011101
127.0.0.1:34648 ---> 127.0.0.1:39992 --> 00000000
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 00011011011001
127.0.0.1:34648 ---> 127.0.0.1:39992 --> 00010011
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 01011011000101
127.0.0.1:34648 ---> 127.0.0.1:39992 --> 00000000
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 00011011111111
127.0.0.1:34648 ---> 127.0.0.1:39992 --> 00010011
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 01011101011000
127.0.0.1:34648 ---> 127.0.0.1:39992 --> 00000000
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 00011110010000
127.0.0.1:34648 ---> 127.0.0.1:39992 --> 00010011
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 01011100110010
127.0.0.1:34648 ---> 127.0.0.1:39992 --> 00000000
```

Sender -



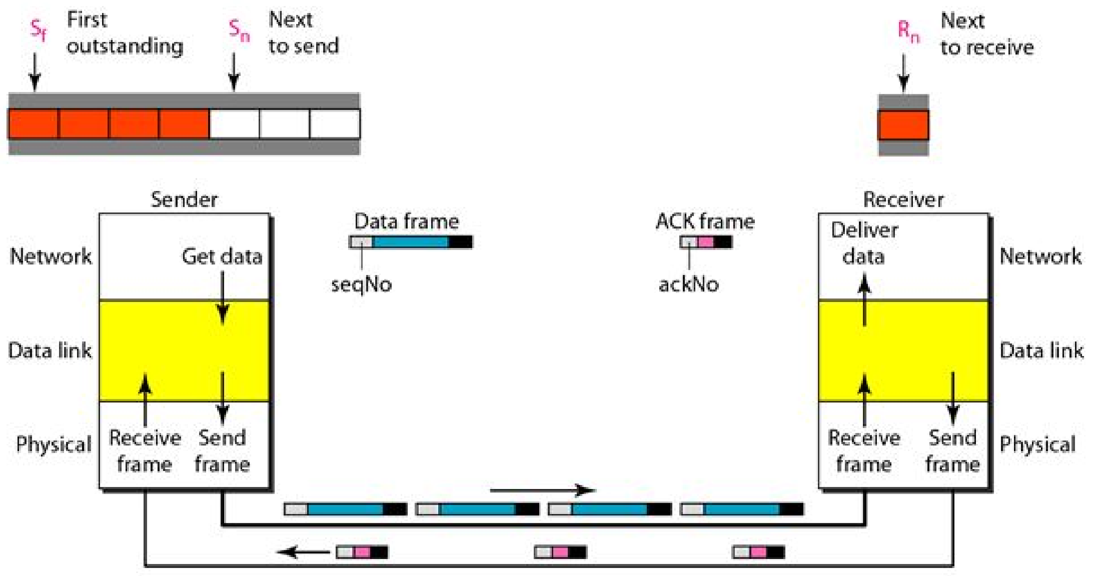
```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass2$ python sw-sender.py
Enter receiver ip: 127.0.0.1
Enter receiver port: 34648
[SEND] new frame  00011010000110  with sn  0
[RESEND] resending frame  00011010000110  with sn  1
[SEND] new frame  00011010000110  with sn  0
[ACK] ack discarded as rn not macthed
[SEND] new frame  01011001011101  with sn  1
[ACK] ack discarded as rn not macthed
[SEND] new frame  00011011011001  with sn  0
[ACK] ack discarded as rn not macthed
[SEND] new frame  01011011000101  with sn  1
[ACK] ack discarded as rn not macthed
[SEND] new frame  00011011111111  with sn  0
[ACK] ack discarded as rn not macthed
[SEND] new frame  01011101011000  with sn  1
[ACK] ack discarded as rn not macthed
[SEND] new frame  00011110010000  with sn  0
[ACK] ack discarded as rn not macthed
[SEND] new frame  01011100110010  with sn  1
[ACK] ack discarded as rn not macthed
No more frames to send
```

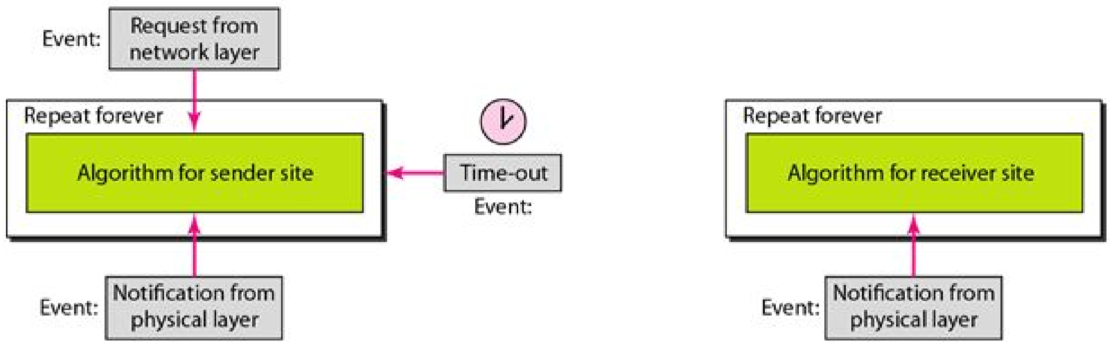
Receiver -

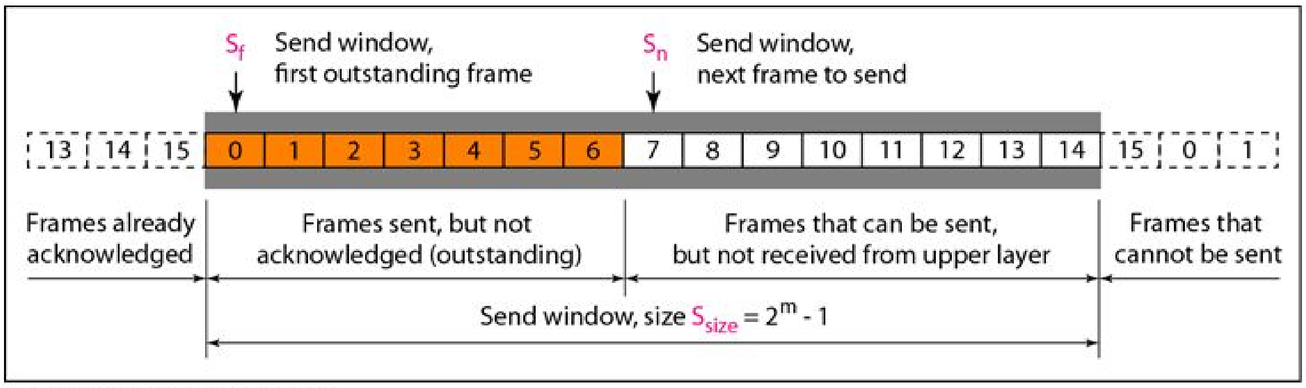


```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass2$ python sw-receiver.py
[DISCARD] frame due to error
Data : 01101000
[ACCEPT] Frame received
[ACK] Sending ACK for frame  1
res 00010011
Data : [ACK] Sending ACK for frame  0
0110100001100101
res 00000000
[ACCEPT] Frame received
Data : [ACK] Sending ACK for frame  1
01101000res 00010011
0110010101101101
[ACCEPT] Frame received
Data : 01101000[ACK] Sending ACK for frame  0
011001010110110101101100
[ACCEPT] Frame received
res 00000000
Data : [ACK] Sending ACK for frame  1
0110100001100101res 00010011
011011010110110001101111
[ACCEPT] Frame received
Data : 01101000011001010110110101101100[ACK] Sending ACK for frame  0
0110111101110101
[ACCEPT] Frame received
res 00000000
Data : 0110100001100101011011010110110001101111011101010111001
[ACCEPT] Frame received
[ACK] Sending ACK for frame  1
res 00010011
Data : 0110100001100101011011010110110001101111011101010111001[ACK] Sending ACK for frame  0
01110011
[ACCEPT] Frame received
res 00000000
```

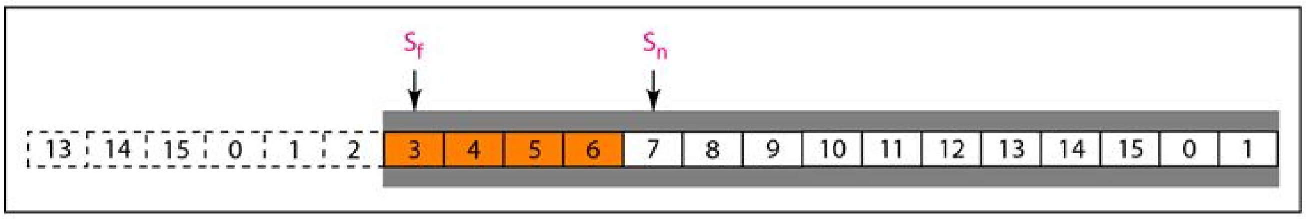
DESIGN OF GO-BACK-N PROTOCOL




Event:
Request from network layer
Repeat forever
Algorithm for sender site
Time-out
Event:
Event:
Notification from physical layer
Repeat forever
Algorithm for receiver site
Event:
Notification from physical layer


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

Code -

Sender Code -

```python
from cmath import sqrt
import socket
import threading
from time import sleep

from helpers import buildFrames, decodeData, encodeData


# FRAME FORMAT:
# For a frame with length of 8, the format is:
# [sn][data][parity]
# [2 ][  8 ][  4   ]

# Ack frame format:
# [sn][parity]
# [ 4 ][  4   ]

class Sender:
    def __init__(self, host, port, timeout, N=4):
        self.host = host
        self.port = port
        self.timeout = timeout
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((self.host, self.port))
        self.ackReceivedEvent = threading.Event()
        self.ackReceivedEvent.clear()

        self.N = N

        self.data = []
        self.frameIndex = 0
        self.sn = 0

        tmp = int(sqrt(N).real)
        self.snmax = int('1'*tmp, 2)

        # Configure the receiver
        receiver_ip = input("Enter receiver ip: ")
        if len(receiver_ip.strip()) == 0:
            receiver_ip = "127.0.0.1"
        receiver_port = input("Enter receiver port: ")
```

```
self.sock.sendall(str.encode(f'connect:{receiver_ip}:{receiver_port}'))
        sleep(1)

    def startProcess(self):
        self.senderThread = threading.Thread(target=self.send)
        self.receiverACKThread = threading.Thread(target=self.recvAck)
        self.receiverACKThread.start()
        self.senderThread.start()

    def send(self):
        while True:
            # Check frames
            if self.frameIndex >= len(self.data):
                print("No more frames to send")
                break
            # Prepare fram e-- by taking the sn and the element of window
            for index in range(self.frameIndex, self.frameIndex+self.N):
                if index >= len(self.data):
                    break
                frame =
encodeData(str(bin(self.sn)[2:]).zfill(2)+self.data[index])
                # Send frame
                print("[SEND] new frame ", frame, " with sn ", self.sn)
                self.sock.sendall(str.encode(frame))
                # Wait before sending next frame
                sleep(0.05)

            # Wait for ack with a timeout
            self.ackReceivedEvent.clear()
            isNotified = self.ackReceivedEvent.wait(timeout=self.timeout)
            if not isNotified:
                print("[RESEND] resending frame ", frame, " with sn ",
self.sn)
            # If timeout, resend the frame -- run the loop again


    def recvAck(self):
        while True:
            # Listen for data
            data = self.sock.recv(1024)
            data = data.decode()
            # If ack is valid
            decodedData = decodeData(data)
            if decodedData[0]:
                seqNo = int(decodedData[1], 2)
                # If ack sn == current sn, just emit ackreceived event
                if seqNo == self.sn:
                    self.increaseSn()
                    self.frameIndex += 1
                    self.ackReceivedEvent.set()
                    print("[ACK] ack received with sn ", seqNo)
                else:
                    print("[ACK] ack discarded as rn not macthed")
            else:
                print("[DISCARD] discarding ACK due to error")


    def increaseSn(self):
        self.sn += 1
        if self.sn > self.snmax:
            self.sn = 0

    def decreaseSn(self):
        self.sn -= 1
        self.sn = max(self.sn, 0)

    def pushData(self, data):
        frames = buildFrames(data, frameSize=8)
        for i in frames:
            self.data.append(i)



sender = Sender('127.0.0.1', 8081, 4)

sender.pushData("01101000011001010110110101101100011011110111010101110010111
0011")
sender.startProcess()
```

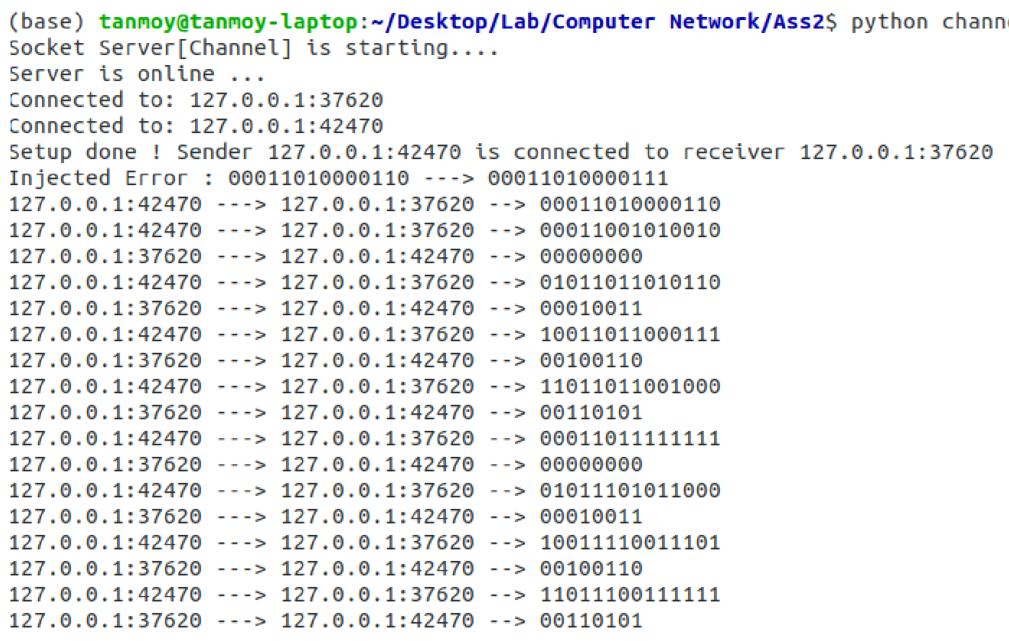
Receiver Code -

```
from cmath import sqrt
import socket
import threading

from helpers import decodeData, generateACK
```

```python
class Receiver:
    def __init__(self, host, port, N=4):
        self.host = host
        self.port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((self.host, self.port))
        self.frameReceived = threading.Event()
        self.frameReceived.clear()
        self.N = N

        self.data = []
        self.rn = 0
        tmp = int(sqrt(N).real)
        self.rnmax = int('1'*tmp, 2)

    def startProcess(self):
        self.sendACKThread = threading.Thread(target=self.sendACK)
        self.receiverThread = threading.Thread(target=self.recv)
        self.receiverThread.start()
        self.sendACKThread.start()


    def recv(self):
        while True:
            # Receieve data
            data = self.sock.recv(1024)
            if data == "disconnect:" : break
            data = data.decode()
            # Decode data
            decodedData = decodeData(data)
            # Check if valid
            if decodedData[0]:
                # Extract frame and seq no
                data = decodedData[1]
                seqNo = int(data[:2], 2)
                frame = data[2:]
                # If seq no == rn, save data and send ack for next frame
                if seqNo == self.rn:
                    self.data.append(frame)
                    self.frameReceived.set()
                    self.printData()
                    print("[ACCEPT] Frame received ")
                # If seq no != rn, discard and send ack fagain
                else:
                    print("[DISCARD] seqNo not matched to rn")
                    self.frameReceived.set()
            else:
                print("[DISCARD] frame due to error")

    def sendACK(self):
        while True:
            # Wait for frame received event
            self.frameReceived.wait()
            print("[ACK] Sending ACK for frame ", self.rn)
            # Send ack for the frame
            self.sock.sendall(str.encode(generateACK(self.rn,
with_parity=True)))
            # Clear frame received event
            self.increaseRn()
            self.frameReceived.clear()

    def increaseRn(self):
        self.rn += 1
        if self.rn > self.rnmax:
            self.rn = 0

    def printData(self):
        print("Data : ", end="", flush=True)
        for i in self.data:
            print(i, end="", flush=True)
        print()

receiver = Receiver("127.0.0.1", 8081)
receiver.startProcess()
```

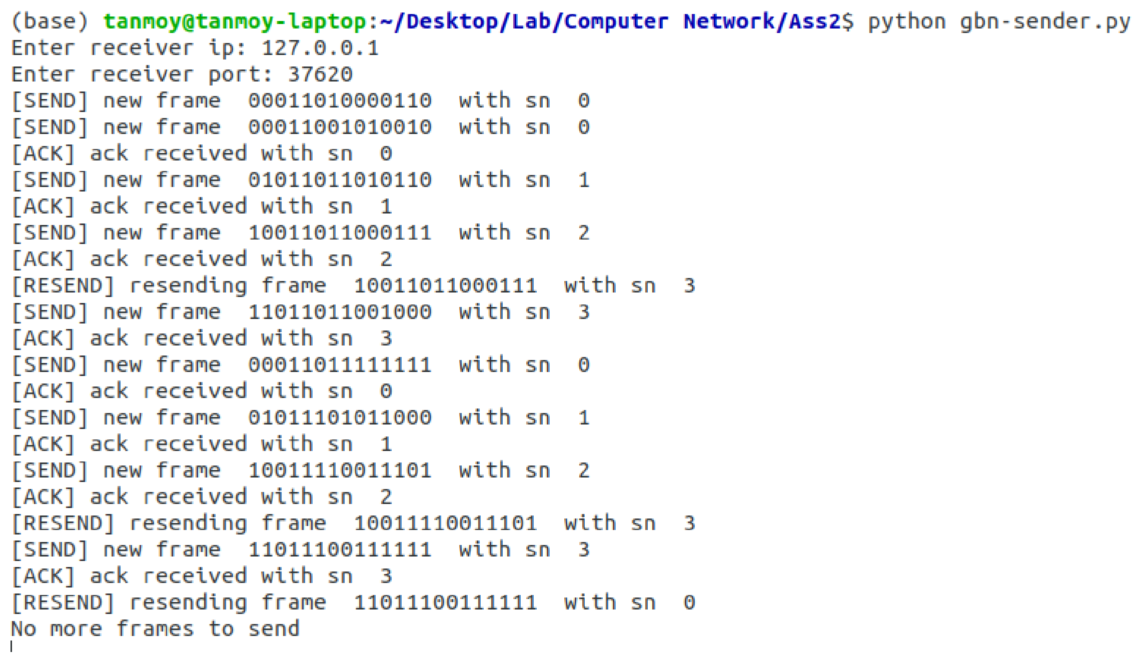
Output -

Channel -



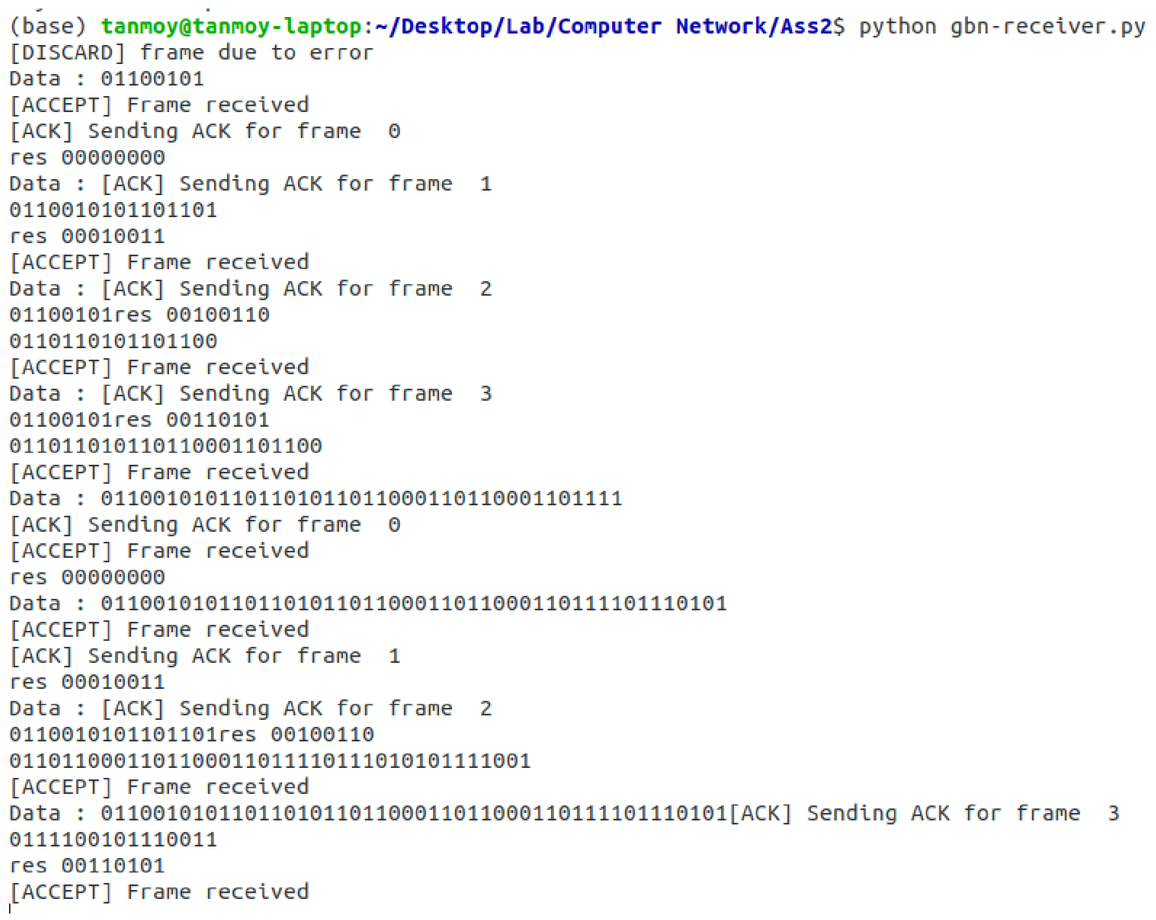
```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass2$ python channel.py
Socket Server[Channel] is starting....
Server is online ...
Connected to: 127.0.0.1:37620
Connected to: 127.0.0.1:42470
Setup done ! Sender 127.0.0.1:42470 is connected to receiver 127.0.0.1:37620
Injected Error : 00011010000110 ---> 00011010000111
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 00011010000110
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 00011001010010
127.0.0.1:37620 ---> 127.0.0.1:42470 --> 00000000
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 01011011010110
127.0.0.1:37620 ---> 127.0.0.1:42470 --> 00010011
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 10011011000111
127.0.0.1:37620 ---> 127.0.0.1:42470 --> 00100110
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 11011011001000
127.0.0.1:37620 ---> 127.0.0.1:42470 --> 00110101
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 00011011111111
127.0.0.1:37620 ---> 127.0.0.1:42470 --> 00000000
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 01011101011000
127.0.0.1:37620 ---> 127.0.0.1:42470 --> 00010011
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 10011110011101
127.0.0.1:37620 ---> 127.0.0.1:42470 --> 00100110
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 11011100111111
127.0.0.1:37620 ---> 127.0.0.1:42470 --> 00110101
```

Sender -



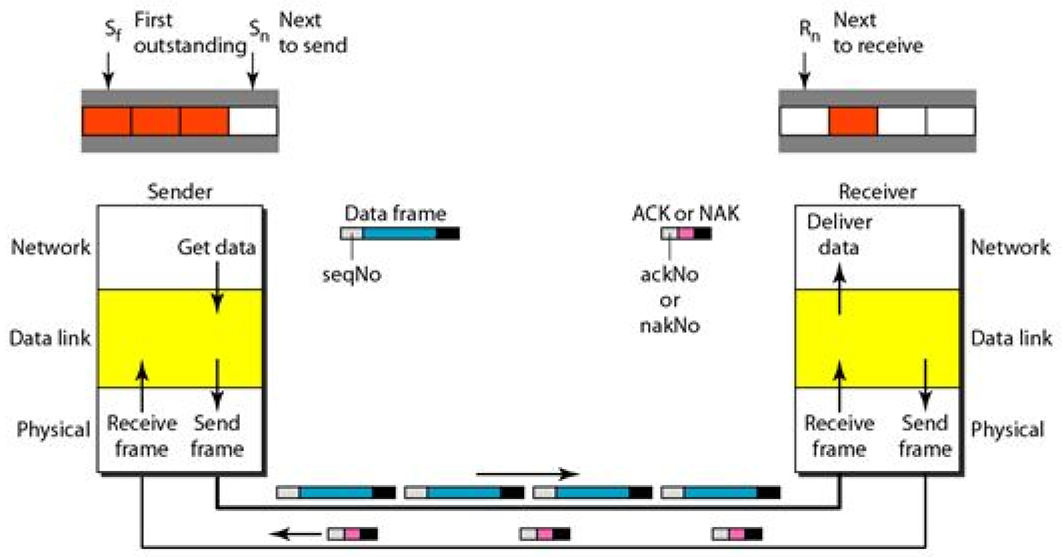
```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass2$ python gbn-sender.py
Enter receiver ip: 127.0.0.1
Enter receiver port: 37620
[SEND] new frame  00011010000110  with sn  0
[SEND] new frame  00011001010010  with sn  0
[ACK] ack received with sn  0
[SEND] new frame  01011011010110  with sn  1
[ACK] ack received with sn  1
[SEND] new frame  10011011000111  with sn  2
[ACK] ack received with sn  2
[RESEND] resending frame  10011011000111  with sn  3
[SEND] new frame  11011011001000  with sn  3
[ACK] ack received with sn  3
[SEND] new frame  00011011111111  with sn  0
[ACK] ack received with sn  0
[SEND] new frame  01011101011000  with sn  1
[ACK] ack received with sn  1
[SEND] new frame  10011110011101  with sn  2
[ACK] ack received with sn  2
[RESEND] resending frame  10011110011101  with sn  3
[SEND] new frame  11011100111111  with sn  3
[ACK] ack received with sn  3
[RESEND] resending frame  11011100111111  with sn  0
No more frames to send
```

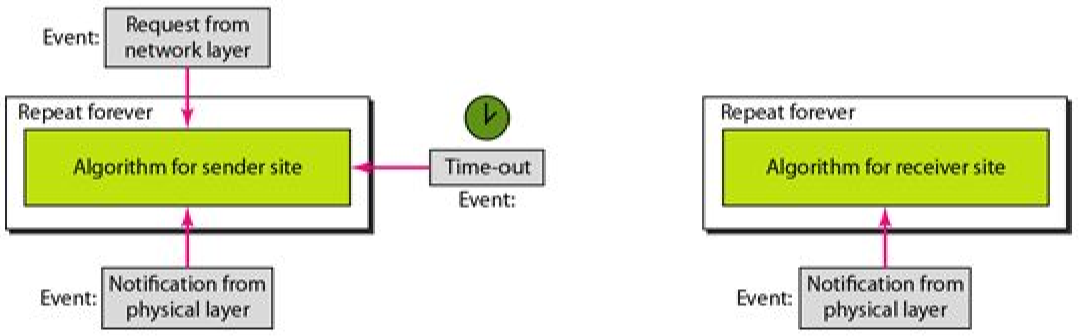
Receiver -

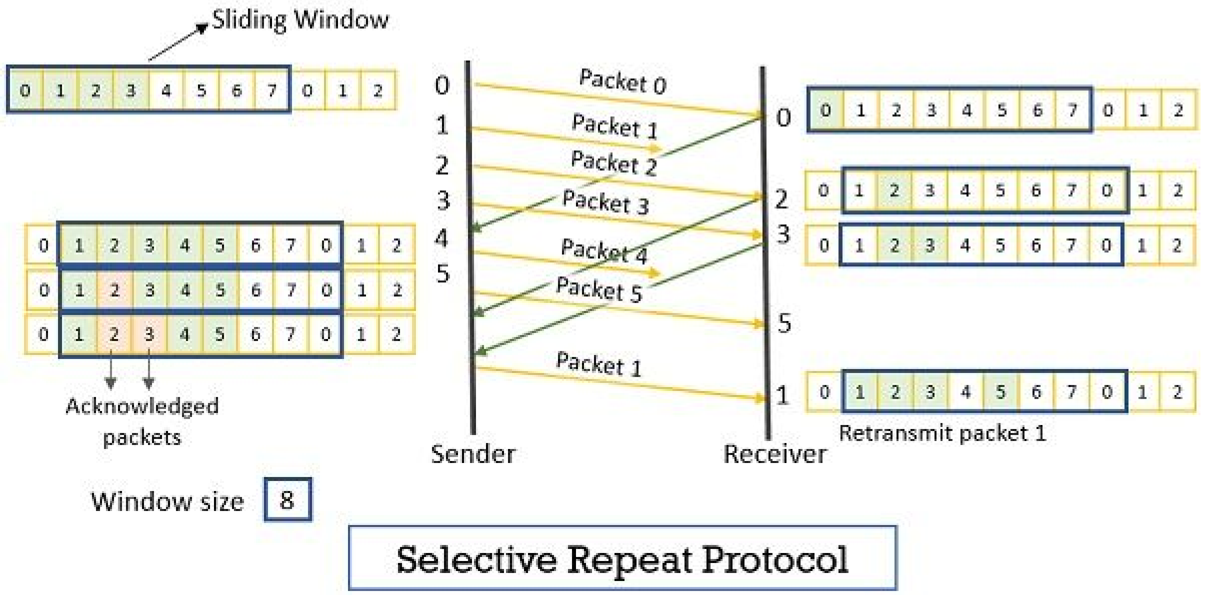


```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass2$ python gbn-receiver.py
[DISCARD] frame due to error
Data : 01100101
[ACCEPT] Frame received
[ACK] Sending ACK for frame  0
res 00000000
Data : [ACK] Sending ACK for frame  1
0110010101101101
res 00010011
[ACCEPT] Frame received
Data : [ACK] Sending ACK for frame  2
01100101res 00100110
0110110101101100
[ACCEPT] Frame received
Data : [ACK] Sending ACK for frame  3
01100101res 00110101
011011010110110001101100
[ACCEPT] Frame received
Data : 0110010101101101011011000110110001101111
[ACK] Sending ACK for frame  0
[ACCEPT] Frame received
res 00000000
Data : 011001010110110101101100011011000110111101110101
[ACCEPT] Frame received
[ACK] Sending ACK for frame  1
res 00010011
Data : [ACK] Sending ACK for frame  2
0110010101101101res 00100110
01101100011011000110111101110101111001
[ACCEPT] Frame received
Data : 011001010110110101101100011011000110111101110101[ACK] Sending ACK for frame  3
0111100101110011
res 00110101
[ACCEPT] Frame received
```

DESIGN OF SELECTIVE REPEAT PROTOCOL




Event: Request from network layer
Repeat forever
Algorithm for sender site
Time-out Event:
Event: Notification from physical layer
Repeat forever
Algorithm for receiver site
Event: Notification from physical layer



Code -

Sender Code -

```
 from cmath import sqrt
```

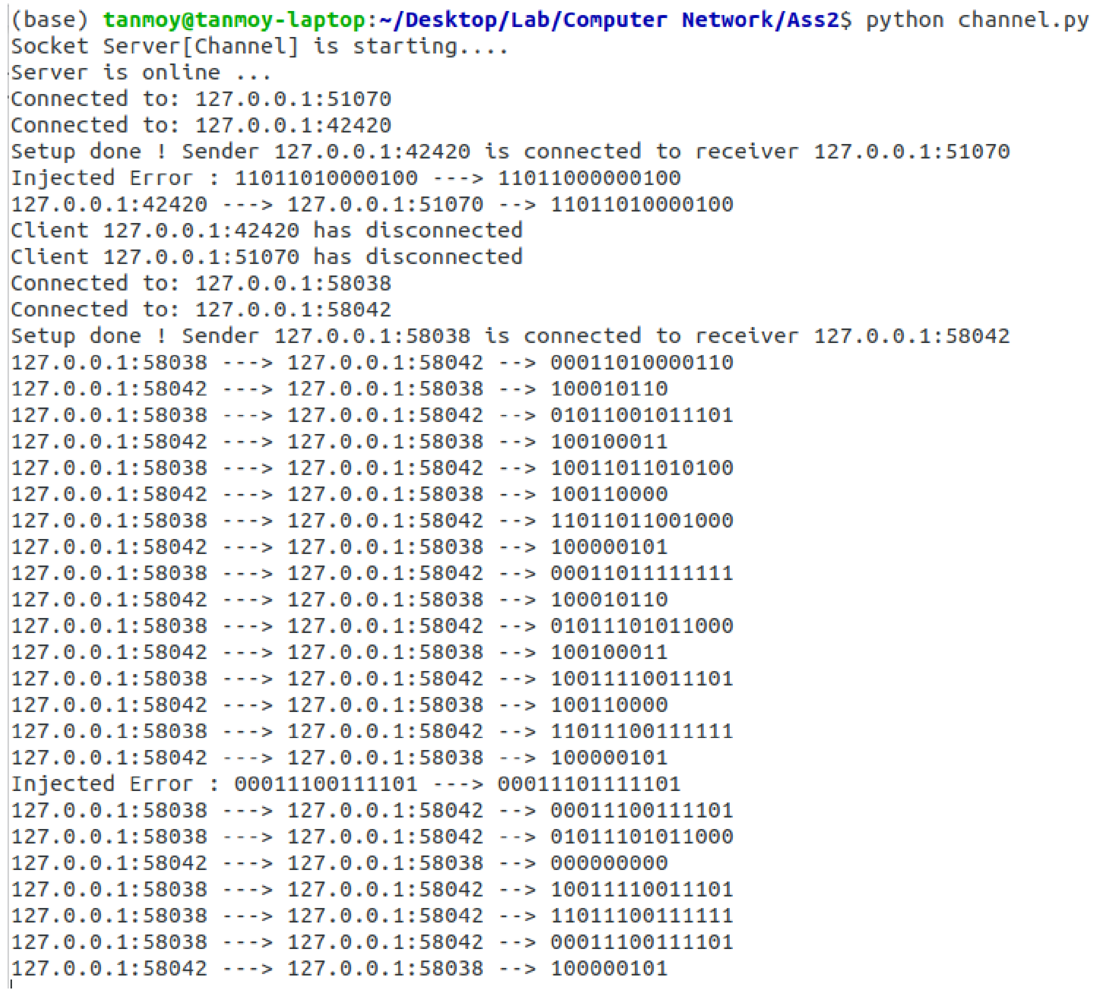
```python
from queue import Queue
import socket
import threading
from time import sleep

from helpers import buildFrames, decodeData, encodeData
from queuec import QueueC


# FRAME FORMAT:
# For a frame with length of 8, the format is:
# [sn][data][parity]
# [2 ][  8 ][  4   ]

# Ack frame format:
# [type][sn][parity]
# [  1 ][ 4 ][  4   ]
# Type -> ACK -> 1, NAK -> 0

class Sender:
    def __init__(self, host, port, timeout, N=4):
        self.host = host
        self.port = port
        self.timeout = timeout
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((self.host, self.port))

        self.data = Queue()
        self.sn = 0
        self.sf = 0


        # tmp = int(sqrt(N).real)
        self.sw = 2**(N-1)
        self.N = N

        self.oldFramesData = {}

        # Window frame status
        self.timers = [None]*(N)

        # Configure the receiver
        receiver_ip = input("Enter receiver ip: ")
        if receiver_ip == None or  len(receiver_ip.strip()) == 0:
            receiver_ip = "127.0.0.1"
        receiver_port = input("Enter receiver port: ")

self.sock.sendall(str.encode(f'connect:{receiver_ip}:{receiver_port}'))
        sleep(1)

    def startProcess(self):
        self.senderThread = threading.Thread(target=self.send)
        self.receiverACKThread = threading.Thread(target=self.recvAck)
        self.receiverACKThread.start()
        self.senderThread.start()

    def send(self):
        while True:
            if self.data.empty():
                print("end....")
                break

            if (self.sn - self.sf) < self.sw:
                # Prepare fram e-- by taking the sn and the element of window
                x = self.data.get()
                # Make frame
                frame =  encodeData(str(bin(self.sn)[2:]).zfill(2)+x)
                # Store frame
                self.oldFramesData[self.sn] = frame
                # Send frame
                print("Sending frame: ", frame)
                self.sock.sendall(str.encode(frame))
                # Start timer
                self.startTimer(self.sn)
                # Increment sn
                self.sn = (self.sn+1)%self.N

            sleep(0.2)

    def recvAck(self):
        while True:
            # Receive frae
            data = self.sock.recv(1024)
            data = data.decode()
            # If ack is valid
            decodedData = decodeData(data)
            if decodedData[0]:
                # print("Received ack: ", decodedData[1])
                seqNo = (int(decodedData[1][1:], 2)-1)%self.N
                if decodedData[1][0] == '1':
                    print("[ACK] for frame ", seqNo)
                    # ACK
                    if seqNo > self.sf and seqNo <= self.sn:
                        while self.sf <= seqNo:
                            # Start timer
                            if self.timers[seqNo]:
                                self.timers[seqNo].cancel()
                                self.timers[seqNo] = None
                            self.sf = (self.sf+1)%self.N
                else:
                    # NAK
                    print("[NAK] for frame ", seqNo)
                    if seqNo > self.sf and seqNo < self.sn:
                        self.resendFrameAndSetTimer(seqNo)
            else:
                print("[DISCARD] acknolwedgement de to error")
            # sleep(0.2)


    def startTimer(self, seqNo):
        if self.timers[seqNo]:
            self.timers[seqNo].cancel()
        self.timers[seqNo] = threading.Timer(self.timeout, self.resendFrame,
[seqNo])
        self.timers[seqNo].start()

    def resendFrame(self, seqNo):
        if seqNo not in self.oldFramesData:
            return
        frame =  self.oldFramesData[seqNo]
        print("Resending frame: ", frame)
        self.sock.sendall(str.encode(frame))
        sleep(0.2)

    def resendFrameAndSetTimer(self, seqNo):
        if seqNo not in self.oldFramesData:
            return
        frame =  self.oldFramesData[seqNo] # It wil have encoded frame
        if len(frame) == 14:
            self.sock.sendall(str.encode(frame))
            print("Resending frame: ", frame)
            self.startTimer(seqNo)
            sleep(0.2)


    def pushData(self, data):
        frames = buildFrames(data, frameSize=8)
        for i in frames:
            self.data.put(i)




sender = Sender('127.0.0.1', 8081, 4)

sender.pushData("01101000011001010110110101101100011011110111010101110010111
001101110011")

sender.startProcess()
```

Receiver Code -

```
from cmath import sqrt
from queue import Queue
import socket
import threading
from time import sleep

from helpers import decodeData, generateACK

# FRAME FORMAT:
# For a frame with length of 8, the format is:
# [sn][data][parity]
# [2 ][  8 ][  4   ]

# Ack frame format:
# [type][sn][parity]
# [  1 ][ 4 ][  4   ]
# Type -> ACK -> 1, NAK -> 0


class Receiver:
    def __init__(self, host, port, N=4):
        self.host = host
        self.port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((self.host, self.port))
        self.N = N
        self.rn = 0
        # tmp = int(sqrt(N).real)
        self.rnmax = 2**(N-1)

        self.data = []
        self.buffer = ["0"]*self.N

        self.nakSent = False
        self.ackNeeded = False

        self.marked = [False]*self.N

    def startProcess(self):
        # self.sendACKThread = threading.Thread(target=self.sendACK)
        self.receiverThread = threading.Thread(target=self.recv)
        self.receiverThread.start()
        # self.sendACKThread.start()


    def recv(self):
        while True:
            # Receieve frame
            data = self.sock.recv(1024)
            if not data:
                continue
            if data == "disconnect:" : break
            data = data.decode()
            print("data", data)

            # Decode frame
            decodedData = decodeData(data)
            # Check if valid
            if decodedData[0]:
                print("[ACCEPTED] Valid frame")
                # Extract frame and seq no
                data = decodedData[1]
                seqNo = int(data[:2], 2)
                frame = data[2:]
                # if seqno != rn
                if seqNo != self.rn and not self.nakSent:
                    self.sendNAK()
                    self.nakSent = True
                # If seq no == rn, save data and send ack for next frame
                if self.rn <= seqNo <= self.rnmax and not self.marked[seqNo]:
                    # Store frame
                    self.buffer[seqNo] = frame
                    # Mark Received
                    self.marked[seqNo] = True
                    while self.marked[self.rn]:
                        self.data.append(self.buffer[self.rn])
                        self.marked[self.rn] = False
                        self.rn = (self.rn+1)%self.N
                        self.ackNeeded = True

                    if self.ackNeeded:
                        self.sendACK()
                        self.ackNeeded = False
                        self.nakSent = False
            else:
                print("[DISCARD] Discarding frame due to error")

            self.printData()

    def sendACK(self):
        print("[ACK] Sending ACK for ", self.rn)
        data = str.encode(generateACK(self.rn, with_parity=True,
for_selective_repeat=True, isNak=False))
        self.sock.sendall(data)
        sleep(0.1)

    def sendNAK(self):
        print("[NAK] Sending NAK for ", self.rn)
        data = str.encode(generateACK(self.rn, with_parity=True,
for_selective_repeat=True, isNak=True))
        self.sock.sendall(data)
        sleep(0.1)

    def printData(self):
        print("Data ", end="")
        for frame in self.data:
            print(frame, end="", flush=True)
        print()

receiver = Receiver("127.0.0.1", 8081)
receiver.startProcess()
```

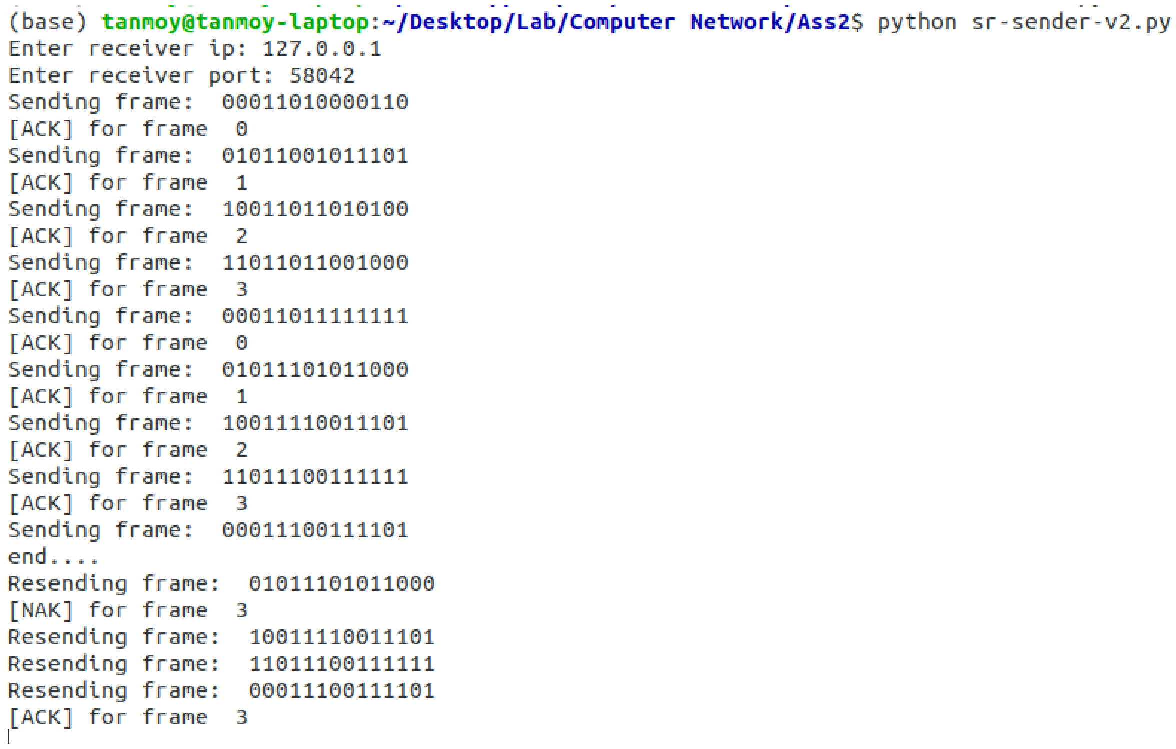
Output -

Channel -



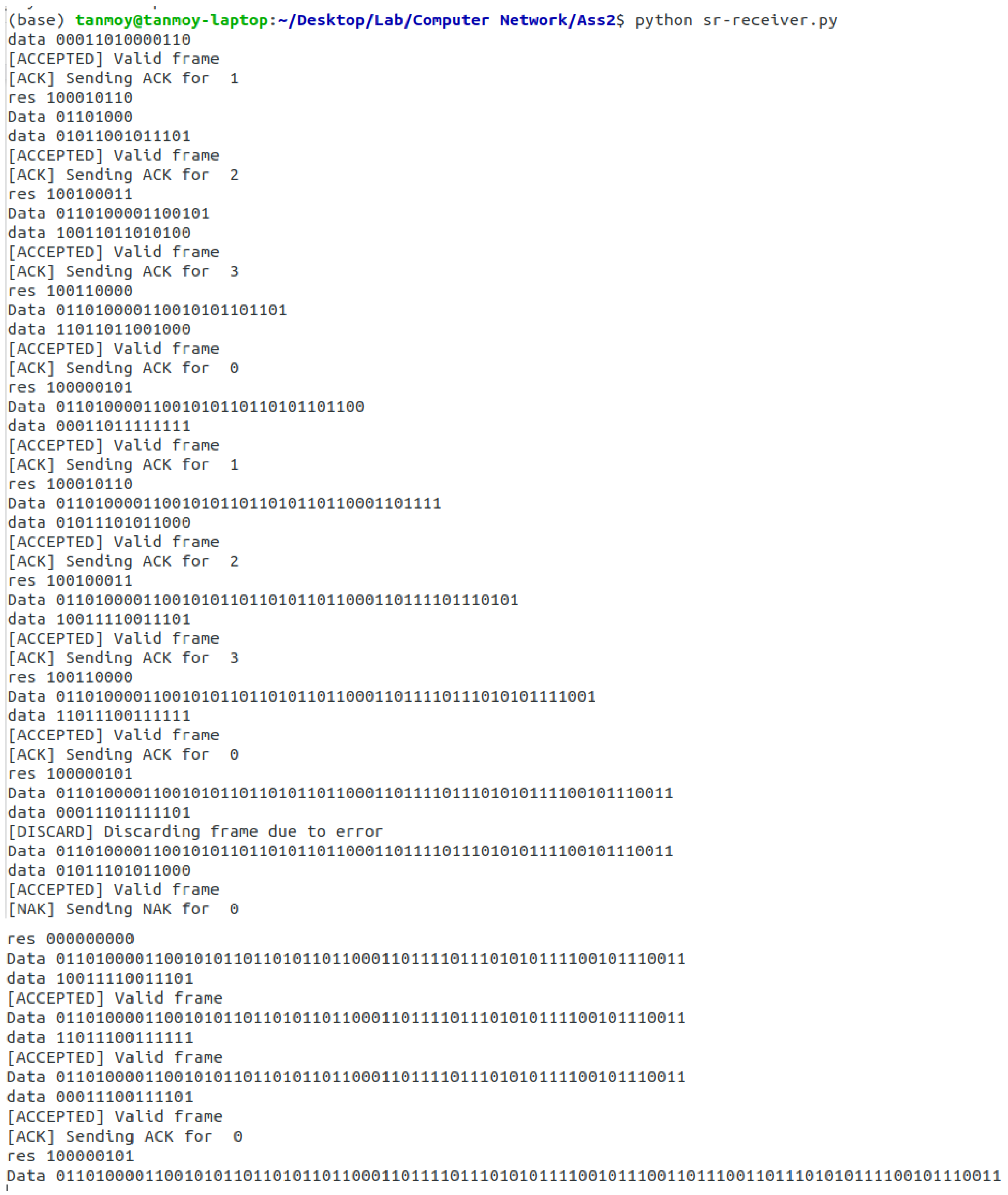
```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass2$ python channel.py
Socket Server[Channel] is starting....
Server is online ...
Connected to: 127.0.0.1:51070
Connected to: 127.0.0.1:42420
Setup done ! Sender 127.0.0.1:42420 is connected to receiver 127.0.0.1:51070
Injected Error : 11011010000100 ---> 11011000000100
127.0.0.1:42420 ---> 127.0.0.1:51070 --> 11011010000100
Client 127.0.0.1:42420 has disconnected
Client 127.0.0.1:51070 has disconnected
Connected to: 127.0.0.1:58038
Connected to: 127.0.0.1:58042
Setup done ! Sender 127.0.0.1:58038 is connected to receiver 127.0.0.1:58042
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 00011010000110
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100010110
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 01011001011101
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100100011
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 10011011010100
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100110000
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 11011011001000
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100000101
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 00011011111111
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100010110
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 01011101011000
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100100011
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 10011110011101
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100110000
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 11011100111111
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100000101
Injected Error : 00011100111101 ---> 00011101111101
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 00011100111101
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 01011101011000
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 000000000
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 10011110011101
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 11011100111111
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 00011100111101
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100000101
```

Sender -



```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass2$ python sr-sender-v2.py
Enter receiver ip: 127.0.0.1
Enter receiver port: 58042
Sending frame:  00011010000110
[ACK] for frame  0
Sending frame:  01011001011101
[ACK] for frame  1
Sending frame:  10011011010100
[ACK] for frame  2
Sending frame:  11011011001000
[ACK] for frame  3
Sending frame:  00011011111111
[ACK] for frame  0
Sending frame:  01011101011000
[ACK] for frame  1
Sending frame:  10011110011101
[ACK] for frame  2
Sending frame:  11011100111111
[ACK] for frame  3
Sending frame:  00011100111101
end....
Resending frame:  01011101011000
[NAK] for frame  3
Resending frame:  10011110011101
Resending frame:  11011100111111
Resending frame:  00011100111101
[ACK] for frame  3
```

Receiver -



```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass2$ python sr-receiver.py
data 00011010000110
[ACCEPTED] Valid frame
[ACK] Sending ACK for  1
res 100010110
Data 01101000
data 01011001011101
[ACCEPTED] Valid frame
[ACK] Sending ACK for  2
res 100100011
Data 0110100001100101
data 10011011010100
[ACCEPTED] Valid frame
[ACK] Sending ACK for  3
res 100110000
Data 011010000110010101101101
data 11011011001000
[ACCEPTED] Valid frame
[ACK] Sending ACK for  0
res 100000101
Data 01101000011001010110110101101100
data 00011011111111
[ACCEPTED] Valid frame
[ACK] Sending ACK for  1
res 100010110
Data 0110100001100101011011010110110001101111
data 01011101011000
[ACCEPTED] Valid frame
[ACK] Sending ACK for  2
res 100100011
Data 011010000110010101101101011011000110111101110101
data 10011110011101
[ACCEPTED] Valid frame
[ACK] Sending ACK for  3
res 100110000
Data 01101000011001010110110101101100011011110111010101111001
data 11011100111111
[ACCEPTED] Valid frame
[ACK] Sending ACK for  0
res 100000101
Data 0110100001100101011011010110110001101111011101010111100101110011
data 00011101111101
[DISCARD] Discarding frame due to error
Data 0110100001100101011011010110110001101111011101010111100101110011
data 01011101011000
[ACCEPTED] Valid frame
[NAK] Sending NAK for  0

res 000000000
Data 0110100001100101011011010110110001101111011101010111100101110011
data 10011110011101
[ACCEPTED] Valid frame
Data 0110100001100101011011010110110001101111011101010111100101110011
data 11011100111111
[ACCEPTED] Valid frame
Data 0110100001100101011011010110110001101111011101010111100101110011
data 00011100111101
[ACCEPTED] Valid frame
[ACK] Sending ACK for  0
res 100000101
Data 011010000110010101101101011011000110111101110101011110010111001101110011011101010111100101110011
```

RESULTS AND ANALYSIS

All the protocol is working as expected. The error will not happen in a real-world scenario as we do in the programme. It can be more random. Also, ack the loss will be much in that case. So in the real world, the performance can be slightly different than expected.