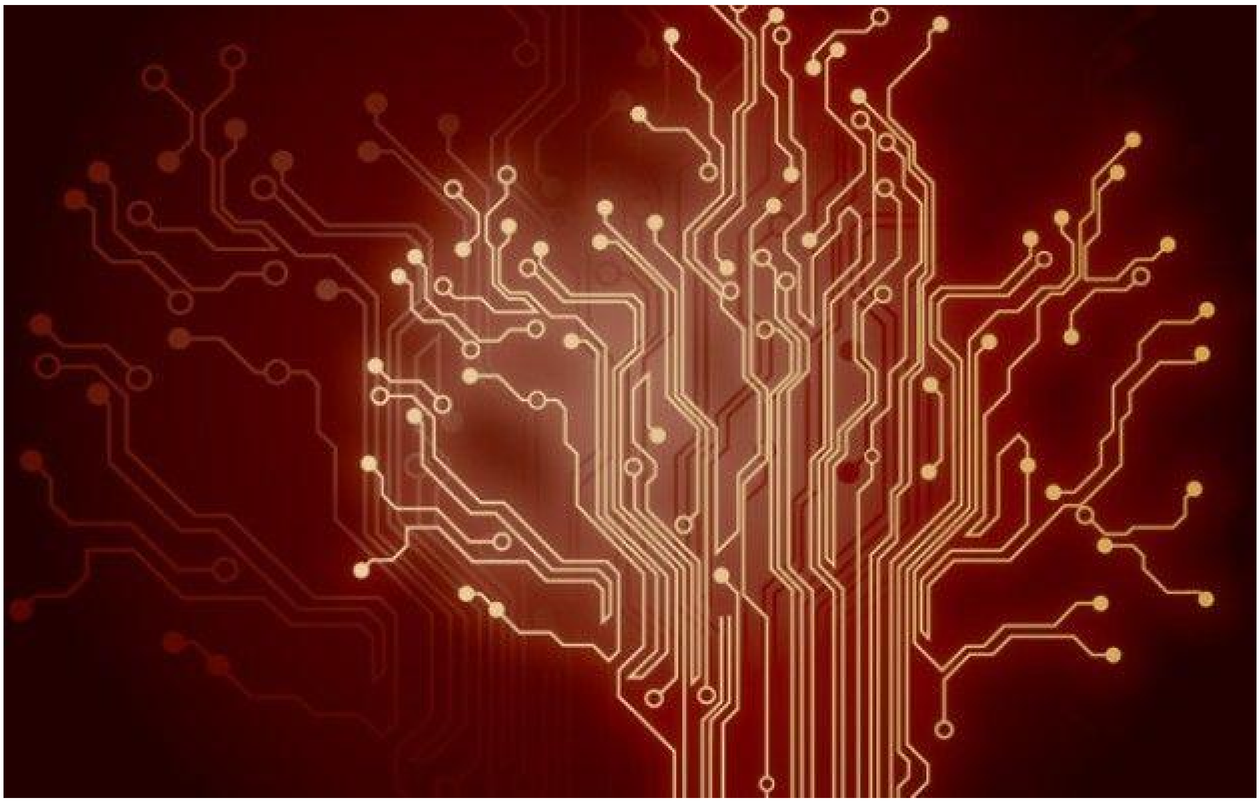
# NETWORK LAB REPORT

*CO2: Implement three data link layer protocols, Stop and Wait, GoBack N Sliding Window and Selective Repeat Sliding Window for flow control.*



**Atanu Ghosh**

BCSE-III (2019-2023) 5th sem, Section: A-1,

Roll: 001910501005, Date: 10/08/2021

## ASSIGNMENT-2

## Design and implement three data link layer protocols

### PROBLEM STATEMENT

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.

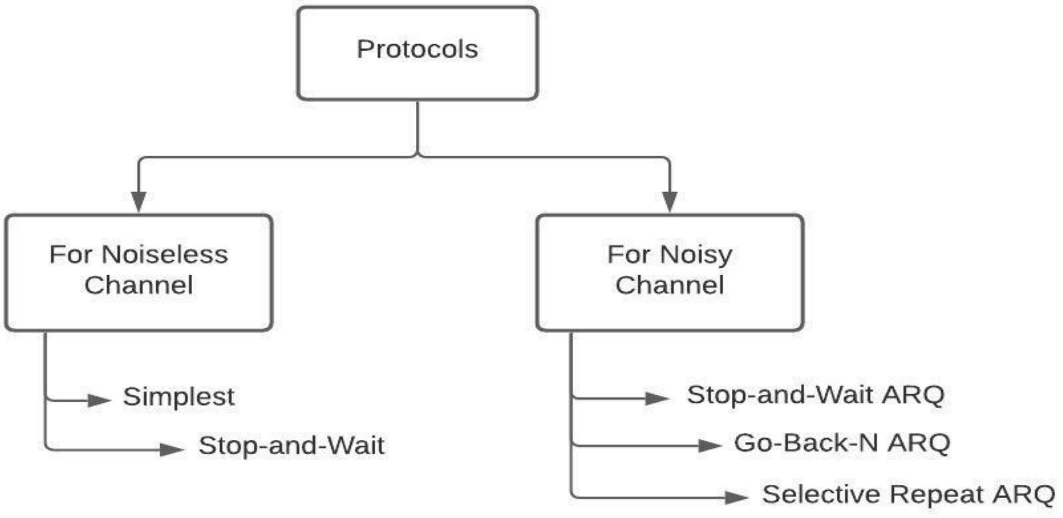
**Sliding window**:

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.

## DESIGN

[ This project is implemented in the Python 3 programming language (Tested with CPython 3.9.6). A mixture of procedural and object-oriented techniques along with inter-process communication using TCP sockets (provided by the runtime system) has been used. ]



We will see how the data link layer can combine framing, flow control and error control to achieve the delivery of data from one node to another. In our implementation, as the channel will be injecting errors, we are going to implement the three protocols for the noisy channels.

In this assignment, we shall discuss the following data link layer protocols in detail.

1. Stop and Wait protocol
2. Go Back N Sliding Window protocol
3. Selective Repeat Sliding Window protocol

I have implemented the error detection module in three program files.

- sender.py (Sender program (we can create multiple sender processes))
- channel.py (Program for the single-channel process)
- receiver.py (Receiver program (we can create multiple receiver processes))

There are 3 principal components of the System (for each protocol):

1. **Sender :** The following are the tasks performed in this Sender program :

    a. We can create more than one sender process, to send a message to the channel.
    b. It first waits for the user to input a binary input string.
    c. The appropriate frame is created using the above input string.
    d. This frame is then sent to the channel.
    e. It waits for an ACK/NAK to be received from the channel, notifying the successful delivery of binary input message.
    f. If it does not receives an ACK/NAK for a time period of 2s, it resends again (according to the protocols defined).

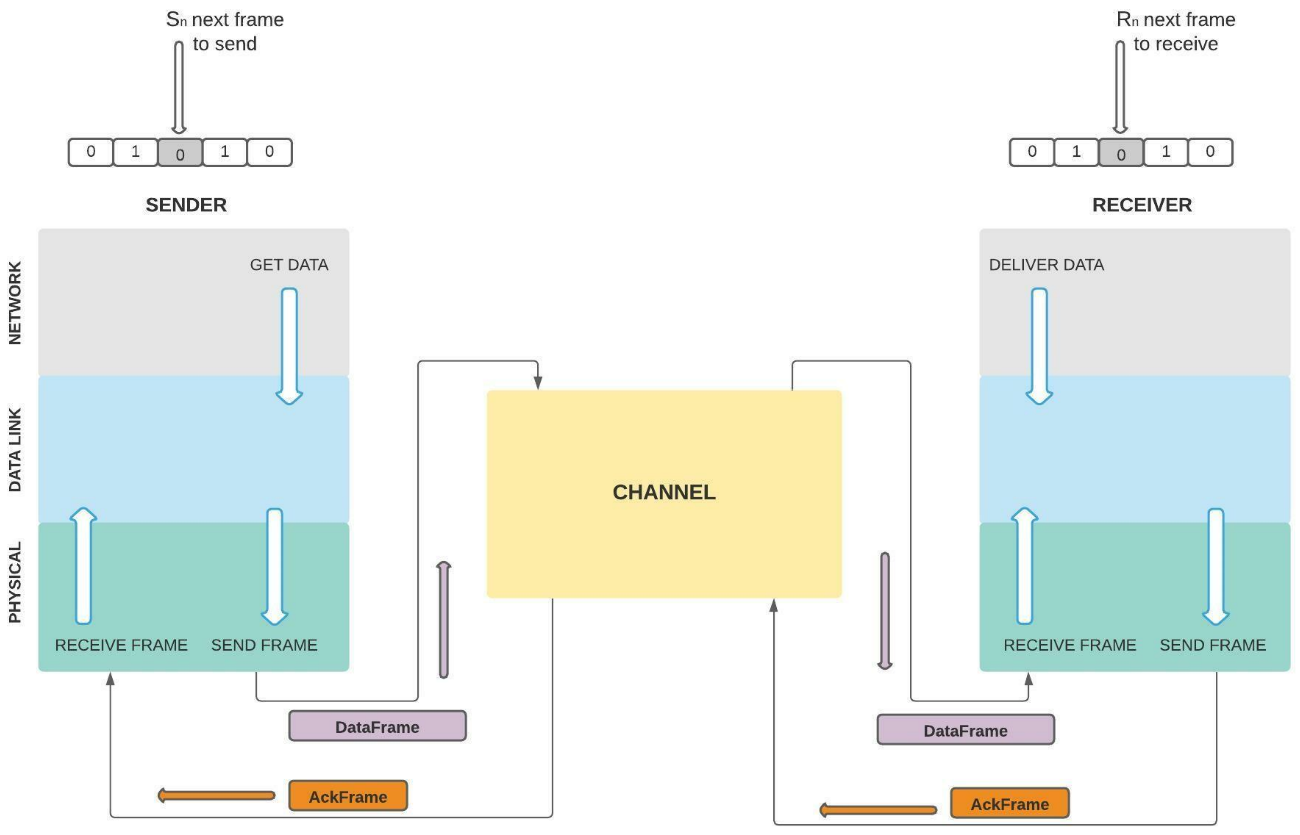
2. **Channel :** The following are the tasks performed in this Channel program :

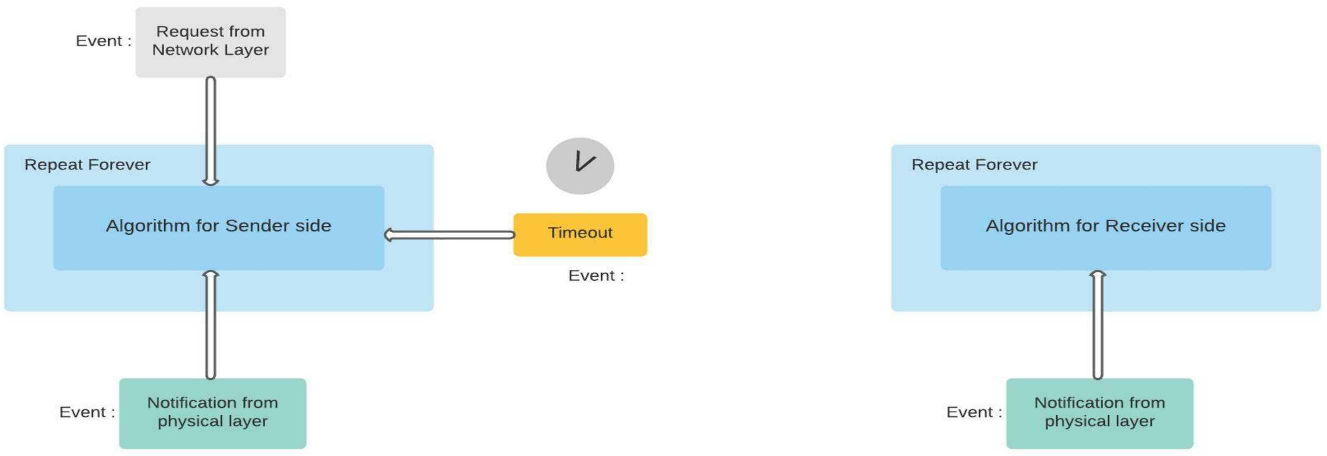
    a. The channel process first takes a number of senders and receivers as input.
    b. It initiates and connects all the sender and receiver processes.
    c. It receives the frame from any of the current senders.
    d. It then injects errors randomly into the data frame.
    e. Then the frame is sent to one of the receiver processes.
    f. The receiver then sends an ACK/NAK for the data received, to the channel.
    g. The channel then passes the ACK/NAK status to its corresponding sender.

3. **Receiver :** The following are the tasks performed in this Receiver program :

    a. The receiver process first waits for a message to be received from the channel.
    b. It then adds a random amount of time delay, before the message is sent back to its channel.
    c. It checks for any error in the data frame received and sends a message ACK/NAK accordingly.
    d. The above message is then sent to the channel.

## SCHEMATIC DIAGRAM (as a part of DESIGN)





Event : Request from Network Layer
Repeat Forever
Algorithm for Sender side
Timeout
Event :
Event : Notification from physical layer
Repeat Forever
Algorithm for Receiver side
Event : Notification from physical layer

Fig : DESIGN OF STOP AND WAIT PROTOCOL

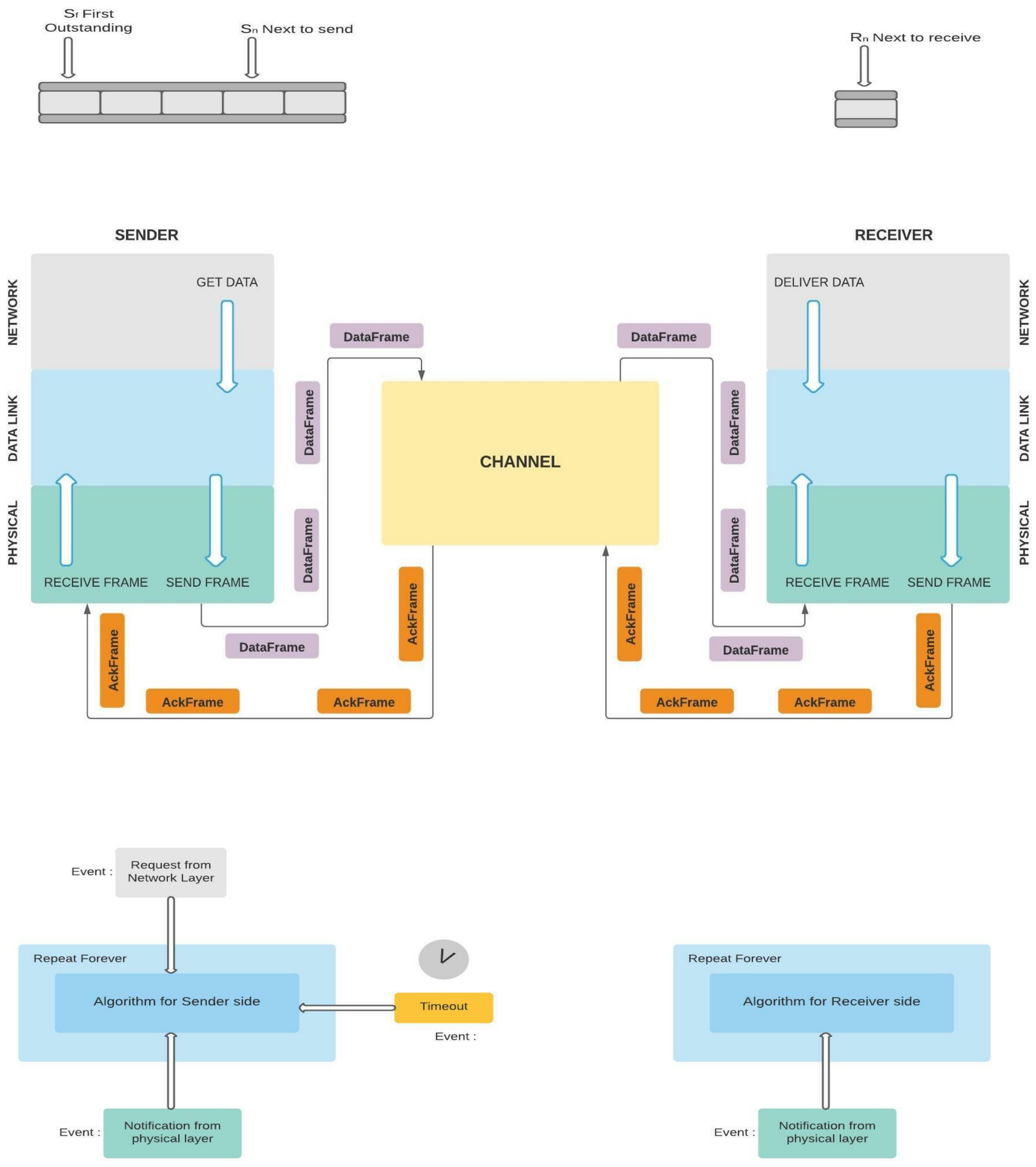
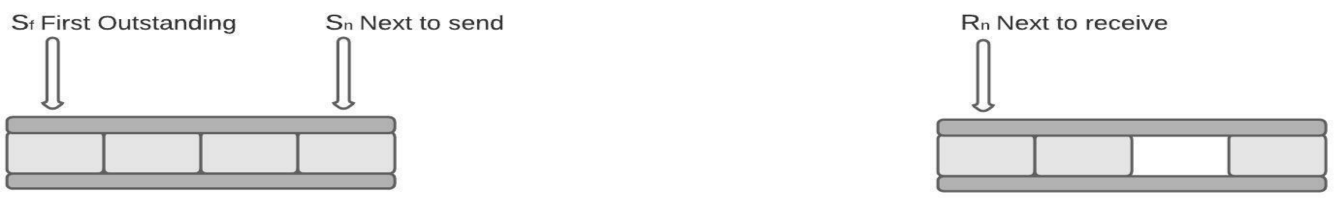
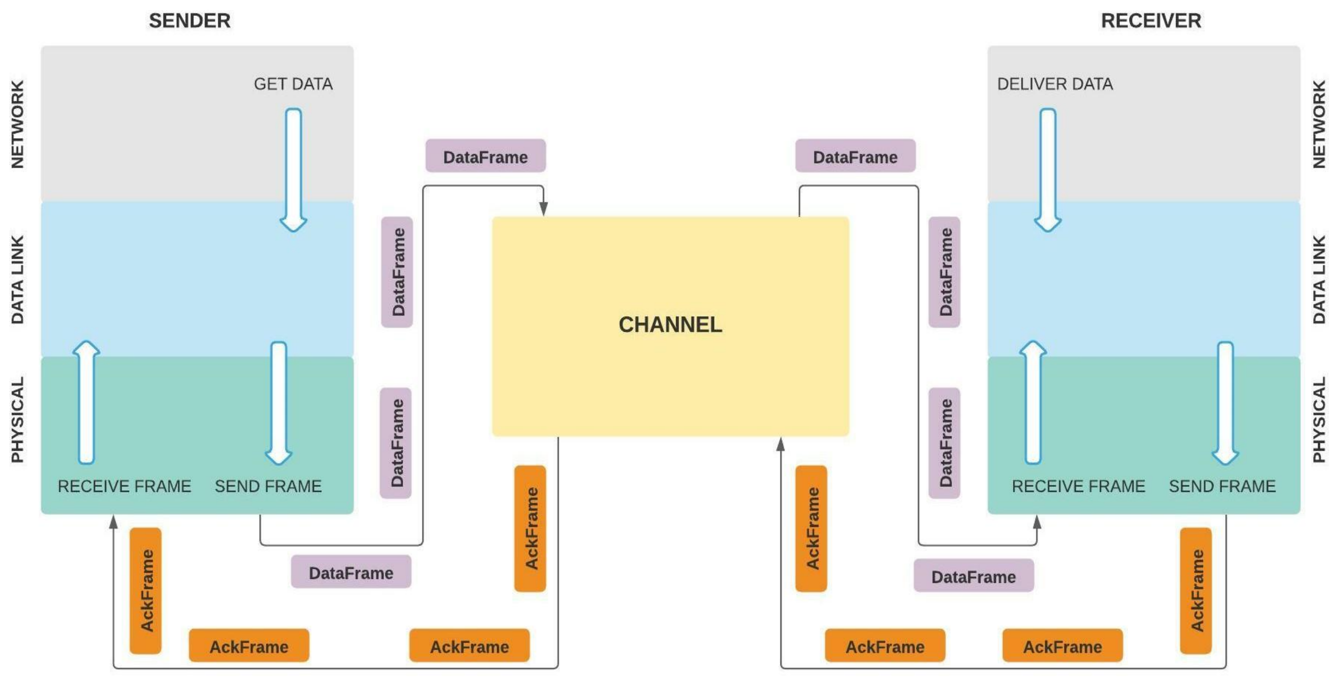


Fig : DESIGN OF GO BACK N SLIDING WINDOW PROTOCOL:





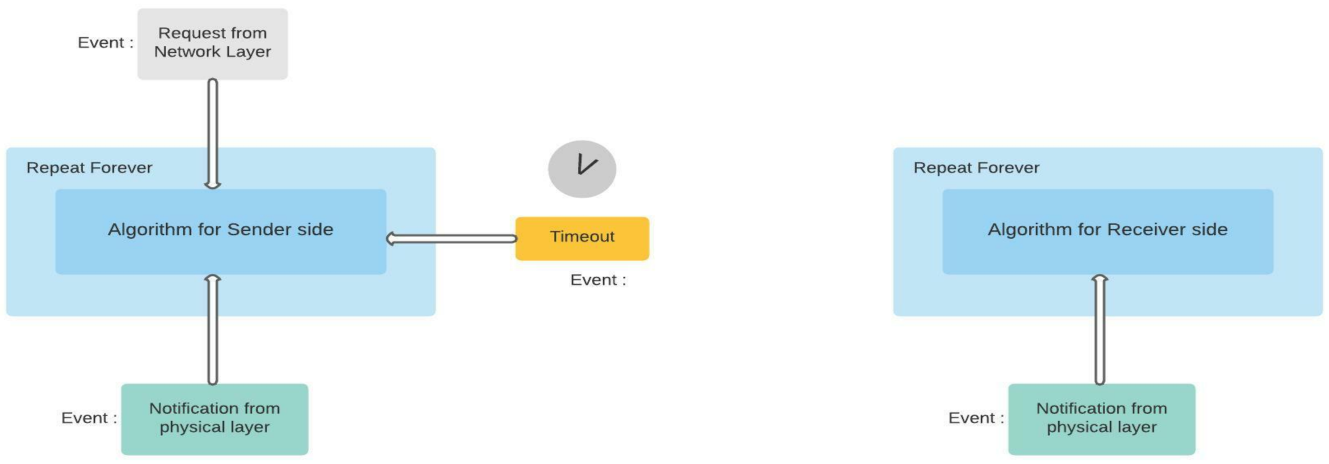

Event :
Request from Network Layer
Repeat Forever
Algorithm for Sender side
Timeout
Event :
Notification from physical layer
Algorithm for Receiver side

Fig : DESIGN OF SELECTIVE REPEAT SLIDING WINDOW PROTOCOL:

# IMPLEMENTATION

## 1. STOP AND WAIT PROTOCOL

### A. Code Snippet for Sender

```
import sys
import time
import socket

def create_frame(data):
    count_ones = 0
    for ch in data:
        if ch == '1': count_ones += 1
    data += str(count_ones % 2)
    return data

def Main(senderno):
    print('Initiating Sender #', senderno)
    host = '127.0.0.1'
    port = 8080
    sender_side_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sender_side_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sender_side_socket.connect((host, port))

    while True:
        print()
        data = input("Enter $ ")
        prevtime = time.time()
        data = create_frame(data)
        print('Sending to channel :', str(data))
        sender_side_socket.send(data.encode())
        if not data: break
        if data == 'q0': break
        rdata = sender_side_socket.recv(1024).decode('utf-8')
        print('Received from channel :', str(rdata))
        curtime = time.time()
        print('Round trip time: ', str(curtime-prevtime))
        if curtime-prevtime > 2: timeout = 1
        else: timeout = 0
        with open('checktime.txt', 'w') as fileout: fileout.write(str(timeout))

        while timeout == 1:
            print()
            prevtime = time.time()
            if timeout == 1: print('TIMEOUT of 2s EXPIRED !!')
            else: print('THE FRAME GOT CORRUPTED !!!')
            print('Again Sending to channel :', str(data))
            sender_side_socket.send(data.encode())
            rdata = sender_side_socket.recv(1024).decode()
            print('Again Received from channel :', str(rdata))
            curtime = time.time()
            print('Round trip time:', str(curtime-prevtime), 'seconds')
            if curtime-prevtime > 2: timeout = 1
            else: timeout = 0
            with open('checktime.txt', 'w') as fileout: fileout.write(str(timeout))

    sender_side_socket.close()
```

```
if __name__ == '__main__':
    if len(sys.argv) > 1: senderno = int(sys.argv[1])
    else: senderno = 1
    Main(senderno)
```

## B. Code Snippet for Channel

```
import os
import sys
import time
import socket
import random


def inject_random_error(frame):
    pos = random.randint(0, len(frame)-1)
    frame = frame[:pos]+'1'+frame[pos+1:]
    return frame


class Channel():

    def __init__(self, totalsender, totalreceiver):
        self.totalsender = totalsender
        self.senderhost = '127.0.0.1'
        self.senderport = 8080
        self.senderconn = []

        self.totalreceiver = totalreceiver
        self.receiverhost = '127.0.0.2'
        self.receiverport = 9090
        self.receiverconn = []


    def initialize_senders(self):
        sender_side_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sender_side_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        sender_side_socket.bind((self.senderhost, self.senderport))
        sender_side_socket.listen(self.totalsender)
        for _ in range(1, self.totalsender+1):
            conn = sender_side_socket.accept()
            self.senderconn.append(conn)
        print('Initiated all sender connections')


    def terminate_senders(self):
        for conn in self.senderconn: conn[0].close()
        print('Closed all sender connections')


    def initialize_receivers(self):
        receiver_side_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        receiver_side_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        receiver_side_socket.bind((self.receiverhost, self.receiverport))
        receiver_side_socket.listen(self.totalreceiver)
        for _ in range(1, self.totalreceiver+1):
            conn = receiver_side_socket.accept()
            self.receiverconn.append(conn)
        print('Initiated all receiver connections')


    def terminate_receivers(self):
        for conn in self.receiverconn: conn[0].close()
        print('Closed all receiver connections')

    def process_data(self):
        while True:
            for i in range(len(self.senderconn)):
                print()
                conn = self.senderconn[i]
                data = conn[0].recv(1024).decode('utf-8')
                if not data: break
                if data == 'q0': break
                print('Received from Sender', i+1, ':', str(data))

                recvno = random.randint(0, len(self.receiverconn)-1)
                print('Sending to Receiver', recvno+1)
                rconn = self.receiverconn[recvno]
                data = inject_random_error(data)
                rconn[0].sendto(data.encode(), rconn[1])

                # received from receiver
                rdata = rconn[0].recv(1024).decode('utf-8')
                print('Received from Receiver', recvno+1, ':', str(rdata))

                print('Sending to Sender', i+1)
                conn[0].send(rdata.encode())

                time.sleep(0.002)
                try:
                    with open('checktime.txt', 'r') as filein:
                        timeout = int(filein.read())
                    os.remove('checktime.txt')
                except:
                    print('Timeout -->>', timeout)
                    break

                while timeout == 1:
                    print()
                    data = conn[0].recv(1024).decode('utf-8')
                    print('Again Received from Sender', i+1, ':', str(data))
                    data = inject_random_error(data)
                    print('Again Sending to Receiver', recvno+1)
                    rconn[0].sendto(data.encode(), rconn[1])
                    rdata = rconn[0].recv(1024).decode('utf-8')
                    print('Again Received from Receiver', recvno+1, ':', str(rdata))
                    print('Again Sending to Sender', i+1)
                    conn[0].send(rdata.encode())

                    time.sleep(0.002)
                    try:
                        with open('checktime.txt', 'r') as filein:
                            timeout = int(filein.read())
                        os.remove('checktime.txt')
                    except:
                        print('Timeout -->>', timeout)
                        break
                    print('Timeout -->>', timeout)

            if data == 'q0':
                break
        return
```

```
if __name__ == '__main__':
    totalsen = int(input('Enter number of senders: '))
    totalrecv = int(input('Enter number of receivers: '))

    ch = Channel(totalsen, totalrecv)
    ch.initialize_senders()
    ch.initialize_receivers()
    ch.process_data()
    ch.terminate_senders()
    ch.terminate_receivers()
```

## C. Code Snippet for Receiver

```
import sys
import time
import socket
import random


def wait_random_time():
    x = random.randint(0, 5)
    if x <= 1: time.sleep(2)

def check_error(frame):
    count_ones = 0
    for ch in frame:
        if ch == '1': count_ones += 1
    return count_ones % 2

def Main(senderno):
    print('Initiating Receiver #', senderno)
    host = '127.0.0.2'
    port = 9090
    receiver_side_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    receiver_side_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    receiver_side_socket.connect((host, port))

    while True:
        print()
        data = receiver_side_socket.recv(1024).decode('utf-8')
        if not data: break
        if data == 'q0': break
        print('Received from channel :', str(data))
        wait_random_time()
        if check_error(data) == 0: rdata = 'ACK'
        else:
            time.sleep(2)
            rdata = 'TIMEOUT'
        print('Sending to channel :', str(rdata))
        receiver_side_socket.send(rdata.encode())

    receiver_side_socket.close()


if __name__ == '__main__':
    if len(sys.argv) > 1: senderno = int(sys.argv[1])
    else: senderno = 1
    Main(senderno)
```

# 2. GO BACK N SLIDING WINDOW PROTOCOL

## A. Code Snippet for Sender

```
import sys
import socket

def create_frame(data):
    count_ones = 0
    for ch in data:
        if ch == '1': count_ones += 1
    data += str(count_ones % 2)
    return data

def extract_message(frame):
    endidx = -1
    for i in range(len(frame)-1):
        if frame[i] == '/' and endidx == -1:
            endidx = i
            break
    return frame[:endidx]

def extract_count(frame):
    startidx = -1
    endidx = -1
    for i in range(len(frame)-1):
        if frame[i] == '/':
            if startidx == -1: startidx = i+1
            else: endidx = i
    cnt = frame[startidx:endidx]
    return int(cnt)

def extract_status(frame):
    count = 0
    startidx = -1
    for i in range(len(frame)-1):
        if frame[i] == '/': count += 1
        if count == 2 and startidx == -1:
            startidx = i+1
            break
    return frame[startidx:]

def Main(senderno):
    count = 0
    sent_frames = []
    print('Initiating Sender #', senderno)
    host = '127.0.0.1'
    port = 8080
    sender_side_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sender_side_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sender_side_socket.connect((host, port))
    while True:
        print()
        data = input("Enter $ ")
        data = create_frame(data) + '/' + str(count) + '/'
        msg = extract_message(data)
        print('Sending to channel :', str(msg))
        sender_side_socket.send(data.encode())
        sent_frames.append(data)
        count += 1
        if not msg: break
        if msg == 'q0': break
    sender_side_socket.close()
```

```
        if __name__ == '__main__':
            if len(sys.argv) > 1: senderno = int(sys.argv[1])
            else: senderno = 1
            Main(senderno)
```

## B. Code Snippet for Channel

```
import time
import socket
import random


def inject_random_error(frame):
    pos = random.randint(0, len(frame)-1)
    frame = frame[:pos]+'1'+frame[pos+1:]
    return frame

def extract_message(frame):
    endidx = -1
    for i in range(len(frame)-1):
        if frame[i] == '/' and endidx == -1:
            endidx = i
            break
    return frame[:endidx]

def extract_count(frame):
    startidx = -1
    endidx = -1
    for i in range(len(frame)-1):
        if frame[i] == '/':
            if startidx == -1: startidx = i+1
            else: endidx = i
    cnt = frame[startidx:endidx]
    return int(cnt)

def extract_status(frame):
    count = 0
    startidx = -1
    for i in range(len(frame)-1):
        if frame[i] == '/': count += 1
        if count == 2 and startidx == -1:
            startidx = i+1
            break
    return frame[startidx:]

class Channel():

    def __init__(self, totalsender, totalreceiver, windowsize):
        self.totalsender = totalsender
        self.senderhost = '127.0.0.1'
        self.senderport = 8080
        self.senderconn = []

        self.totalreceiver = totalreceiver
        self.receiverhost = '127.0.0.2'
        self.receiverport = 9090
        self.receiverconn = []

        self.windowsize = windowsize
        self.slidingwindow = []
        self.currentcount = 0

    def initialize_senders(self):
        sender_side_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sender_side_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        sender_side_socket.bind((self.senderhost, self.senderport))
        sender_side_socket.listen(self.totalsender)
        for _ in range(1, self.totalsender+1):
            conn = sender_side_socket.accept()
            self.senderconn.append(conn)
        print('Initiated all sender connections')

    def terminate_senders(self):
        for conn in self.senderconn: conn[0].close()
        print('Closed all sender connections')

    def initialize_receivers(self):
        receiver_side_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        receiver_side_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        receiver_side_socket.bind((self.receiverhost, self.receiverport))
        receiver_side_socket.listen(self.totalreceiver)
        for _ in range(1, self.totalreceiver+1):
            conn = receiver_side_socket.accept()
            self.receiverconn.append(conn)
        print('Initiated all receiver connections')

    def terminate_receivers(self):
        for conn in self.receiverconn: conn[0].close()
        print('Closed all receiver connections')

    def process_data(self):
        while True:
            for i in range(len(self.senderconn)):
                print()
                conn = self.senderconn[i]
                data = conn[0].recv(1024).decode('utf-8')
                prevtime = time.time()
                data = str(data)
                origmsg = extract_message(data)
                if not origmsg: break
                if origmsg == 'q0': break
                print('Received from Sender', i+1, ':', str(data))

                recvno = random.randint(0, len(self.receiverconn)-1)
                print('Sending to Receiver', recvno+1)
                rconn = self.receiverconn[recvno]
                cnt = extract_count(data)
                msg = inject_random_error(origmsg)
                newdata = msg + '/' + str(cnt) + '/'
                rconn[0].sendto(newdata.encode(), rconn[1])

                # received from receiver
                rdata = rconn[0].recv(1024).decode('utf-8')
                rdata = str(rdata)
                time.sleep(0.5)
                curtime = time.time()
                if curtime-prevtime > 2: newdata += 'TIMEOUT'
                else: newdata += rdata
                self.slidingwindow.append([data, newdata, i, recvno])

                msg = extract_message(newdata)
                cnt = extract_count(newdata)
                status = extract_status(newdata)
                print(msg, str(cnt), status)
                print('Round trip time: ', str(curtime-prevtime))
                print('Current frame no:', str((self.currentcount % windowsize)+1))
                if (self.currentcount % windowsize)+1 == self.windowsize:
                    idx = 0
                    flag = 1
                    while flag == 1:
                        idx = 0
                        flag = 0
                        while idx < self.windowsize:
                            currframe = self.slidingwindow[idx][1]
                            msg = extract_message(currframe)
                            cnt = extract_count(currframe)
                            status = extract_status(currframe)

                            if status == 'NAK' or status == 'TIMEOUT':
                                flag = 1
                                break
                            idx += 1

                        print(' ------------------------------ ')
                        if flag == 1: print('RESEND FROM FRAME NO:', str(idx+1))
                        else: print('BLOCK OF WINDOW SIZE', self.windowsize,'SUCCESSFULLY SENT')
                        print(' ------------------------------ ')

                        while flag == 1 and idx < self.windowsize:
                            print()
                            prevtime = time.time()
                            prevframe = self.slidingwindow[idx][0]
                            currframe = self.slidingwindow[idx][1]
                            sendno = self.slidingwindow[idx][2]
                            recvno = self.slidingwindow[idx][3]
                            conn = self.senderconn[sendno]
                            rconn = self.receiverconn[recvno]

                            # sending all frames to its sender from first NAK
                            print('Current frame no:', str(idx+1))
                            print('Again Sending to Receiver', recvno+1)

                            msg = extract_message(prevframe)
                            msg = inject_random_error(msg)
                            data = msg + '/' + str(cnt) + '/'
                            rconn[0].sendto(data.encode(), rconn[1])

                            # receiving ACK or NAK from receiver
                            rdata = rconn[0].recv(1024).decode('utf-8')
                            rdata = str(rdata)
                            data += rdata

                            msg = extract_message(data)
                            cnt = extract_count(data)
                            stat = extract_status(data)
                            curtime = time.time()
                            print(msg, str(cnt), stat)
                            print('Round trip time: ', str(curtime-prevtime))
                            self.slidingwindow[idx][1] = data
                            idx += 1

                self.currentcount += 1
            if origmsg == 'q0':
                break
        return
```

```
if __name__ == '__main__':
    totalsen = int(input('Enter number of senders: '))
    totalrecv = int(input('Enter number of receivers: '))
    windowsize = int(input('Enter window size: '))

    ch = Channel(totalsen, totalrecv, windowsize)
    ch.initialize_senders()
    ch.initialize_receivers()
    ch.process_data()
    ch.terminate_senders()
    ch.terminate_receivers()
```

## D. Code Snippet for Receiver

```
import sys
import time
import socket
import random


def wait_random_time():
    x = random.randint(0, 5)
    if x <= 1: time.sleep(2)

def check_error(frame):
    count_ones = 0
    for ch in frame:
        if ch == '1': count_ones += 1
    return count_ones % 2

def extract_message(frame):
    endidx = -1
    for i in range(len(frame)-1):
        if frame[i] == '/' and endidx == -1:
            endidx = i
            break
    return frame[:endidx]

def extract_count(frame):
    startidx = -1
    endidx = -1
    for i in range(len(frame)-1):
        if frame[i] == '/':
            if startidx == -1: startidx = i+1
            else: endidx = i
    cnt = frame[startidx:endidx]
    return int(cnt)

def extract_status(frame):
    count = 0
    startidx = -1
    for i in range(len(frame)-1):
        if frame[i] == '/': count += 1
        if count == 2 and startidx == -1:
            startidx = i+1
            break
    return frame[startidx:]
```

```
def Main(senderno):
    print('Initiating Receiver #', senderno)
    host = '127.0.0.2'
    port = 9090
    receiver_side_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    receiver_side_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    receiver_side_socket.connect((host, port))

    while True:
        print()
        data = receiver_side_socket.recv(1024).decode('utf-8')
        msg = extract_message(data)
        if not msg:  break
        if msg == 'q0':  break
        print('Received from channel :', str(data))
        wait_random_time()
        if check_error(msg) == 0: rdata = 'ACK'
        else: rdata = 'NAK'
        print('Sending to channel :', str(rdata))
        receiver_side_socket.send(rdata.encode())

    receiver_side_socket.close()


if __name__ == '__main__':
    if len(sys.argv) > 1: senderno = int(sys.argv[1])
    else: senderno = 1
    Main(senderno)
```

## 3. SELECTIVE REPEAT SLIDING WINDOW PROTOCOL

### A. Code Snippet for Sender

```
import sys
import socket


def create_frame(data):
    count_ones = 0
    for ch in data:
        if ch == '1': count_ones += 1
    data += str(count_ones % 2)
    return data

def extract_message(frame):
    endidx = -1
    for i in range(len(frame)-1):
        if frame[i] == '/' and endidx == -1:
            endidx = i
            break
    return frame[:endidx]

def extract_count(frame):
    startidx = -1
    endidx = -1
    for i in range(len(frame)-1):
        if frame[i] == '/':
            if startidx == -1: startidx = i+1
            else: endidx = i
    cnt = frame[startidx:endidx]
    return int(cnt)

def extract_status(frame):
    count = 0
    startidx = -1
    for i in range(len(frame)-1):
        if frame[i] == '/': count += 1
        if count == 2 and startidx == -1:
            startidx = i+1
            break
    return frame[startidx:]

def Main(senderno):
    count = 0
    sent_frames = []
    print('Initiating Sender #', senderno)
    host = '127.0.0.1'
    port = 8080
    sender_side_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sender_side_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sender_side_socket.connect((host, port))

    while True:
        print()
        data = input("Enter $ ")
        data = create_frame(data) + '/' + str(count) + '/'
        msg = extract_message(data)
        print('Sending to channel :', str(msg))
        sender_side_socket.send(data.encode())
        sent_frames.append(data)
        count += 1
        if not msg: break
        if msg == 'q0': break

    sender_side_socket.close()


if __name__ == '__main__':
    if len(sys.argv) > 1: senderno = int(sys.argv[1])
    else: senderno = 1
    Main(senderno)
```

## B. Code Snippet for Channel

```
import time
import socket
import random


def inject_random_error(frame):
        pos = random.randint(0, len(frame)-1)
        frame = frame[:pos]+'1'+frame[pos+1:]
        return frame

def extract_message(frame):
        endidx = -1
        for i in range(len(frame)-1):
                if frame[i] == '/' and endidx == -1:
                        endidx = i
                        break
        return frame[:endidx]

def extract_count(frame):
        startidx = -1
        endidx = -1
        for i in range(len(frame)-1):
                if frame[i] == '/':
                        if startidx == -1: startidx = i+1
                        else: endidx = i
        cnt = frame[startidx:endidx]
        return int(cnt)

def extract_status(frame):
        count = 0
        startidx = -1
        for i in range(len(frame)-1):
                if frame[i] == '/': count += 1
                if count == 2 and startidx == -1:
                        startidx = i+1
                        break
        return frame[startidx:]

class Channel():

        def __init__(self, totalsender, totalreceiver, windowsize):
                self.totalsender = totalsender
                self.senderhost = '127.0.0.1'
                self.senderport = 8080
                self.senderconn = []

                self.totalreceiver = totalreceiver
                self.receiverhost = '127.0.0.2'
                self.receiverport = 9090
                self.receiverconn = []

                self.windowsize = windowsize
                self.slidingwindow = []
                self.currentcount = 0

        def initialize_senders(self):
                sender_side_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                sender_side_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
                sender_side_socket.bind((self.senderhost, self.senderport))
                sender_side_socket.listen(self.totalsender)
                for _ in range(1, self.totalsender+1):
                        conn = sender_side_socket.accept()
                        self.senderconn.append(conn)
                print('Initiated all sender connections')

        def terminate_senders(self):
                for conn in self.senderconn: conn[0].close()
                print('Closed all sender connections')

        def initialize_receivers(self):
                receiver_side_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                receiver_side_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
                receiver_side_socket.bind((self.receiverhost, self.receiverport))
                receiver_side_socket.listen(self.totalreceiver)
                for _ in range(1, self.totalreceiver+1):
                        conn = receiver_side_socket.accept()
                        self.receiverconn.append(conn)
                print('Initiated all receiver connections')

        def terminate_receivers(self):
                for conn in self.receiverconn: conn[0].close()
                print('Closed all receiver connections')

        def process_data(self):
                while True:
                        for i in range(len(self.senderconn)):
                        print()
                        conn = self.senderconn[i]
                        data = conn[0].recv(1024).decode('utf-8')
                        prevtime = time.time()
                        data = str(data)
                        origmsg = extract_message(data)
                        if not origmsg: break
                        if origmsg == 'q0': break
                        print('Received from Sender',i+1,':',str(data))

                        recvno = random.randint(0,len(self.receiverconn)-1)
                        print('Sending to Receiver',recvno+1)
                        rconn = self.receiverconn[recvno]
                        cnt = extract_count(data)
                        msg = inject_random_error(origmsg)
                        newdata = msg + '/' + str(cnt) + '/'
                        rconn[0].sendto(newdata.encode(), rconn[1])

                        #received from receiver
                        rdata = rconn[0].recv(1024).decode()
                        rdata = str(rdata)
                        time.sleep(0.5)
                        curtime = time.time()
                        if curtime-prevtime > 2: newdata += 'TIMEOUT'
                        else: newdata += rdata
                        self.slidingwindow.append([data, newdata, i, recvno])

                        msg = extract_message(newdata)
                        cnt = extract_count(newdata)
                        status = extract_status(newdata)
                        print(msg,str(cnt),status)
                        print('Round trip time: ',str(curtime-prevtime))
                        print('Current frame no:',str((self.currentcount % windowsize)+1))
                        if (self.currentcount % windowsize)+1 == self.windowsize:
                            idx = 0
                            flag = 1
                            while flag == 1:
                                idx = 0
                                flag = 0
                                nakframes = []
                                indices = []
                                while idx < self.windowsize:
                                    currframe = self.slidingwindow[idx][1]
                                    msg = extract_message(currframe)
                                    cnt = extract_count(currframe)
                                    status = extract_status(currframe)

                                    if status == 'NAK' or status == 'TIMEOUT':
                                        flag = 1
                                        nakframes.append(self.slidingwindow[idx])
                                        indices.append(idx+1)
                                    idx += 1

                                if flag==0:
                                    print(' ------------------------------ ')
                                    print('BLOCK OF WINDOW SIZE',self.windowsize,
                                                             'SUCCESSFULLY SENT')
                                    print(' ------------------------------ ')

                                idx = 0
                                while flag == 1 and idx < len(nakframes):
                                    print()
                                    prevtime = time.time()
                                    prevframe = nakframes[idx][0]
                                    currframe = nakframes[idx][1]
                                    sendno = nakframes[idx][2]
                                    recvno = nakframes[idx][3]
                                    conn = self.senderconn[sendno]
                                    rconn = self.receiverconn[recvno]
                                    stat = extract_status(currframe)

                                    print(' ------------------------------ ')
                                    print('RESENDING FRAME NO:',str(indices[idx]))
                                    print(' ------------------------------ ')

                                    #sending all frames to its sender from first NAK
                                    print('Current frame no:',str(indices[idx]))
                                    print('Again Sending to Receiver',recvno+1)

                                    msg = extract_message(prevframe)
                                    msg = inject_random_error(msg)
                                    data = msg + '/' + str(cnt) + '/'
                                    rconn[0].sendto(data.encode(), rconn[1])

                                    # receiving ACK or NAK from receiver
                                    rdata = rconn[0].recv(1024).decode()
                                    rdata = str(rdata)
                                    data += rdata

                                    msg = extract_message(data)
                                    cnt = extract_count(data)
                                    stat = extract_status(data)
                                    curtime = time.time()
                                    print(msg,str(cnt),stat)
                                    print('Round trip time: ',str(curtime-prevtime))
                                    self.slidingwindow[indices[idx]-1][1] = data
                                    idx += 1

                        self.currentcount += 1
                        if origmsg == 'q0':
                        break
                return


if __name__ == '__main__':
    totalsen = int(input('Enter number of senders: '))
    totalrecv = int(input('Enter number of receivers: '))
    windowsize =int(input('Enter window size: '))

    ch = Channel(totalsen, totalrecv, windowsize)
    ch.initialize_senders()
    ch.initialize_receivers()
    ch.process_data()
    ch.terminate_senders()
    ch.terminate_receivers()
```

## C. Code Snippet for Receiver

```
import sys
import time
import socket
import random

def wait_random_time():
    x = random.randint(0, 5)
    if x <= 1: time.sleep(2)

def check_error(frame):
    count_ones = 0
    for ch in frame:
        if ch == '1': count_ones += 1
    return count_ones % 2

def extract_message(frame):
    endidx = -1
    for i in range(len(frame)-1):
        if frame[i] == '/' and endidx == -1:
            endidx = i
            break
    return frame[:endidx]

def extract_count(frame):
    startidx = -1
    endidx = -1
    for i in range(len(frame)-1):
        if frame[i] == '/':
            if startidx == -1: startidx = i+1
            else: endidx = i
    cnt = frame[startidx:endidx]
    return int(cnt)

def extract_status(frame):
    count = 0
    startidx = -1
    for i in range(len(frame)-1):
        if frame[i] == '/': count += 1
        if count == 2 and startidx == -1:
            startidx = i+1
            break
    return frame[startidx:]

def Main(senderno):
    print('Initiating Receiver #', senderno)
    host = '127.0.0.2'
    port = 9090
    receiver_side_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    receiver_side_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    receiver_side_socket.connect((host, port))
    while True:
        print()
        data = receiver_side_socket.recv(1024).decode('utf-8')
        msg = extract_message(data)
        if not msg:  break
        if msg == 'q0':  break
        print('Received from channel :', str(data))
        wait_random_time()
        if check_error(msg) == 0: rdata = 'ACK'
        else: rdata = 'NAK'
        print('Sending to channel :', str(rdata))
        receiver_side_socket.send(rdata.encode())

    receiver_side_socket.close()
```

```
if __name__ == '__main__':
    if len(sys.argv) > 1: senderno = int(sys.argv[1])
    else: senderno = 1
    Main(senderno)
```

## SOURCE CODE STRUCTURE

The SourceCode Folder contains three sub-folders - **./stopnwait**, **./gobackn** and **./selectiverepeat**.

### **./stopnwait** contains -

1. `sender.py`
2. `channel.py`
3. `receiver.py`
4. `checktime.txt`

### **./gobackn** contains -

1. `sender.py`
2. `channel.py`
3. `receiver.py`

### **./selectiverepeat** contains -

1. `sender.py`
2. `channel.py`
3. `receiver.py`

## TEST CASES

### Stop and Wait ARQ:

1. Channel.py:

C:\Users\Atanu\Documents\Network-Simulations\FlowControl\stopnwait>python channel.py

```
Enter number of senders: 2
Enter number of receivers: 2
Initiated all sender connections
Initiated all receiver connections

Again Received from Sender 1 : 11110
Again Sending to Receiver 1
Again Received from Receiver 1 : ACK
Again Sending to Sender 1
Timeout -->> 0

Received from Sender 2 : 11000
Sending to Receiver 2
Received from Receiver 2 : ACK
Sending to Sender 2

Again Received from Sender 2 : 11000
Again Sending to Receiver 2
Again Received from Receiver 2 : ACK
Again Sending to Sender 2
Timeout -->> 0
```

2. Sender.py (Sender no. 1):

C:\Users\Atanu\Documents\Network-Simulations\FlowControl\stopnwait>python sender.py 1

```
Initiating Sender # 1

Enter $ 1111
Sending to channel : 11110
Received from channel : ACK
Round trip time:  2.02493029774291

TIMEOUT of 2s EXPIRED !!
Again Sending to channel : 11110
Again Received from channel : ACK
Round trip time: 2.02193021774292 seconds

TIMEOUT of 2s EXPIRED !!
Again Sending to channel : 11110
Again Received from channel : ACK
```

```
Round trip time: 2.02347731590271 seconds

TIMEOUT of 2s EXPIRED !!
Again Sending to channel : 11110
Again Received from channel : ACK
Round trip time: 2.021240711212158 seconds

TIMEOUT of 2s EXPIRED !!
Again Sending to channel : 11110
Again Received from channel : ACK
Round trip time: 0.016021728515625 seconds
```

3. Sender.py (Sender no. 2):

C:\Users\Atanu\Documents\Network-Simulations\FlowControl\stopnwait>python sender.py 2

```
Initiating Sender # 2

Enter $ 1100
Sending to channel : 11000
Received from channel : ACK
Round trip time:  65.70393419265747

TIMEOUT of 2s EXPIRED !!
Again Sending to channel : 11000
Again Received from channel : ACK
Round trip time: 0.014997720718383789 seconds
```

4. Receiver.py (Receiver no. 1):

C:\Users\Atanu\Documents\Network-Simulations\FlowControl\stopnwait>python receiver.py 1

```
Initiating Receiver # 1

Received from channel : 11110
Sending to channel : ACK

Received from channel : 11111
Sending to channel : TIMEOUT

Received from channel : 11110
Sending to channel : ACK

Received from channel : 11110
Sending to channel : ACK

Received from channel : 11110
Sending to channel : ACK

Received from channel : 11110
```

```
Sending to channel : ACK
```

5. Receiver.py (Receiver no. 2):

C:\Users\Atanu\Documents\Network-Simulations\FlowControl\stopnwait>python receiver.py 2

```
Initiating Receiver # 2

Received from channel : 11000
Sending to channel : ACK

Received from channel : 11000
Sending to channel : ACK
```

## Go Back N ARQ:

1. Channel.py:

C:\Users\Atanu\Documents\Network-Simulations\FlowControl\gobackn>python channel.py

```
Enter number of senders: 2
Enter number of receivers: 2
Enter window size: 3
Initiated all sender connections
Initiated all receiver connections

Received from Sender 1 : 10010/0/
Sending to Receiver 1
10110 0 NAK
Round trip time:  0.5001943111419678
Current frame no: 1

Received from Sender 2 : 01001/0/
Sending to Receiver 2
01001 0 ACK
Round trip time:  0.5059702396392822
Current frame no: 2

Received from Sender 1 : 10100/1/
Sending to Receiver 1
10101 1 NAK
Round trip time:  0.5004317760467529
Current frame no: 3
 ------------------------------
RESEND FROM FRAME NO: 1
 ------------------------------

Current frame no: 1
Again Sending to Receiver 1
10110 0 NAK
```

```
Round trip time:  0.0

Current frame no: 2
Again Sending to Receiver 2
01001 0 ACK
Round trip time:  2.0019752979278564

Current frame no: 3
Again Sending to Receiver 1
10100 0 ACK
Round trip time:  0.0
 ------------------------------
RESEND FROM FRAME NO: 1
 ------------------------------

Current frame no: 1
Again Sending to Receiver 1
10110 0 NAK
Round trip time:  2.011181116104126

Current frame no: 2
Again Sending to Receiver 2
01001 0 ACK
Round trip time:  2.007286548614502

Current frame no: 3
Again Sending to Receiver 1
10110 0 NAK
Round trip time:  2.008167266845703
 ------------------------------
RESEND FROM FRAME NO: 1
Round trip time:  0.0

Current frame no: 3
Again Sending to Receiver 1
10110 0 NAK
Round trip time:  0.0
 ------------------------------
RESEND FROM FRAME NO: 3
 ------------------------------

Current frame no: 3
Again Sending to Receiver 1
10100 0 ACK
Round trip time:  0.0
 ------------------------------
BLOCK OF WINDOW SIZE 3 SUCCESSFULLY SENT
 ------------------------------
```

2. Sender.py (Sender no. 1)

```
C:\Users\Atanu\Documents\Network-Simulations\FlowControl\gobackn>python sender.py 1
Initiating Sender # 1

Enter $ 1001
Sending to channel : 10010

Enter $ 1010
Sending to channel : 10100
```

3. Sender.py (Sender no. 2)

```
C:\Users\Atanu\Documents\Network-Simulations\FlowControl\gobackn>python sender.py 2
Initiating Sender # 2

Enter $ 0100
Sending to channel : 01001
```

4. Receiver.py (Receiver no. 1)

```
C:\Users\Atanu\Documents\Network-Simulations\FlowControl\gobackn>python receiver.py 1
Initiating Receiver # 1

Received from channel : 10110/0/
Sending to channel : NAK

Received from channel : 10101/1/
Sending to channel : NAK

Received from channel : 10110/0/
Sending to channel : NAK

Received from channel : 10100/0/
Sending to channel : ACK

Received from channel : 10110/0/
Sending to channel : NAK

Received from channel : 10110/0/
Sending to channel : NAK

Received from channel : 10110/0/
Sending to channel : NAK

Received from channel : 10100/0/
Sending to channel : ACK
```

```
Received from channel : 10010/0/
Sending to channel : ACK

Received from channel : 10110/0/
Sending to channel : NAK

Received from channel : 10100/0/
Sending to channel : ACK
```

5. Receiver.py (Receiver no. 2)

```
C:\Users\Atanu\Documents\Network-Simulations\FlowControl\gobackn>python receiver.py 2
```

```
Initiating Receiver # 2

Received from channel : 01001/0/
Sending to channel : ACK

Received from channel : 01001/0/
Sending to channel : ACK

Received from channel : 01001/0/
Sending to channel : ACK

Received from channel : 01001/0/
Sending to channel : ACK

Received from channel : 01001/0/
Sending to channel : ACK
```

## Selective Repeat ARQ:

1. Channel.py:

```
C:\Users\Atanu\Documents\Network-Simulations\FlowControl\selectiverepeat>python channel.py
```

```
Enter number of senders: 2
Enter number of receivers: 2
Enter window size: 3
Initiated all sender connections
Initiated all receiver connections

Received from Sender 1 : 11110/0/
Sending to Receiver 2
11110 0 TIMEOUT
Round trip time:  2.501133918762207
Current frame no: 1
```

```
Received from Sender 2 : 11000/0/
Sending to Receiver 2
11010 0 TIMEOUT
Round trip time:  2.5159244537353516
Current frame no: 2

Received from Sender 1 : 10100/1/
Sending to Receiver 2
11100 1 NAK
Round trip time:  0.5006747245788574
Current frame no: 3

 ------------------------------
RESENDING FRAME NO: 1
 ------------------------------
Current frame no: 1
Again Sending to Receiver 2
11110 1 ACK
Round trip time:  0.0

 ------------------------------
RESENDING FRAME NO: 2
 ------------------------------
Current frame no: 2
Again Sending to Receiver 2
11000 1 ACK
Round trip time:  2.0022072792053223

 ------------------------------
RESENDING FRAME NO: 3
 ------------------------------
Current frame no: 3
Again Sending to Receiver 2
10101 1 NAK
Round trip time:  0.0

 ------------------------------
RESENDING FRAME NO: 3
 ------------------------------
Current frame no: 3
Again Sending to Receiver 2
10100 1 ACK
Round trip time:  2.0059573650360107
 ------------------------------
BLOCK OF WINDOW SIZE 3 SUCCESSFULLY SENT
 ------------------------------
```

2. Sender.py (Sender no. 1)

```
C:\Users\Atanu\Documents\Network-Simulations\FlowControl\selectiverepeat>python sender.py 1
Initiating Sender # 1

Enter $ 1111
Sending to channel : 11110

Enter $ 1010
Sending to channel : 10100
```

3. Sender.py (Sender no. 2)

```
C:\Users\Atanu\Documents\Network-Simulations\FlowControl\selectiverepeat>python sender.py 2
Initiating Sender # 2

Enter $ 1100
Sending to channel : 11000
```

4. Receiver.py (Receiver no. 1)

```
C:\Users\Atanu\Documents\Network-Simulations\FlowControl\selectiverepeat>python receiver.py 1
Initiating Receiver # 1
```

5. Receiver.py (Receiver no. 2)

```
C:\Users\Atanu\Documents\Network-Simulations\FlowControl\selectiverepeat>python receiver.py 2
Initiating Receiver # 2

Received from channel : 11110/0/
Sending to channel : ACK

Received from channel : 11010/0/
Sending to channel : NAK

Received from channel : 11100/1/
Sending to channel : NAK

Received from channel : 11110/1/
Sending to channel : ACK

Received from channel : 11000/1/
Sending to channel : ACK

Received from channel : 10101/1/
Sending to channel : NAK

Received from channel : 10100/1/
Sending to channel : ACK
```

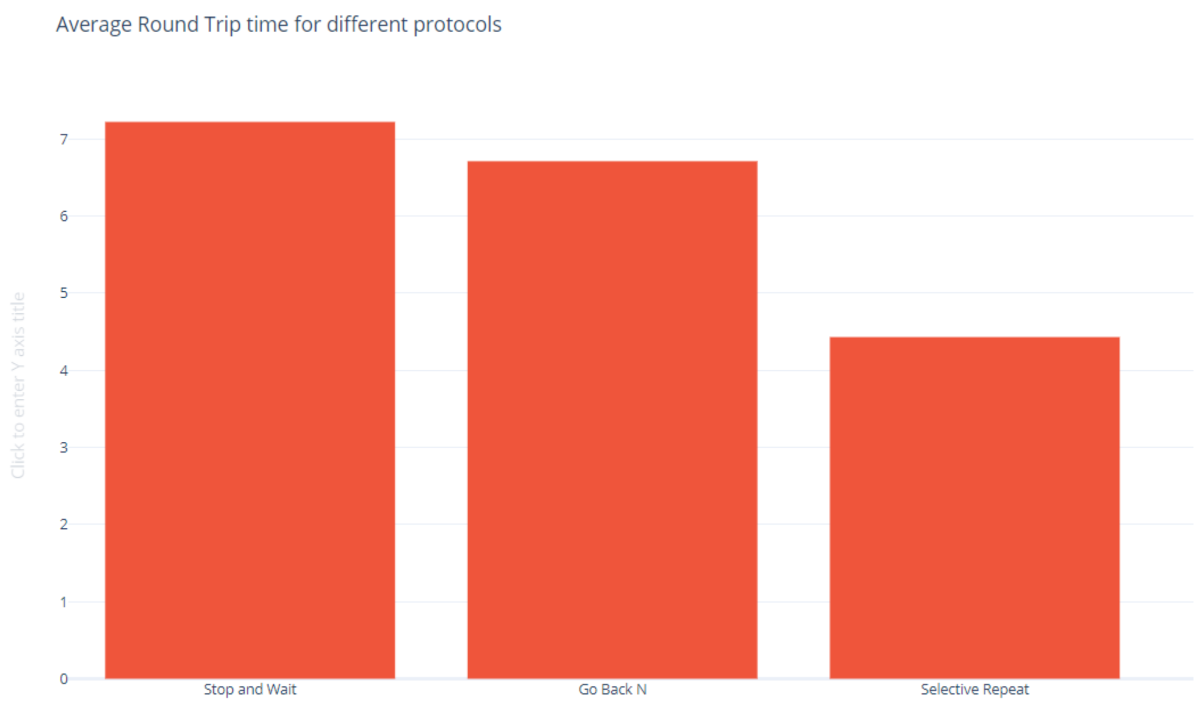
## RESULTS

Number of Total Frames required to Send a Long Input Data

| Name of Protocol | Number of frames sent approximately | Number of frames required if only delay is inserted | Number of frames required if both delay, dropout error introduced |
|---|---|---|---|
| Stop and Wait | 50 | 77 | 133 |
| Go Back N | 50 | 81 | 129 |
| Selective Repeat | 50 | 60 | 95 |

Time Taken to Send a Long Input Data

| Name of Protocol | Number of frames sent approximately | Time Taken (in Minutes) |
|---|---|---|
| Stop and Wait | 50 | 7.22 |
| Go Back N | 50 | 6.71 |
| Selective Repeat | 50 | 4.43 |



## ANALYSIS and CONCLUSION

### Stop and Wait ARQ:

Stop and Wait ARQ adds a simple error control mechanism to the Stop and Wait protocol for noiseless channels. To detect and correct corrupted frames, we need to add redundancy bits to our data frame. When the frame arrives at the receiver site, it is checked if it is corrupted.

The corrupted and lost frames need to be present in this protocol. If the receiver does not respond when there is an error, the sender keeps a copy of the sent frame. At the same time, it starts a timer. If the timer expires and there is no ACK for the sent frame, the frame is resent, the copy is held, and the timer is restarted.

The Stop and Wait ARQ is very inefficient if our channel is **thick** and **long**. By thick, we mean that our channel has a large bandwidth; by long, we mean the round-trip delay is long. The product of the two is called the bandwidth-delay product. It is the volume of the pipe in bits.

### Go Back N ARQ:

To improve the efficiency of transmission (filling the pipe), multiple frames are in transition while waiting for acknowledgment. In this protocol, we can send several frames before receiving acknowledgments; we keep a copy of these frames until acknowledgments arrive.

In this protocol, when the timer expires, the sender resends all the outstanding frames. For example, suppose the sender has already sent frame 6, but the timer for frame 3 expires. This means that frame 3 has not been acknowledged; the sender goes back and sends frames 3, 4, 5, and 6 again.

### Selective Repeat ARQ:

Go Back N ARQ simplifies the process at the receiver site. The receiver keeps track of only one variable, and there is no need to buffer out-of-order frames; they are simply discarded. However, it is very inefficient for a noisy link. In a noisy link a frame has a higher probability of damage, which means the resending of multiple frames. This resending uses up the bandwidth and slows down the transmission.

For noisy links, there is another mechanism that does not resend N frames when just one frame is damaged; only the damaged frame is resent. This mechanism is called Selective Repeat ARQ. It is more efficient for noisy links, but the processing at the receiver is more complex.

## COMMENTS

This assignment has helped me in understanding the different data link layer protocols immensely, by researching and implementing them. It has also helped in understanding the demerits of a protocol, and how such demerits are overcome by other protocols.