

# Computer Networks Lab Report- Assignment 2

TITLE:

Name: Debarghya Maitra

Class: BCSE 3 rd Year

Group: A3

Submission Date: 28/8/2022

PROBLEM STATEMENT: Implement three data link layer protocols, Stop and Wait, Go Back N Sliding Window and Selective Repeat Sliding Window for flow control.

DESIGN:

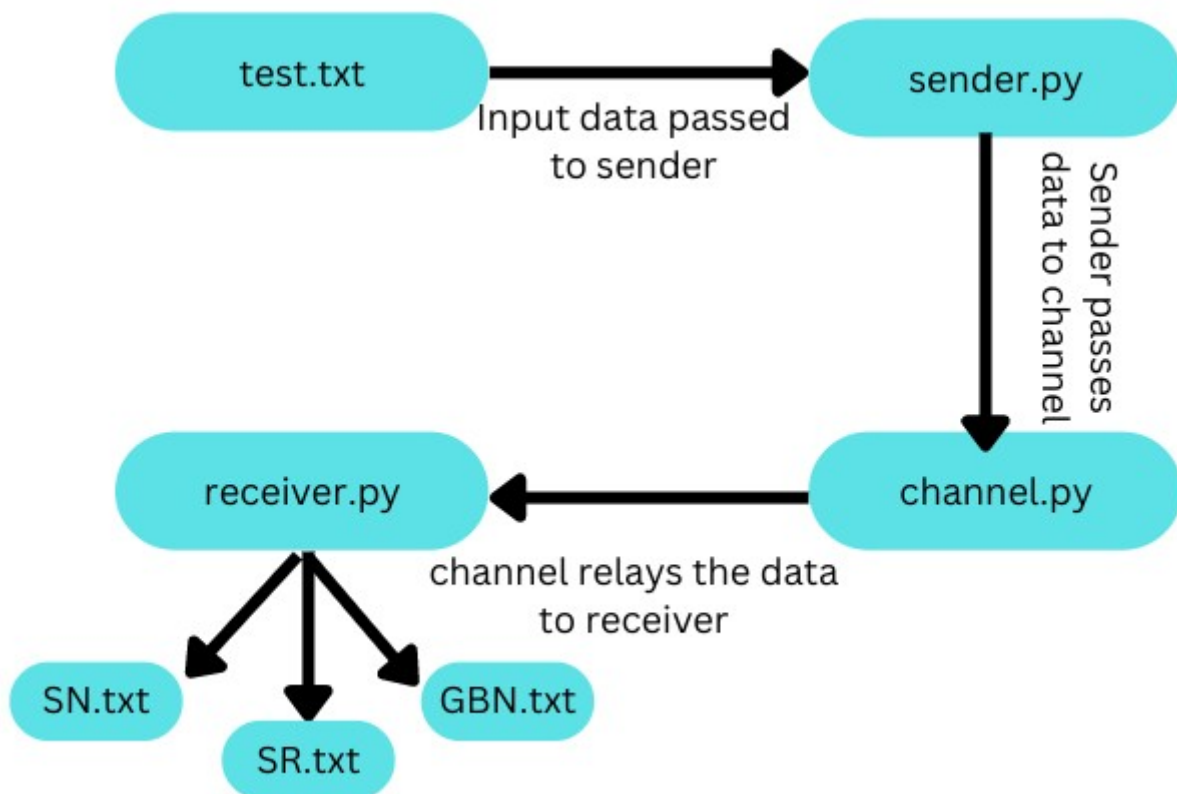


Fig: Design structure for the implementation

Design Roles for different modules:

1. **Sender:** Reads the data from file test.txt and Converts the data into packet following the IEEE 802.3 Frame Format (using the module packet manager). It starts three threads listening to the same port 1232. First one is responsible for sending the packet to the channel. Second to receive any acknowledgement (ACK or NAK). Third, send the previous packets, if ACK (positive acknowledgement) is not received. The three processes are implemented using a lock class to prevent any type racing condition while listening to the same port.

2. **Channel:** Receives the data packets from the sender (s). Processes the packet and introduces random delay or error into the packets. It is designed in a way that it will send the packet with 65 % chance, introduce delay with 10 % chance, injects error into the packet with 15 % chance and drops the packet with a 10 % chance. Finally it will receive acknowledgement from the receiver and pass them to the sender.

**3.Receiver:** Receives data packets from the sender via the channel. Checks if the data is erroneous or not. If the data is error free it will send a positive acknowledgement to the channel. In sliding window protocols, if the packet ranges aren't valid at any moment, send a NAK. If the sequence number is still stuck at the last packet, resend the acknowledgement for the previous packet simultaneously.

#### 4. Packet Structure:

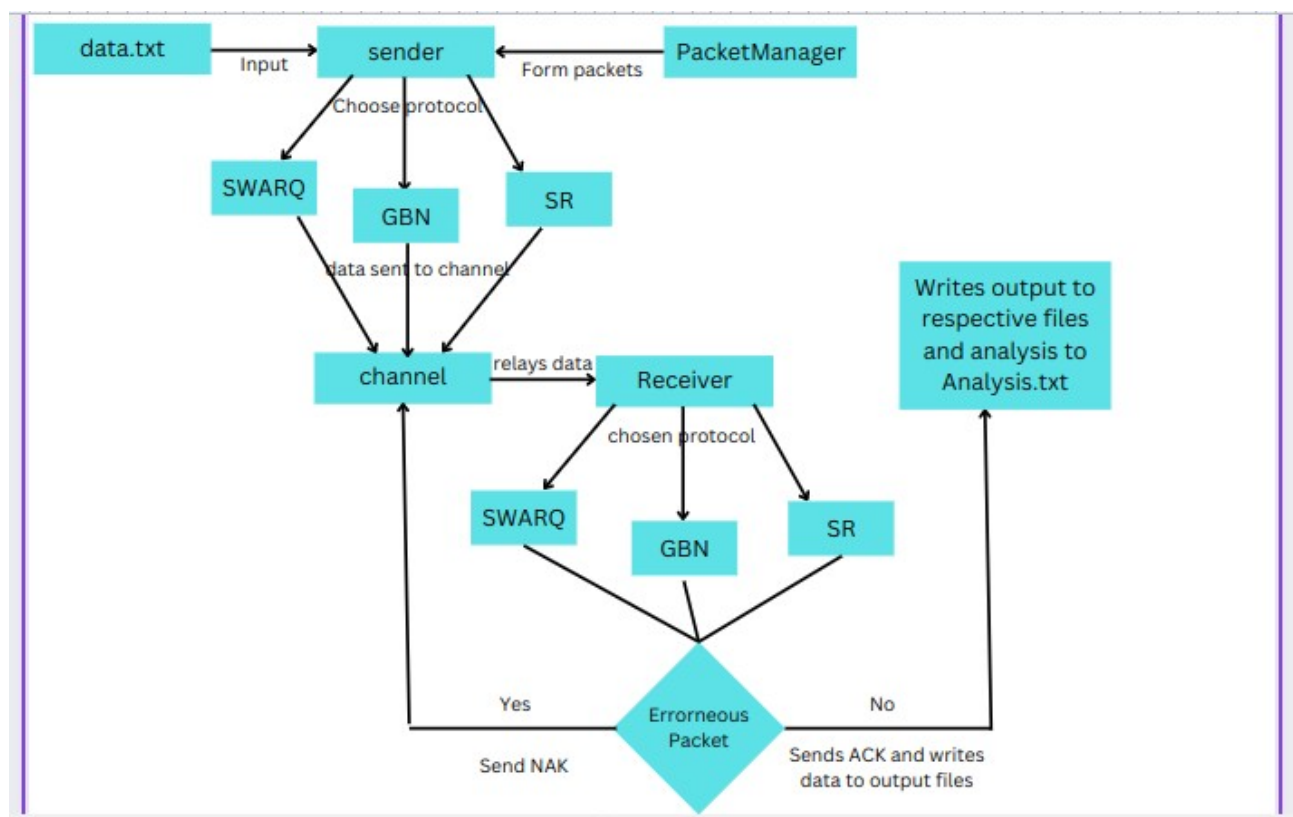
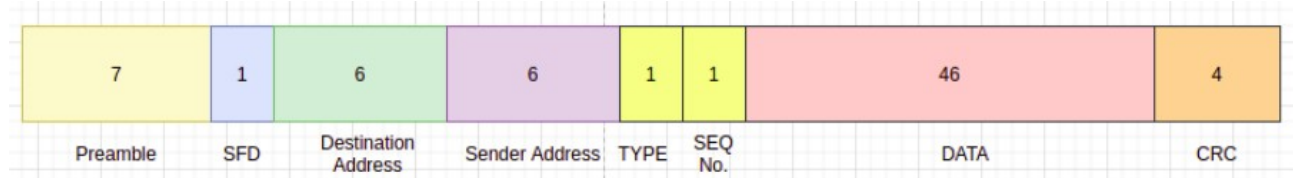


Fig: Flow Structure of the whole implementation

## IMPLEMENTATION

### 1. Stop and Wait ARQ (For noisy channel)

**Sender side Stop and Wait:** Sender sends one data packet at a time. Sender sends the next packet only when it receives the acknowledgment of the previous packet. Therefore, the idea of stop and wait protocol in the sender's side is very simple, i.e., send one packet at a time, and do not send another packet before receiving the acknowledgment.

**Receiver Side Stop and Wait:** Receive and then consume the data packet. When the data packet is consumed, receiver sends the acknowledgment to the sender. Therefore, the idea of stop and wait protocol in the receiver's side is also very simple, i.e., consume the packet, and once the packet is

consumed, the acknowledgment is sent. This is known as a flow control mechanism.

## 2. Go Back N ARQ

**Sender side GBN:** It is a sliding window protocol responsible for sending packets in windows of some fixed size and waiting for ACK for the entire window. If some frame is lost in the window before reaching the receiver, it will send the entire frame again.

**Receiver side GBN:** The receiver side is responsible for sending an ACK when it receives the entire window of frames and sending a NAK when a packet is lost due to time out or an erroneous packet is sent.

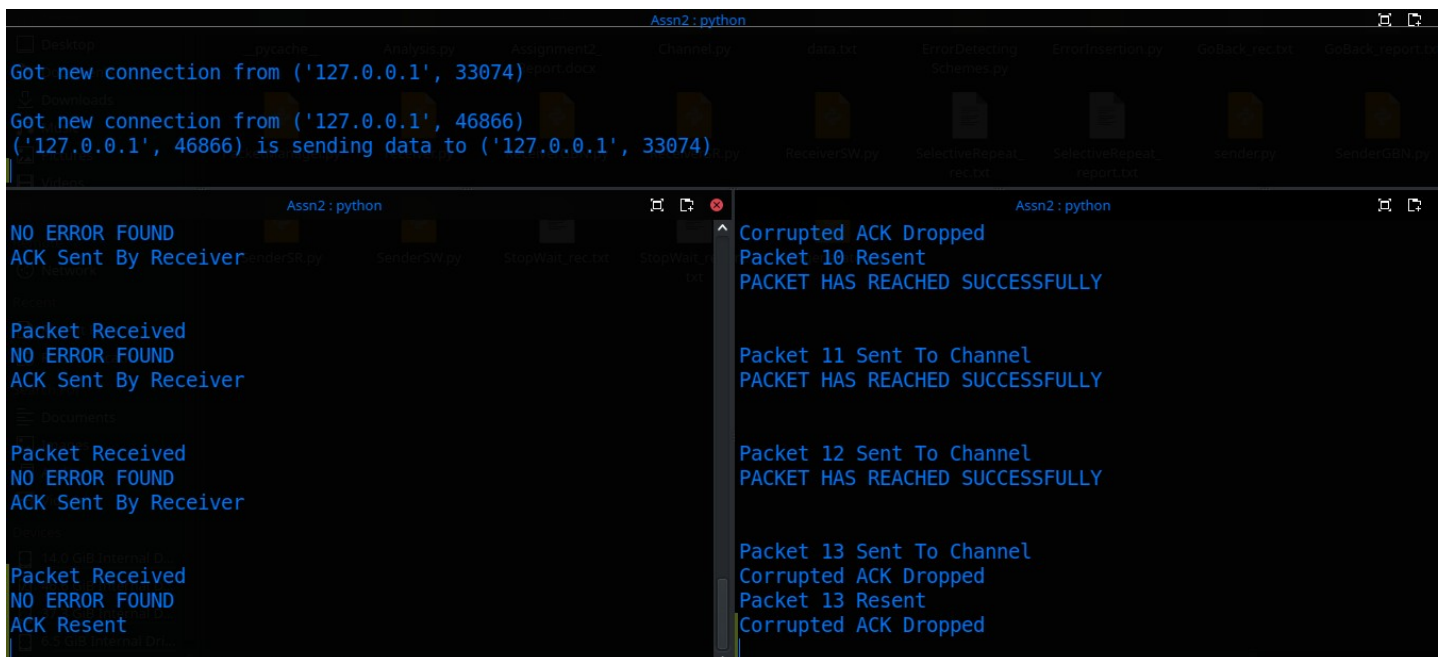
## 3. Selective Repeat ARQ

(Window size should be less than or equal to the half of the sequence number in SR protocol. This is to avoid packets being recognized incorrectly. If the windows size is greater than half of the sequence number space, then the ACK is lost, the sender may send new packets that the receiver believes are retransmissions. )

**Sender Side SR:** Sender can transmit new packets as long as their number is with W of all unACKed packets. It retransmits un-ACKed packets after a timeout – Or upon a NAK if NAK is employed.

**Receiver Side SR:** Receiver ACKs all correct packets. Receiver stores correct packets until they can be delivered in order to the higher layer.

## RESULTS:



```
Assn2 : python
Got new connection from ('127.0.0.1', 33074)
Got new connection from ('127.0.0.1', 46866)
('127.0.0.1', 46866) is sending data to ('127.0.0.1', 33074)

Assn2 : python
NO ERROR FOUND
ACK Sent By Receiver

Packet Received
NO ERROR FOUND
ACK Sent By Receiver

Packet Received
NO ERROR FOUND
ACK Sent By Receiver

Packet Received
NO ERROR FOUND
ACK Resent

Assn2 : python
Corrupted ACK Dropped
Packet 10 Resent
PACKET HAS REACHED SUCCESSFULLY

Packet 11 Sent To Channel
PACKET HAS REACHED SUCCESSFULLY

Packet 12 Sent To Channel
PACKET HAS REACHED SUCCESSFULLY

Packet 13 Sent To Channel
Corrupted ACK Dropped
Packet 13 Resent
Corrupted ACK Dropped
```

Fig: The working of Go-BackN shown above

```

Assn2 : python
Got new connection from ('127.0.0.1', 47442)
Got new connection from ('127.0.0.1', 48384)
('127.0.0.1', 48384) is sending data to ('127.0.0.1', 47442)

PACKET RECEIVED SUCCESSFULLY
Sent ACK no = 10

Packet Reached
NO ERRORS FOUND
Sent NAK no = 10

Packet Reached
NO ERRORS FOUND

Packet Reached
NO ERRORS FOUND

Packet Reached
NO ERRORS FOUND

Packet Reached
NO ERRORS FOUND

PACKET 10 SENT TO CHANNEL
PACKET 11 SENT TO CHANNEL
PACKET 12 SENT TO CHANNEL
PACKET 13 SENT TO CHANNEL
PACKET 14 SENT TO CHANNEL
PACKET 15 SENT TO CHANNEL
Corrupt NAK Dropped
Packet 8 HAS REACHED SUCCESSFULLY

Corrupt ACK Dropped
PACKET 0 SENT TO CHANNEL

```

Fig: The working of Selective Repeat shown above

Receiver Throughput

| protocols / Data size | Stop and Wait | Go Back N | Selective Repeat |
|-----------------------|---------------|-----------|------------------|
| 32768 bytes           | 317 bps       | 878 bps   | 1498 bps         |
| 65536 bytes           | 412 bps       | 1003 bps  | 1598 bps         |
| average               | 364.5         | 940.5     | 1548 bps         |

Efficiency

| protocols / Data size | Stop and Wait | Go Back N | Selective Repeat |
|-----------------------|---------------|-----------|------------------|
| 32768 bytes           | 7.94 %        | 11.25 %   | 17.09%           |
| 65536 bytes           | 8.64 %        | 12.34 %   | 18.92%           |
| average               | 8.29 %        | 11.795 %  | 18.005 %         |

Average Successful Transmission Time of a packet

| protocols / Data size | Stop and Wait (seconds) | Go Back N (seconds) | Selective Repeat (seconds) |
|-----------------------|-------------------------|---------------------|----------------------------|
| 32768 bytes           | 0.54667                 | 0.09505             | 0.199779                   |
| 65536 bytes           | 1.27865                 | 1.45634             | 1.784322                   |
| average               | 0.91266                 | 0.82322             | 0.9920505                  |

## **ANALYSIS:**

From the tables above we clearly rule out the differences between the performance measured in terms of throughput efficiency and average transmission time of the different flow control protocols.

Selective Repeat is better than Go Back N and stop and wait. The sliding window protocols are more efficient than the primitive protocol Stop and Wait.

Round Trip Time, which is the total time taken by a packet to move to and for should not be a measuring parameter because it's a property of the network and not the protocol. Although It can be

checked by evaluating the RTT for a noiseless channel and then comparing with the other protocols. Also while measuring the RTT the processing and transmission time has been ignored which can play a significant role. Similarly the Bandwidth Delay Product because it is too a property of the channel and not the protocol.

Instead of multiprocessing ,multithreading has been incorporated due to the difficulties faced while

establishing a communication between different ports. All the threads running are made to communicate through a single port 1232 here.

## **COMMENTS:**

Since these are low level protocols, we can't say that the results that we got are quite fair and accurate on the basis of mere simulations we made. This assignment was no way a practical one and was too much hectic too simulate. On the other hand, if we think closely, we're implementing these protocols with TCP sockets which lie on the layers above Network layer, so under the hood the same protocols are working in the lower levels hence we can't visualize the actual scenario.