

# ANALYSIS OF GO-BACK-N AND SELECTIVE REPEAT PROTOCOLS

*Report submitted to the SASTRA Deemed to be University  
as the requirement for the course*

## CSE302: COMPUTER NETWORKS

*Submitted by*

**Annam Venkata Sai**  
**(123003283,B-Tech CSE)**

**February 2022**



**SCHOOL OF COMPUTING**  
**THANJAVUR, TAMIL NADU, INDIA – 613 401**



**SCHOOL OF COMPUTING**  
**THANJAVUR – 613 401**

**Bonafide Certificate**

This is to certify that the report titled “**Analysis Of Go-Back-N And Selective Repeat Protocols**” submitted as a requirement for the course, **CSE302: COMPUTER NETWORKS** for B.Tech. is a bonafide record of the work done by **Shri. Annam Venkata Sai (Reg.No.123003283, B-Tech CSE)** during the academic year 2021-22, in the School of Computing, under my supervision

**Signature of Project Supervisor** :

**Name with Affiliation** :

**Date** :

Project Based Work *Viva voce* held on 11 Feb 2022

**Examiner 1**

**Examiner 2**

## ACKNOWLEDGEMENTS

First of all, I would like to thank God Almighty for his endless blessings.

I would like to express my sincere gratitude to **Dr S. Vaidyasubramaniam, Vice-Chancellor** for his encouragement during the span of my academic life at SASTRA Deemed University.

I would forever remain grateful and I would like to thank **Dr A.Umamakeswri, Dean, School of Computing** and **R. Chandramouli, Registrar** for their overwhelming support provided during my course span in SASTRA Deemed University.

I am extremely grateful to **Dr. Shankar Sriram, Associate Dean, School of Computing** for his constant support, motivation and academic help extended for the past three years of my life in School of Computing.

I would specially thank and express my gratitude to **N.Sasikala Devi , Associate Professor, School of Computing** for providing me an opportunity to do this project and for her guidance and support to successfully complete the project.

I also thank all the Teaching and Non-teaching faculty, and all other people who have directly or indirectly help me through their support, encouragement and all other assistance extended for completion of my project and for successful completion of all courses during my academic life at SASTRA Deemed University.

Finally, I thank my parents and all others who help me acquire this interest in project and aided me in completing it within the deadline without much struggle.

## List Of Figures

Figure No	Figure Name	Page No
Figure 1.1	Buffer in Sliding Window	10
Figure 1.2	Sender's Window	11
Figure 1.3	Receiver Sliding Window	12
Figure 2.1	TCP Sliding Window	14
Figure 2.2	Error Control Techniques	15
Figure 3.1	Stop and Wait operation	16
Figure 3.2	Retransmission due to lost Frame	17
Figure 3.3	Retransmission due to damaged Frame	17
Figure 3.4	Sender's view of sequence numbers in Go-Back-N	18
Figure 3.5	GBN Sender	19
Figure 3.6	GBN Receiver	20
Figure 3.7	Selective-repeat (SR) sender and receiver views of sequence-number space	21
Figure 3.8	SR Operation	25
Figure 3.9	Summary of reliable data transfer mechanisms and their use	26

### **List Of Tables:**

<b>Table No.</b>	<b>Table Name</b>	<b>Page No</b>
Table 1	Effect of Window Size(GBN)	50
Table 2	Effect of Window Size(SRR)	51
Table 3	Effect Of MSS(GBN)	52
Table 4	Effect Of MSS(SRR)	53
Table 5	Effect Of Loss Probability(GBN)	54
Table 6	Effect Of Loss Probability(SRR)	55

### **List of Graphs:**

<b>Table No.</b>	<b>Table Name</b>	<b>Page No</b>
Graph 1	Effect of Window Size(GBN)	50
Graph 2	Effect of Window Size(SRR)	51
Graph 3	Effect Of MSS(GBN)	52
Graph 4	Effect Of MSS(SRR)	53
Graph 5	Effect Of Loss Probability(GBN)	54
Graph 6	Effect Of Loss Probability(SRR)	55

## ABBREVIATIONS

<b>GBN</b>	GO-BACK-N(Transport Layer Protocol)
<b>SR</b>	Selective Repeat(Transport Layer Protocol)
<b>PKT</b>	Used to Mention a transport layer segment/packet
<b>RTT</b>	Round Trip Time(between sender and receiver)
<b>ACK</b>	Positive acknowledgements (a control packet, sent by receiver, upon receiving a valid packet )
<b>NACK</b>	Negative acknowledgment (a control packet, sent by receiver, upon receiving a corrupt/invalid packet)
<b>RDT</b>	reliable data transfer

## ABSTRACT

A **Sliding window Protocol** is a feature of packet base data transmission protocols. In sliding window protocol data is transmitted in order. This technique is used for reliable data transfer over **Data Link Layer** (OSI model) as well as in **Transmission Control Protocol** (TCP).

This techniques works on keeping limits on the number of packets that can be transmitted or received at any given time , allows unlimited number packets to be communicated using fixed size sequence numbers. In these both sender and receiver uses buffer called “**sending window**”.

“window” at sender side represents maximum number of packets that can be transmitted at a time , whereas packets to be acknowledged at receiver end. If sequence number of frames is  $n$  bit , then range of sequence number can be assigned 0 to  $2^n-1$ .Size of window size will be  $2^n-1$ .For example, if window size is 4, then sequence numbers will be 0,1,2,3,0,1,2,3... and so on . Number bits for sequence number is 2 and binary sequence is 00,01,10,11. It is called Sliding Window protocol because , window of the sequence slides only when it receives the least expected sequence packet in the window.

Sliding Window Protocol has two types:

- 1)Go-Back-N ARQ and
- 2)Selective Repeat ARQ

**Go-Back-N ARQ :** In this technique we send multiple frames until we receive acknowledgement for the first frame. All the frames are transmitted sequentially numbered and without waiting for ACK . The receiver checks for sequence number for next frame it expects to receive . If expected is not received then it will discard the frame and sends ACK for last correct order frame . Once the sender has sent all of the frames in its window, it will detect that all of the frames since the first lost frame are outstanding, and will go back to the sequence number of the last ACK it received from the receiver process and fill its window starting with that frame and continue the process over again . Go-Back-N is more efficient of Stop and wait ARQ, Since waiting time for each ACK is utilized efficiently.

**Selective Repeat ARQ:** This protocol also uses sliding window for reliable data transfer of Data over data link layer . In this technique we use buffer for storing frames out-of-order and only the lost frames are retransmitted. The receiver records the sequence number of the earliest incorrect or un-received frame. It then fills the receiving window with the subsequent frames that it has received. It sends the sequence number of the missing frame along with every acknowledgement frame. The sender continues to send frames that are in its sending window. Once, it has sent all the frames in the window, it retransmits the frame whose sequence number is given by the acknowledgements. It then continues sending the other frames.

**Keywords:** reliable data transfer, Data Link Layer, Transmission Control Protocol(TCP), Sending window , acknowledgement (ACK), Go-Back-N (GBN), Stop and wait, Selective Repeat, Buffer, Subsequent frames, Sequence Number.



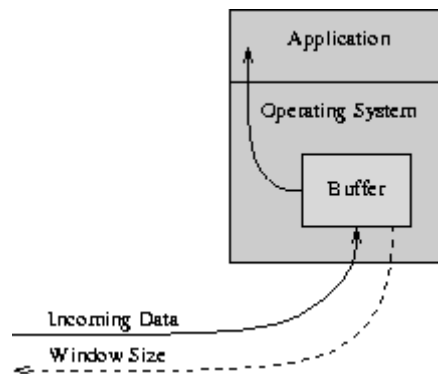
## TABLE OF CONTENTS

S.No	TOPIC	Page No.
•	Bonafide Certificate	ii
•	Acknowledgements	iii
•	List of Figures	iv
•	List of Tables	v
•	List of Graphs	v
•	Abbreviations	vi
•	Abstract	vii
1.	<a href="#">Chapter 1 Introduction</a>	2
2.	<a href="#">Chapter 2 TCP And ERROR Control Technique</a>	5
3.	<a href="#">Chapter 3 Protocols Explanation</a>	8
4.	<a href="#">Chapter 4 Algorithm And Source Code</a>	19
5.	<a href="#">Chapter 5 Performance Evaluation</a>	46
6.	<a href="#">Chapter 6 Conclusion &amp; Future Plans</a>	53
7.	<a href="#">Chapter 7 References</a>	54

# Chapter 1

## Introduction

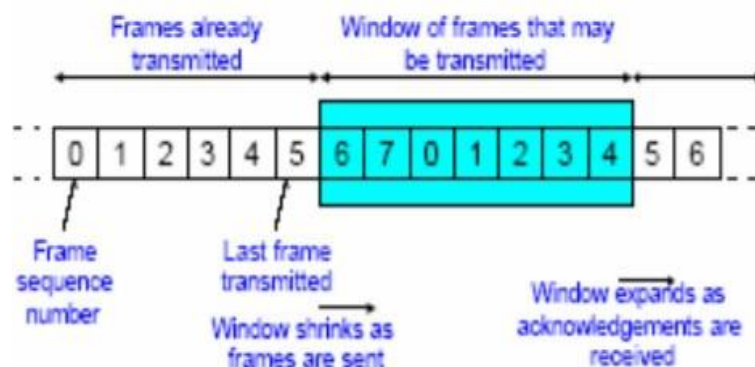
In Stop and wait protocol we can transmit only one frame at a time which is very less efficient. Efficiency can be greatly improved by allowing multiple frames to be in transit at the same time. Efficiency can also be improved by making use of the full-duplex line. Sender sends sequentially in order to keep track of frames. If the frame header allows  $k$  bits, then sequence number will be in the range of  $2^k - 1$ . Sender needs to maintain the sequence numbers that is allowed to send. Size of sender's window should be at most  $2^k - 1$ . Sender needs to maintain buffer equal to window size. Similarly Receiver also should maintain a window size  $2^k - 1$ . Receiver sends an ACK frame which also includes sequence number next expected frame. This also explicitly announces that it is prepared to receive the next  $N$  frames, beginning with the number specified. This scheme can be used to acknowledge multiple frames. Sliding window algorithm is a method of flow control for network data transfers. TCP, the Internet's stream transfer protocol, uses a sliding window algorithm. Buffer is placed between the application program and network data flow. The buffer is typically in OS kernel.



**Figure 1.1 Buffer in sliding window**

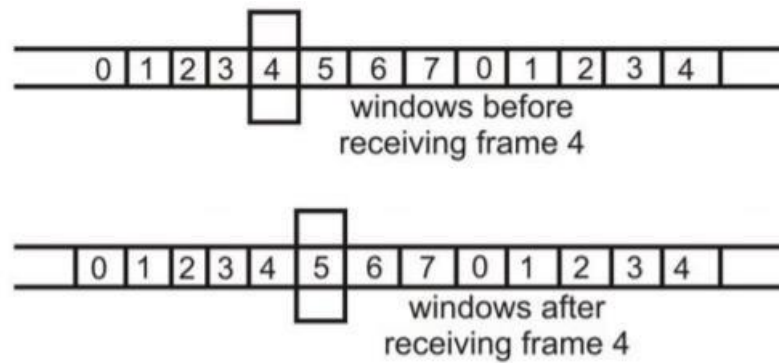
Data received from the network is stored in the buffer, from where the application can read at its own pace. When application reads the data then the buffer is freed up and accepts input from the network. Window announcements are used to inform the remote host of the current window size.

**Sender Sliding Window :** The sender sends frames at any instant with sequence numbers in the range of sending window as shown in Fig.



**Figure 1.2 Sender's window**

**Receiver Sliding Window :** The receiver always maintains a window of size 1 as shown in Fig. 3.3.4. It looks for a specific frame (frame 4 as shown in the figure) to arrive in a specific order. If it receives any other frame (out of order), it is discarded and it needs to be present. However, the receiver window also slides by one as the specific frame is received and accepted as shown in the figure. The receiver acknowledges a frame by sending an ACK frame that includes the sequence number of the next frame expected. This also explicitly announces that it is prepared to receive the next N frames, beginning with the number specified. This scheme can be used to acknowledge multiple frames. It could receive frames 2, 3, 4 but withhold ACK until frame 4 has arrived. By returning an ACK with sequence number 5, it acknowledges frames 2, 3, 4 at one time. The receiver needs a buffer of size 1



**Figure 1.3 Receiver sliding window**

On the other hand, if the local application can process data at the rate it's being transferred; sliding window still gives us an advantage. If the window size is larger than the packet size, then multiple packets can be outstanding in the network, since the sender knows that buffer space is available on the receiver to hold all of them. Ideally, a steadystate condition can be reached where a series of packets (in the forward direction) and window announcements (in the reverse direction) are constantly in transit. As each new window announcement is received by the sender, more data packets are transmitted. As the application reads data from the buffer (remember, we're assuming the application can keep up with the network), more window announcements are generated. Keeping a series of data packets in transit ensures the efficient use of network resources.

Hence, Sliding Window Flow Control

- Allows transmission of multiple frames
- Assigns each frame a k-bit sequence number
- Range of sequence number is  $[0 \dots 2k - 1]$ , i.e., frames are counted modulo  $2k$ .

The link utilization in case of Sliding Window Protocol

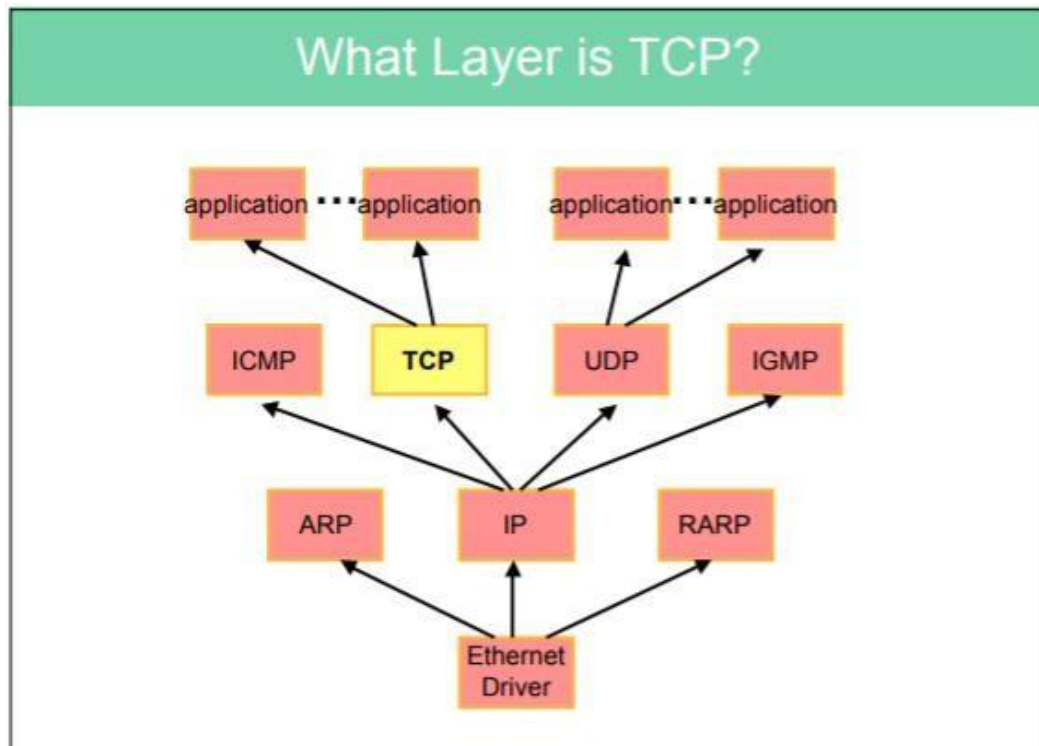
$$U = 1, \quad \text{for } N > 2a + 1$$

$$N/(1+2a), \quad \text{for } N < 2a + 1 \text{ Where } N = \text{the window size,}$$

and  $a = \text{Propagation time} / \text{transmission time}$

## CHAPTER 2

### TCP Connection:



**TCP** processes sends and receives data as unstructured stream of bytes. It divides data stream into segments independently of application program writes. TCP establishes an end-to-end connection maintained by the endpoints not by the network.

The connection is full-duplex i.e., bi-directional:

- Concurrent transfer in both directions is possible.
- Connection establishment sets up both directions.
- May terminate each direction individually.

#### TCP Provides:

1. Addressing
2. Reliable delivery
3. In-order delivery
4. Flow control
5. Congestion control
6. Segmentation

## TCP Sliding Windows:

Window size limits how much data can be sent without an acknowledgment. It Allows efficient transmission.

Window Size:

- number of bytes the receiver is willing to accept, starting with byte specified in ACK

Number

- 16 bits long; max value = 65535 (bytes)

- **Example** – for ACK Number = 5400, Window Size = 500, sender is allowed to transmit 5400...5899 to the receiver

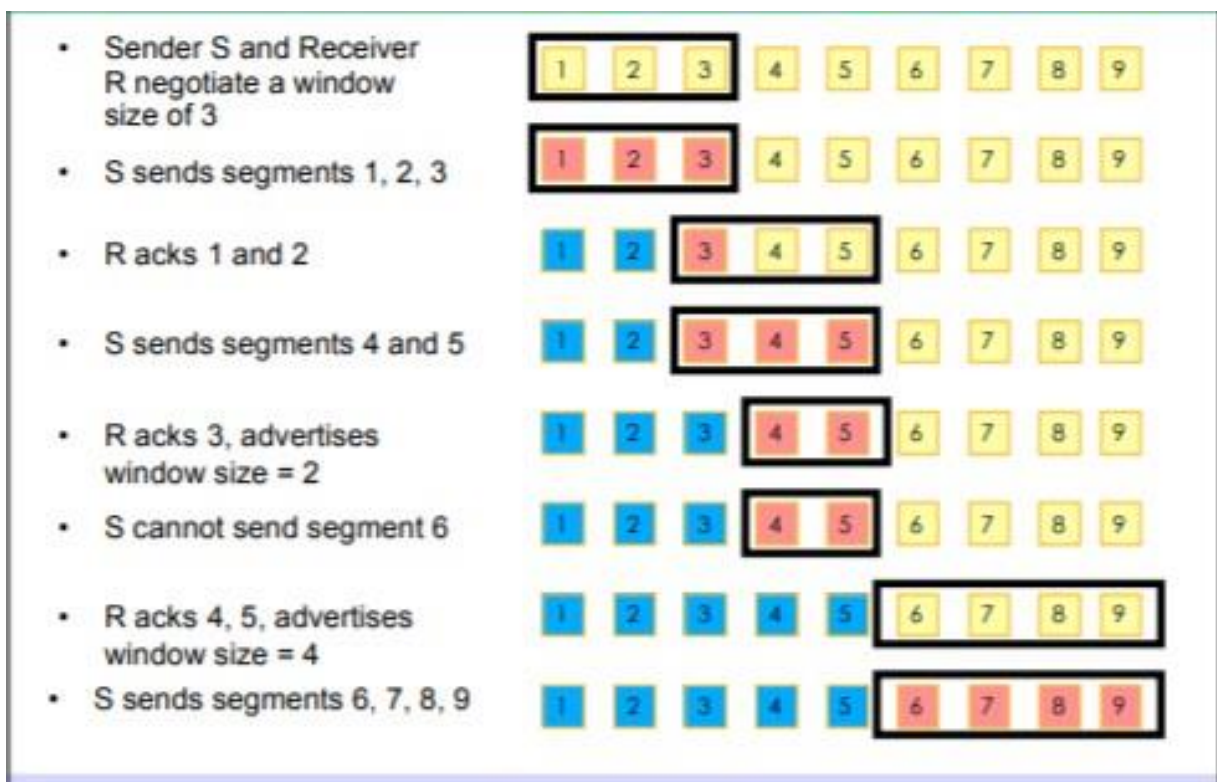


Figure 2.1 TCP Sliding Window

## Error Control techniques :

Receiver sends request to the Transmitter for retransmission of packets, when an error is detected in the message. Automatic repeat request is the most popular retransmission scheme. where receiver asks transmitter to re-transmit if it detects an error, are known as reverse error correction techniques.

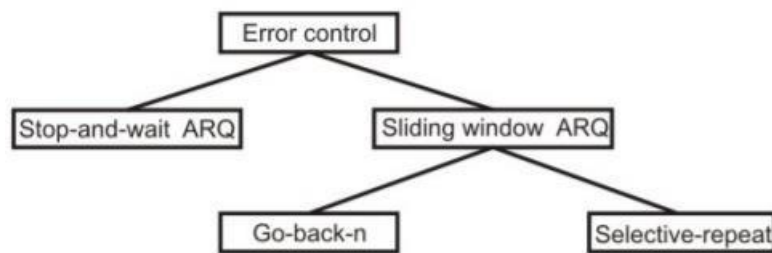
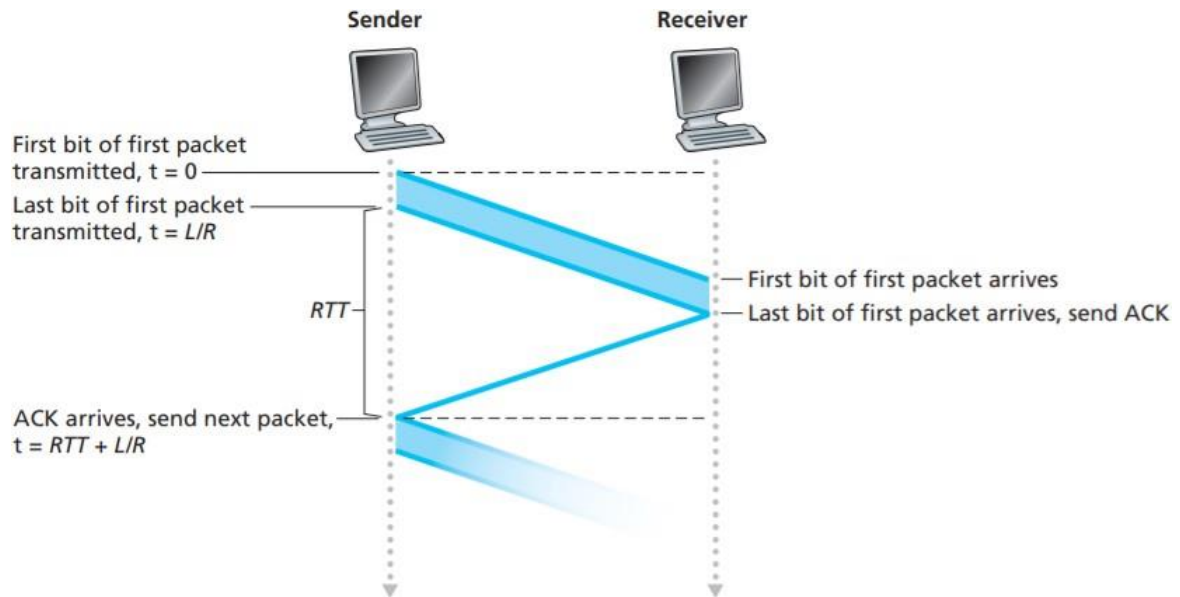


Figure 2.2 Error control techniques

## CHAPTER 3

### Stop and Wait ARQ :

This technique is simplest among all, in which sender transmits the packet and waits till it receives the positive ACK or NACK from the receiver. Receiver sends ACK if it receives the packet correctly, otherwise it sends NACK. Then the Sender re-transmits the packet again.



**Figure 3.1 Stop-and-wait-operation**

In case of lost ACK, sender retransmits the old frame. The sender is not aware of loss frame, but starts a timer when it sends the packet to the receiver. The ACK is received before timer expires. If the ACK is received the timer is set to ZERO and the next packet is transmitted.



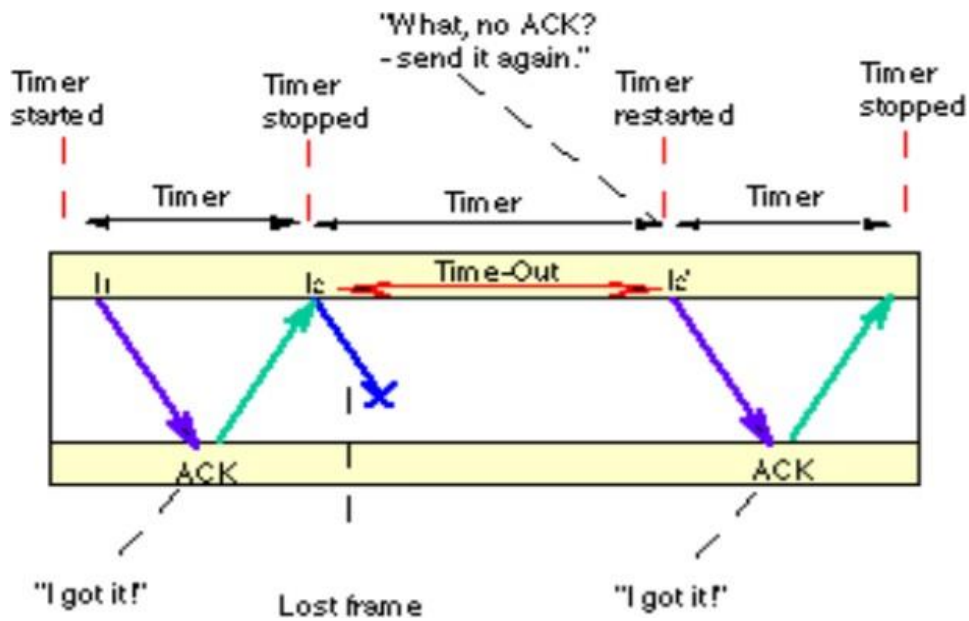


Figure 3.2 Retransmission due to lost frame

The receiver can identify the duplicate packet from label of the packet and it is discarded.

In order to tackle problem of damaged frames i.e., if the frame is corrupted during the transmission due to noise NACK is transmitted by the receiver to the sender. When a NACK is received by a transmitter before the time-out, the old frame is sent again.

Simplicity and minimum buffer size are the main advantage of Stop-and-Wait protocol.

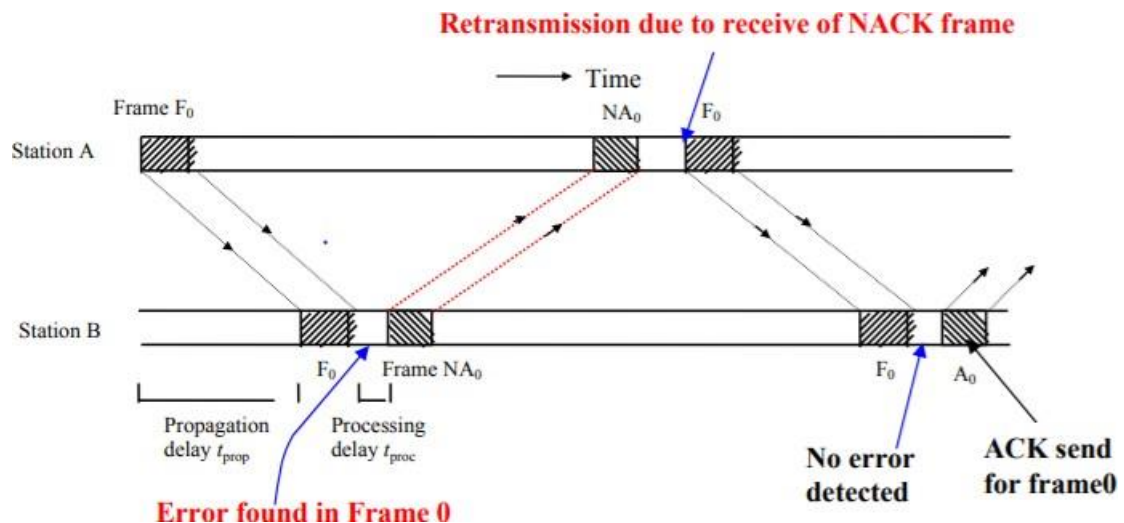


Figure 3.3 Retransmission due to damaged frame

## Go-Back-N (GBN) :

In GBN sender can multiple packets without waiting for acknowledgement but not more than maximum allowable packets in the pipeline.

Figure 3.19 shows the sender's view of the range of sequence numbers in a GBN protocol. Let base be sequence number of the oldest unacknowledged packet and the nextseqnum to be the sequence number of the next packet to be sent.

- Sequence number are in the range  $[0, \text{base}-1]$ . w.r.t to packets that have already been transmitted and acknowledged.
- Val  $[\text{base}, \text{nextseqnum}-1]$  corresponds to packets that have transmitted but not yet acknowledged.
- Sequence number in interval  $[\text{nextseqnum}, \text{base}+N-1]$  used for packets that can be sent immediately. But data should be available form upper layer.
- Sequence numbers greater than or equal to  $\text{base}+N$  cannot be used until an unacknowledged packet currently in the pipeline.

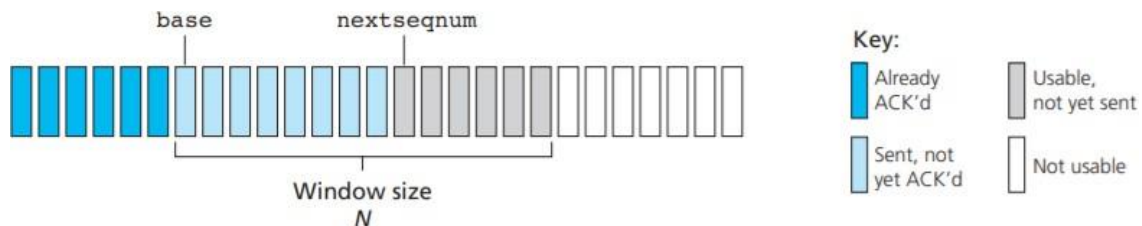


Figure 3.4 Sender's view of sequence numbers in Go-Back-N

From fig. interval of permissible sequence numbers for transmitted but not yet acknowledged packets can be viewed as window size of  $N$  in the range of sequence numbers. The window slides forward over sequence number range as the protocol operates.  $N$  refers to the **window size** and the **GBN** itself as a **sliding-window protocol**. You will be wondering why we are limiting outstanding, unacknowledged packets to a number of  $N$ . **Why not unlimited ?** Flow control is one reason to impose limit on the sender.

The packets sequence number is fixed length field in packet header. If  $k$  is the bits of packet sequence number then  $[0, 2^k - 1]$  be the range of sequence number. To obtain finite range of sequence number we must perform  $2^k$  arithmetic modulo operation (That is, the sequence number space can be thought of as a ring of size  $2^k$ , where sequence number  $2^k - 1$  is immediately followed by sequence number 0.)

Following figures represent FSM of sender and receiver sides of an ACK-based, NACK-free, GBN protocol.

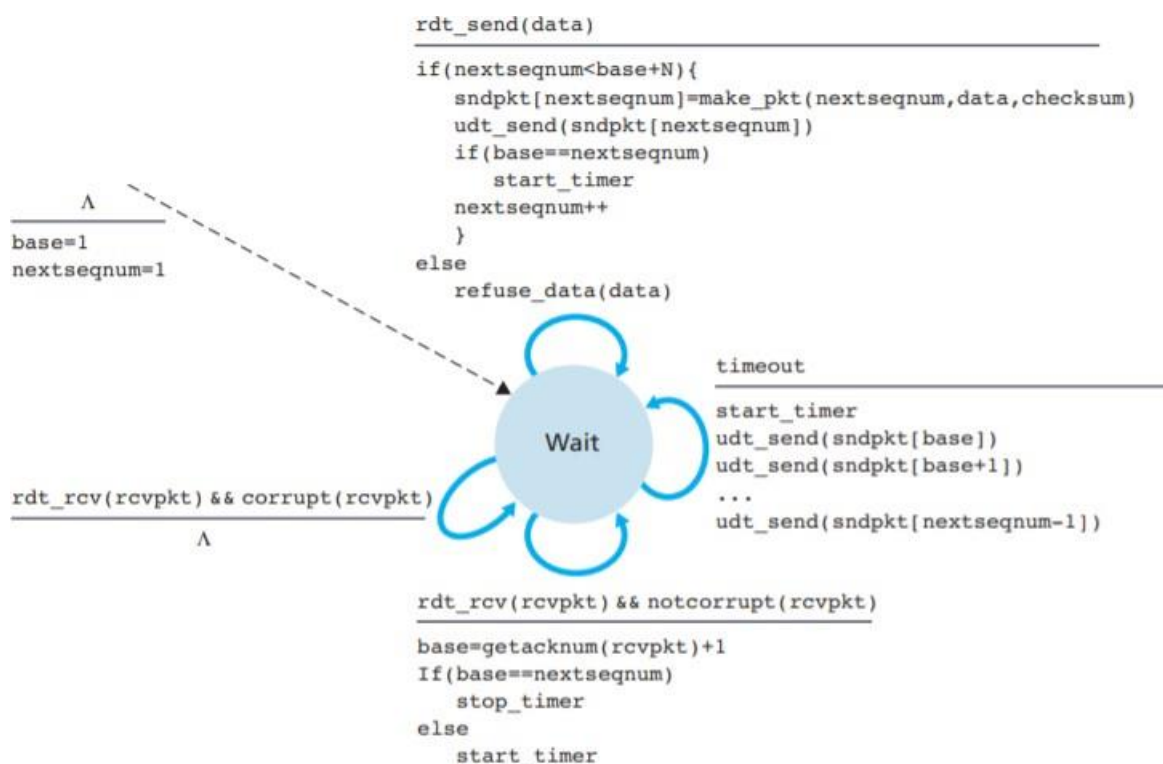


Figure 3.5 GBN Sender

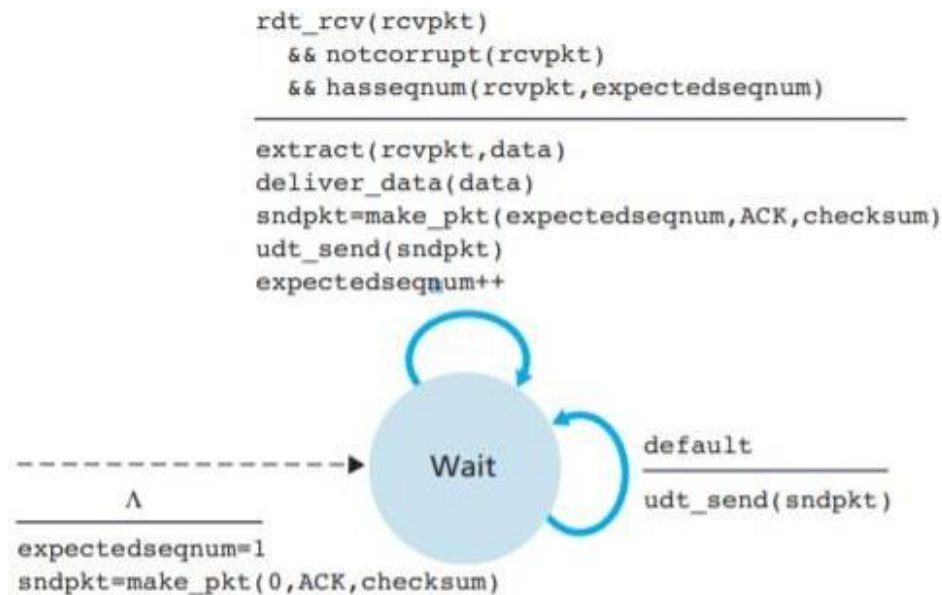


Figure 3.6 GBN Receiver

### GBN sender must respond to three types of events:

- Invocation from above. When `rdt_send()` is called from above, the sender first checks to see if the window is full, that is, whether there are  $N$  outstanding, unacknowledged packets. If the window is not full, a packet is created and sent, and variables are appropriately updated. If the window is full, the sender simply returns the data back to the upper layer, an implicit indication that the window is full. The upper layer would presumably then have to try again later. In a real implementation, the sender would more likely have either buffered (but not immediately sent) this data, or would have a synchronization mechanism (for example, a semaphore or a flag) that would allow the upper layer to call `rdt_send()` only when the window is not full.
- Receipt of an ACK. In our GBN protocol, an acknowledgment for a packet with sequence number  $n$  will be taken to be a cumulative acknowledgment, indicating that all packets with a sequence number up to and including  $n$  have been correctly received at the receiver. We'll

come back to this issue shortly when we examine the receiver side of GBN.

- A timeout event. The protocol's name, "Go-Back-N," is derived from the sender's behaviour in the presence of lost or overly delayed packets. As in the stop-and-wait protocol, a timer will again be used to recover from lost data or acknowledgment packets. If a timeout occurs, the sender resends all packets that have been previously sent but that have not yet been acknowledged. Our sender uses only a single timer, which can be thought of as a timer for the oldest transmitted but not yet acknowledged packet. If an ACK is received but there are still additional transmitted but not yet acknowledged packets, the timer is restarted. If there are no outstanding, unacknowledged packets, the timer is stopped.

Receiver action in GBN is simple i.e., if a packet with sequence number is correctly and is in Order, the receiver sends an ACK for packet  $n$  and delivers the data portion of the packet to the upper layer. In other cases receiver discards packet and resends the ACK for most recently received in-order packet. Note that since packets are delivered one at a time to the upper layer, if packet  $k$  has been received and delivered, then all packets with a sequence number lower than  $k$  have also been delivered. Thus, the use of cumulative acknowledgments is a natural choice for GBN.

In GBN receiver discard out of order packets. Although it seems silly and waste of discarding correctly received packet, there is a justification for doing so. Receiver must deliver data in order to the upper layer. If packet  $n$  is expected, but packet  $n+1$  arrives.

## Selective Repeat

As the name suggests, **Selective Repeat** avoids unnecessary packets for retransmissions by allowing sender to retransmit only those were lost or corrupted at the receiver. For this retransmission receiver need to individually acknowledge each individually correctly received packets. Here also we require window size  $n$  to limit the transmission packets i.e, outstanding and unacknowledged at the sender and receiver side.

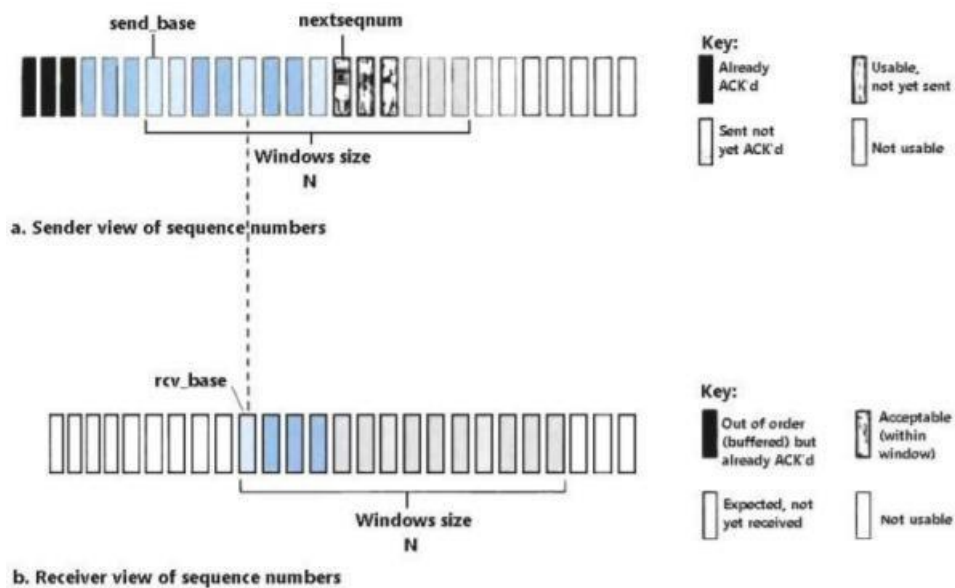


Figure 3.7 Selective-repeat (SR) sender and receiver views of sequence-number space

Unlike GBN, in SR sender have received ACK for some transmitted packets in the window.

Fig 3.7 shows the sender's and receivers view of sequence number space.

SR receiver will acknowledge correctly received packets tough they are out of order. These out of order packets are buffered at the receiver end until any missing packets are received, with this receiver will transmits in-order packets to its upper layer.

**Sender Window:**

Sender window is much smaller in selective repeat protocol compared to go-back-n protocol. Size of Sender window for k bits size is  $2^{m-1}$ . Here m is the number of bits used by the packet header to express the sequence of the packet.

Sender Window covers the packets that are sent by the sender but not acknowledged, one that is acknowledged out of order and the one that can be sent once the data for the corresponding are received by the sender's application layer.

**Receiver Window:**

Size of the receiver window also  $2^{m-1}$  which is also similar to the sender window. It covers the sequence of packets that are received out-of-order and are waiting for the packets that were sent earlier but not yet received.

Receiver transport layer delivers only packets in order to the application layer. It waits for the set of consecutive packets, then they will be delivered to the application layer.

Here we attach timer at the sender side for every packet. If the acknowledgment is not received in time, Then the packet will be retransmitted.

**SR sender events and actions:**

- Data received from above. When data is received from above, the SR sender checks the next available sequence number for the packet. If the sequence number is within the sender's window, the data is packetized and sent; otherwise it is either buffered or returned to the upper layer for later transmission, as in GBN.
- Timeout. Timers are again used to protect against lost packets. However, each packet must now have its own logical timer, since only a single packet will be transmitted on timeout. A single hardware timer can be used to mimic the operation of multiple logical timers.
- ACK received. If an ACK is received, the SR sender marks that packet as having been received, provided it is in the window. If the packet's sequence number is equal to send\_base, the window base is moved forward to the unacknowledged packet with the Smallest sequence number. If the window moves and there are untransmitted packets with sequence numbers that now fall within the window, these packets are transmitted.

**SR Receiver events and actions:**

- Packet with sequence number in  $[rcv\_base, rcv\_base+N-1]$  is correctly received. In this case, the received packet falls within receiver's window and a selective ACK packet is returned to the sender. If the packet was not previously received, it is buffered. If this packet has a sequence number equal to the base of the receive window ( $rcv\_base$ ) then this packet, and any previously buffered and consecutively numbered (beginning with  $rcv\_base$ ) packets are delivered to the upper layer. The receiver window is then moved forward by the number of packets delivered to the upper layer. When a packet with a sequence number of  $rcv\_base+2$  is received, it packets 3, 4 and 5 can be delivered to the upper layer.
- Packet with sequence number in  $[rcv\_base-N, rcv\_base-1]$  is correctly received. In this case, an ACK must be generated, even though this is a packet that the receiver has previously acknowledged.
- Otherwise. Ignore packet.



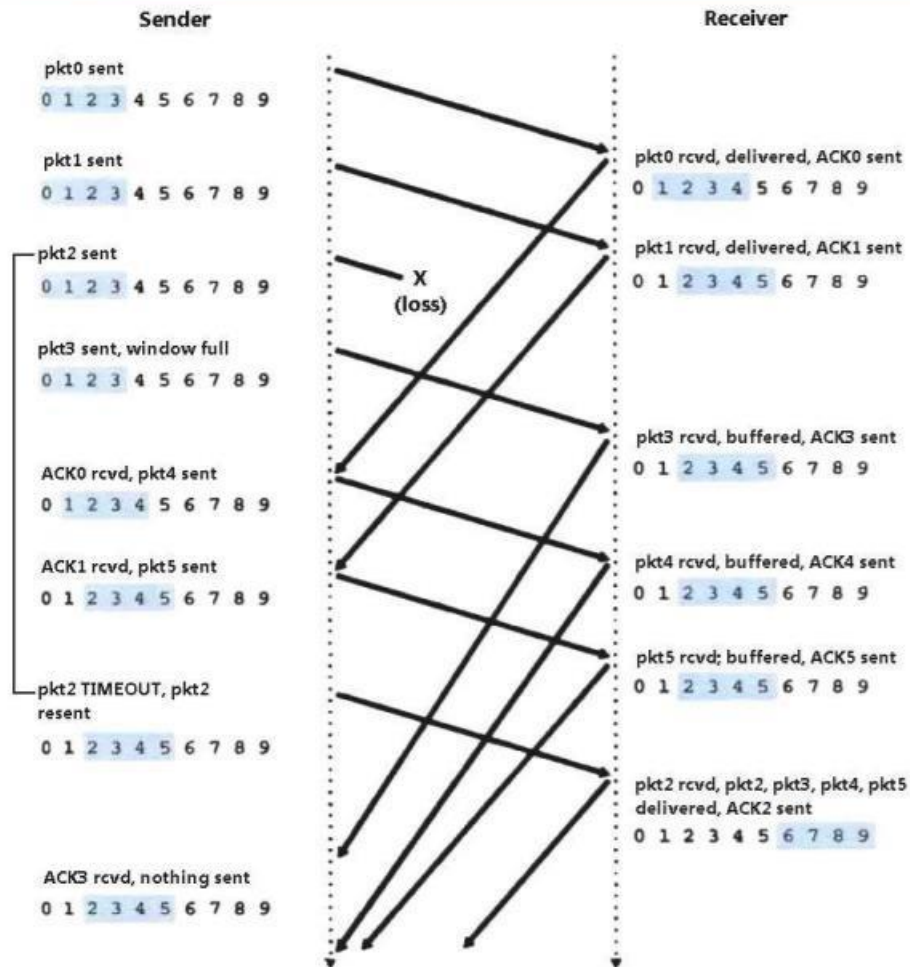


Figure 3.8 SR Operation

Synchronization is required between sender and receiver windows. consider four packet window size and sequence numbers 0,1,2,3 and window size of three. Packets 0 to 2 are transmitted and correctly received and acknowledged at the receiver end. . At this point, the receiver's window is over the fourth, fifth, and sixth packets, which have sequence numbers 3, 0, and 1 respectively.

Mechanism	Use, Comments
Checksum	Used to detect bit errors in a transmitted packet.
Timer	Used to timeout/retransmit a packet, possibly because the packet (or its ACK) was lost within the channel. Because timeouts can occur when a packet is delayed but not lost (premature timeout), or when a packet has been received by the receiver but the receiver-to-sender ACK has been lost, duplicate copies of a packet may be received by a receiver.
Sequence number	Used for sequential numbering of packets of data flowing from sender to receiver. Gaps in the sequence numbers of received packets allow the receiver to detect a lost packet. Packets with duplicate sequence numbers allow the receiver to detect duplicate copies of a packet.
Acknowledgment	Used by the receiver to tell the sender that a packet or set of packets has been received correctly. Acknowledgments will typically carry the sequence number of the packet or packets being acknowledged. Acknowledgments may be individual or cumulative, depending on the protocol.
Negative acknowledgment	Used by the receiver to tell the sender that a packet has not been received correctly. Negative acknowledgments will typically carry the sequence number of the packet that was not received correctly.
Window, pipelining	The sender may be restricted to sending only packets with sequence numbers that fall within a given range. By allowing multiple packets to be transmitted but not yet acknowledged, sender utilization can be increased over a stop-and-wait mode of operation. We'll see shortly that the window size may be set on the basis of the receiver's ability to receive and buffer messages, or the level of congestion in the network, or both.

**Figure 3.9 Summary of reliable data transfer mechanisms and their use**

## CHAPTER 4

### ALGORITHMS

#### Go-Back-N Sender:

Function Sender is

```
    Send_base  $\leftarrow$  0;
    nextseqnum  $\leftarrow$  0;
    while true do
        if nextseqnum < send_base + N then
            send packet nextseqnum;
            nextseqnum  $\leftarrow$  nextseqnum+1;
        end
        if receive ACK n then
            send_base  $\leftarrow$  n+1;
            if send_base == nextseqnum then
                stop timer;
            else
                start timer;
            end
        end
        if timeout then
            start timer;
            send packet send_base;
            send packet send_base+1;
            .....
            .....
            Send packet nextseqnum-1;
        end
    end
end
```

### **Go-Back-N Receiver:**

**Function Receiver is**

```
    nextseqnum  $\leftarrow$  0;
    while True do
        if A packet is received then
            if The received packet is not corrupted and sequence_number ==
nextseqnum then
                deliver the data to the upper layer;
                send ACK nexteqnum;
                nextseqnum  $\leftarrow$  nextseqnum + 1;
            else
                /* If the packet is corrupted or out of order,simply drop it */
                send ACK nextseqnum – 1;
            end
        else
            end
    end
end
```

## Source Code:

### GBN Sender:

```
from multiprocessing.connection import Connection
import time, socket, sys

def decimalToBinary(n):
    return n.replace("0b", "")

def binarycode(s):
    b_array = bytearray(s, "utf8")

    b_list = []

    for byte in b_array:
        b_represent = bin(byte)
        b_list.append(decimalToBinary(b_represent))

    print(b_list)
    a=""
    for i in b_list:
        a=a+i
    return a

print("\nWelcome.....\n")
time.sleep(1)

#creating a socket
sock = socket.socket()
host_name = socket.gethostname()
ip_addr = socket.gethostbyname(host_name)
port_no = 1234
#binding socket with port number (0 --> 65535)
```

```

sock.bind((host_name, port_no))
print(host_name, "[", ip_addr, "]\n")
client_name = input(str("Enter your name: "))
#Accepting one connection
sock.listen(1)
print("\nWaiting for incoming connections...\n")
#Accepting client request
connection, addr = sock.accept() #conn client socket, and address
print("Received connection from ", addr[0], "(", addr[1], ")\n")
#receiver with buffer 1024
s_name = connection.recv(1024)
s_name = s_name.decode()
print(s_name, " connected the chat room")
print("Enter 'bye' to exit\n")

#sending server name

connection.send(client_name.encode())

#accepting messages to send
while True:
    msg = input(str("Enter Data : "))

    connection.send(msg.encode())
    #accepts input until user enter '[e]'
    if msg == "bye":
        msg = "Left chat room!"
        connection.send(msg.encode())
        print("        \n")
        break

    #converting message to binary format
    msg=binarycode(msg)
    f=str(len(msg))
    connection.send(f.encode())

```

```

n=0
m=int(input("Enter the window size -> ")) #window size
b=""
m=m-1
f=int(f)
k=m
count=0
while n!=f:
    for cnt in range(n,k+1):
        print("sending...",cnt)
    while(n!=(f-m)):
        connection.send(msg[n].encode())
        if(count>m):
            print("sending...",count)
        #receiving ACK
        b=connection.recv(1024)
        b=b.decode()
        print(b)
        #validating ACK
        if(b!="ACK Lost"):
            time.sleep(1)
            print("Acknowledgement "+str(count)+" Received!")
            st1=str(n+1);
            st2=str(k+1);
            print(" Sliding window --> ["+st1+" .... "+st2+"] sending next packet
to receiver")
            #sliding the window
            n=n+1
            k=k+1
            count+=1
            time.sleep(1)
        #if NACK received
    else:
        time.sleep(1)
        print("Acknowledgement "+str(count)+" Received!")

```

```

        st1=str(n+1);
        st2=str(k+1);
        print(" Sliding window --> ["+st1+" .... "+st2+"] sending next packet
to receiver")

        time.sleep(1)

#resending Packets
while(n!=f):
    connection.send(msg[n].encode())
    print("sending....",count)
    b=connection.recv(1024)
    b=b.decode()
    print(b)
    if(b!="ACK Lost"):
        time.sleep(1)
        print("Acknowledgement "+str(count)+" Received!")
        st1=str(n+1);
        st2=str(k+1);
        print("Sliding window --> ["+st1+" .... "+st2+"] sending next packet
to receiver")

        count+=1
        n=n+1
        time.sleep(1)
    else:
        time.sleep(1)
        print("Acknowledgement "+str(count)+" Received!")
        st1=str(n+1);
        st2=str(k+1);
        print("Sliding window --> ["+st1+" .... "+st2+"] sending next packet
to receiver")

        time.sleep(1)

```



### GBN Receiver :

```
import time, socket, sys
import random

print("\nWelcome....\n")
print("Initialising....\n")
time.sleep(1)

#creating a client socket

s = socket.socket()
shost_name = socket.gethostname()
ip_addr = socket.gethostbyname(shost_name)
print(shost_name, "(", ip_addr, ")\n")
host_name = input(str("Enter server address: "))
name = input(str("\nEnter your name: "))
port_no = 1234
print("\nConnecting to ", host_name, "(", port_no, ")\n")
time.sleep(1)

#connecting to server using IP address and port number

s.connect((host_name, port_no))
print("Connected...\n")

s.send(name.encode())

#receiving server name

s_name = s.recv(1024)
s_name = s_name.decode()
print(s_name, " joined...\n")
print("Enter 'bye' to exit\n")
#Accepting Messages
while True:

    m=s.recv(1024)
    m=m.decode()
    k=s.recv(1024)
```

```

k=k.decode()
k=int(k)
i=0
a=""
b=""
count=0
#Generating random number
f=random.randint(0,1)
msg=""
while i!=k:

    f=random.randint(0,1)
    if(f==0): #sending NACK
        b="ACK Lost"
        print("NACK....:",count)
        msg = s.recv(1024)
        msg = msg.decode()
        s.send(b.encode())

    elif(f==1): #Sending ACK
        b="ACK "+str(i)
        msg = s.recv(1024)
        msg = msg.decode()
        print("Received bit:",count)
        s.send(b.encode())
        print("Acknowledged...:)",count)
        a=a+msg
        count+=1
        i=i+1

print("Received message is :", m)

```

## Sender Side Output:

### Window size : 5

Command Prompt

```
D:\Desktop\CN\CN Project>python sender.py

Welcome to Chat Room

Initialising....

MR-ANNAM-123 ( 192.168.164.1 )

Enter your name: annam

Waiting for incoming connections...

Received connection from 192.168.164.1 ( 52272 )

sai has connected to the chat room
Enter 'bye' to exit chat room

Enter Data : ab
['1100001', '1100010']
Enter the window size -> 5
sending.... 0
sending.... 1
sending.... 2
sending.... 3
sending.... 4
ACK 0
Acknowledgement 0 Received! The sliding window is in the range 1 to 5 Now sending the next packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 2 to 6 Now Resending the same packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 2 to 6 Now Resending the same packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 2 to 6 Now Resending the same packet
ACK 1
Acknowledgement 1 Received! The sliding window is in the range 2 to 6 Now sending the next packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 3 to 7 Now Resending the same packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 3 to 7 Now Resending the same packet
ACK 2
Acknowledgement 2 Received! The sliding window is in the range 3 to 7 Now sending the next packet
ACK 3
Acknowledgement 3 Received! The sliding window is in the range 4 to 8 Now sending the next packet
ACK 4
Acknowledgement 4 Received! The sliding window is in the range 5 to 9 Now sending the next packet
sending.... 5
ACK 5
Acknowledgement 5 Received! The sliding window is in the range 6 to 10 Now sending the next packet
sending.... 6
ACK 6
Acknowledgement 6 Received! The sliding window is in the range 7 to 11 Now sending the next packet
sending.... 7
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 8 to 12 Now Resending the same packet
sending.... 7
ACK 7
Acknowledgement 7 Received! The sliding window is in the range 8 to 12 Now sending the next packet
sending.... 8
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 9 to 13 Now Resending the same packet
```

Command Prompt

```
sending.... 5
ACK 5
Acknowledgement 5 Received! The sliding window is in the range 6 to 10 Now sending the next packet
sending.... 6
ACK 6
Acknowledgement 6 Received! The sliding window is in the range 7 to 11 Now sending the next packet
sending.... 7
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 8 to 12 Now Resending the same packet
sending.... 7
ACK 7
Acknowledgement 7 Received! The sliding window is in the range 8 to 12 Now sending the next packet
sending.... 8
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 9 to 13 Now Resending the same packet
sending.... 8
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 9 to 13 Now Resending the same packet
sending.... 8
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 9 to 13 Now Resending the same packet
sending.... 8
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 9 to 13 Now Resending the same packet
sending.... 8
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 9 to 13 Now Resending the same packet
sending.... 8
ACK 8
Acknowledgement 8 Received! The sliding window is in the range 9 to 13 Now sending the next packet
sending.... 9
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 10 to 14 Now Resending the same packet
sending.... 9
ACK 9
Acknowledgement 9 Received! The sliding window is in the range 10 to 14 Now sending the next packet
sending.... 10
ACK 10
Acknowledgement 10 Received! The sliding window is in the range 11 to 14 Now sending the next packet
sending.... 11
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 12 to 14 Now Resending the same packet
sending.... 11
ACK 11
Acknowledgement 11 Received! The sliding window is in the range 12 to 14 Now sending the next packet
sending.... 12
ACK 12
Acknowledgement 12 Received! The sliding window is in the range 13 to 14 Now sending the next packet
sending.... 13
ACK 13
Acknowledgement 13 Received! The sliding window is in the range 14 to 14 Now sending the next packet
Enter Data : bye
```

D:\Desktop\CN\CN Project>\_

## Receiver Side Output:

```

C:\> Command Prompt

Welcome to Chat Room

Initialising....

MR-ANNAM-123 ( 192.168.164.1 )

Enter server address: 192.168.164.1

Enter your name: sai

Trying to connect to 192.168.164.1 ( 1234 )

Connected...

annam has joined the chat room
Enter 'bye' to exit chat room

Received bit: 0
Acknowledged....) 0
NACK....( 1
NACK....( 1
NACK....( 1
Received bit: 1
Acknowledged....) 1
NACK....( 2
NACK....( 2
Received bit: 2
Acknowledged....) 2
Received bit: 3
Acknowledged....) 3
Received bit: 4
Acknowledged....) 4
Received bit: 5
Acknowledged....) 5
Received bit: 6
Acknowledged....) 6
NACK....( 7
Received bit: 7
Acknowledged....) 7
NACK....( 8
NACK....( 8
NACK....( 8
NACK....( 8
NACK....( 8
Received bit: 8
Acknowledged....) 8
NACK....( 9
Received bit: 9
Acknowledged....) 9
Received bit: 10
Acknowledged....) 10
NACK....( 11
Received bit: 11
Acknowledged....) 11
Received bit: 12
Acknowledged....) 12
Received bit: 13
Acknowledged....) 13
The message received is : ab
Traceback (most recent call last):
  File "receiver.py", line 39, in <module>
    k=int(k)
ValueError: invalid literal for int() with base 10: 'Left chat room!'

```

## Window Size 7:

### Sender Side Output:

```
Command Prompt
D:\Desktop\CN\CN Project>python sender.py

Welcome to Chat Room

Initialising...

MR-ANNAM-123 ( 192.168.164.1 )

Enter your name: annam

Waiting for incoming connections...

Received connection from 192.168.164.1 ( 60217 )


sai has connected to the chat room
Enter 'bye' to exit chat room

Enter Data : ab
['1100001', '1100010']
Enter the window size -> 7
sending.... 0
sending.... 1
sending.... 2
sending.... 3
sending.... 4
sending.... 5
sending.... 6
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 1 to 7 Now Resending the same packet
ACK 0
Acknowledgement 0 Received! The sliding window is in the range 1 to 7 Now sending the next packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 2 to 8 Now Resending the same packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 2 to 8 Now Resending the same packet
ACK 1
Acknowledgement 1 Received! The sliding window is in the range 2 to 8 Now sending the next packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 3 to 9 Now Resending the same packet
ACK 2
Acknowledgement 2 Received! The sliding window is in the range 3 to 9 Now sending the next packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 4 to 10 Now Resending the same packet
ACK 3
Acknowledgement 3 Received! The sliding window is in the range 4 to 10 Now sending the next packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 5 to 11 Now Resending the same packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 5 to 11 Now Resending the same packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 5 to 11 Now Resending the same packet
```

```
Command Prompt
Acknowledgement of the data bit is LOST! The sliding window remains in the range 5 to 11 Now Resending the same packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 5 to 11 Now Resending the same packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 5 to 11 Now Resending the same packet
ACK 4
Acknowledgement 4 Received! The sliding window is in the range 5 to 11 Now sending the next packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 6 to 12 Now Resending the same packet
ACK 5
Acknowledgement 5 Received! The sliding window is in the range 6 to 12 Now sending the next packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 7 to 13 Now Resending the same packet
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 7 to 13 Now Resending the same packet
ACK 6
Acknowledgement 6 Received! The sliding window is in the range 7 to 13 Now sending the next packet
sending.... 7
ACK 7
Acknowledgement 7 Received! The sliding window is in the range 8 to 14 Now sending the next packet
sending.... 8
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 9 to 14 Now Resending the same packet
sending.... 8
ACK 8
Acknowledgement 8 Received! The sliding window is in the range 9 to 14 Now sending the next packet
sending.... 9
ACK 9
Acknowledgement 9 Received! The sliding window is in the range 10 to 14 Now sending the next packet
sending.... 10
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 11 to 14 Now Resending the same packet
sending.... 10
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 11 to 14 Now Resending the same packet
sending.... 10
ACK 10
Acknowledgement 10 Received! The sliding window is in the range 11 to 14 Now sending the next packet
sending.... 11
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 12 to 14 Now Resending the same packet
sending.... 11
ACK 11
Acknowledgement 11 Received! The sliding window is in the range 12 to 14 Now sending the next packet
sending.... 12
ACK Lost
Acknowledgement of the data bit is LOST! The sliding window remains in the range 13 to 14 Now Resending the same packet
sending.... 12
ACK 12
Acknowledgement 12 Received! The sliding window is in the range 13 to 14 Now sending the next packet
sending.... 13
ACK 13
Acknowledgement 13 Received! The sliding window is in the range 14 to 14 Now sending the next packet
Enter Data : bye
```

## **Receiver Side Output:**

```

 Select Command Prompt
Initialising....
MR-ANNAM-123 ( 192.168.164.1 )
Enter server address: 192.168.164.1
Enter your name: sai
Trying to connect to 192.168.164.1 ( 1234 )
Connected...
annam has joined the chat room
Enter 'bye' to exit chat room

NACK....( 0
Received bit: 0
Acknowledged....) 0
NACK....( 1
NACK....( 1
Received bit: 1
Acknowledged....) 1
NACK....( 2
Received bit: 2
Acknowledged....) 2
NACK....( 3
Received bit: 3
Acknowledged....) 3
NACK....( 4
NACK....( 4
NACK....( 4
NACK....( 4
Received bit: 4
Acknowledged....) 4
NACK....( 5
Received bit: 5
Acknowledged....) 5
NACK....( 6
NACK....( 6
Received bit: 6
Acknowledged....) 6
Received bit: 7
Acknowledged....) 7
NACK....( 8
Received bit: 8
Acknowledged....) 8
Received bit: 9
Acknowledged....) 9
NACK....( 10
NACK....( 10
Received bit: 10
Acknowledged....) 10
NACK....( 11
Received bit: 11
Acknowledged....) 11
NACK....( 12
Received bit: 12
Acknowledged....) 12
Received bit: 13
Acknowledged....) 13
The message received is : ab

```



## **Algorithm:**

### **Selective Repeat Sender:**

$S_w = 2^{m-1}$ ,

$S_f = 0$ ,

$S_n = 0$ ,

While(true)

{

    WaitForEvent();

    if(Event(RequestToSend))

    {

        if( $S_n - S_f \geq S_w$ )

            Sleep();

        GetData();

        MakeFrame( $S_n$ );

        StoreFrame( $S_n$ );

        SendFrame( $S_n$ );

$S_n = S_{n+1}$ ;

        StartTimer( $S_n$ );

    }

If (Event(ArrivalNotification))

{

    Receive(frame);

    if(corrupted(frame))

        Sleep();

    if(FrameType == NACK)

        if( NACKNo between  $S_f$  and  $S_n$  )

        {

            resend(NACKNo);

            StartTimer(NACKNo);

        }

    if(FrameType == ACK)

        if(ackNo between  $S_f$  and  $S_n$ )

        {

```

        While( $S_f$  and  $S_n$ )
        {
            Purge( $S_f$ );
            StopTimer( $S_f$ );
             $S_f = S_f + 1$ ;
        }
    }
}

if(Event (TimeOut(t)))
{
    StartTimer(t);
    SendFrame(t);
}
}

```

### **Selective Repeat Receiver:**

```

 $R_n = 0$ ;
NackSent = false;
AckNeeded = false;
Repeat(for all slots)
    Marked(slot) = false;
While(true)
{
    WaitForEvent();
    If(Event(ArrivalNotification))
    {
        Receive(Frame);
        If(corrupted(Frame)) && (NOT NackSent)
        {
            SendNACK( $R_n$ );
            NACKSent = true;
            Sleep();
        }
        if(seqNo  $\neq$   $R_n$ ) && (NOT NackSent)
    }
}

```

```

{
    Receive(Frame);
    if(corrupted(Frame)) && (NOT NackSet)
    {
        SendNACK(Rn);
        NackSent = true;
        Sleep();
    }
    if (SeqNo <> Rn ) && (NOT NackSent)
    {
        SendNACK(Rn);
        NackSent = true;
        if((seqNo in Window) && (!Marked (seqNo))
        {
            StoreFrame(seqNo)
            Marked(seqNo) = true;
            While(Marked(Rn))
            {
                DeliverData(Rn);
                Purge(Rn);
                Rn = Rn + 1;
                AckNeeded = true;
            }
            if(AckNeeded)
            {
                SendAck(Rn);
                AckNeeded = false;
                NackSent = false;
            }
        }
    }
}
}

```

## Selective Repeat:

### Sender Code:

```
import socket
import time
def sender():
    print("Initialising...\n")
    n=int(input("Enter Window Size -->"));
    W_S = 0
    W_E = W_S + n - 1
    s_host = socket.gethostname() # as both code is running on same pc

    ip_addr = socket.gethostbyname(s_host)
    print(s_host, "(", ip_addr, ")\n")
    host_name = input(str("Enter server address: "))
    #Server port Number
    port_no = 12344
    print("\nConnecting to host: ", host_name, "at port:", port_no, "\n")
    time.sleep(1)
    sender = []
    #Send whole list if 0 else only start window frame
    f = 0
    #Creating a socket
    c_sock = socket.socket()
    #Connecting to the Server
    c_sock.connect((host_name, port_no))
    print("Connected\n")
    print ('***** Enter "bye" to close connection *****')
    message = input("Hit any key to send required frames -> ") # take input
    while message.lower().strip() != 'bye':
        print ("Sending frames...")
        if (f == 0):

            for i in range(n):
```

```

        sender.append(W_S + i)
    for i in sender :
        time.sleep(1)
        print ("Frame -> ", i)
    else:
        print ("Frame -> ", W_S)
    msg = str(W_S)
    # sending message to server
    c_sock.send(msg.encode())
    rec_data = c_sock.recv(1024).decode() # receive NA
    msg = str(rec_data)
    ack = int(msg)
    if ack not in sender:
        W_S = ack
        W_E = W_S + n - 1
        #sending new frame
        f = 0
        print("-----window slided-----")
        for i in range(n):
            sender.pop()
    else:
        W_S = int(msg)
        #sending old frame
        f = 1

    print ("*****")
    print ('Received ACK server: ' + rec_data)
    # again taking input
    message = input("Hit any key to send required frames -> ")
    # closing the connection
    c_sock.close()

sender()

```

## Receiver Code:

```
import socket
import random
import time

def receiver():
    #hostname

    host_addr = socket.gethostname()
    ip = socket.gethostbyname(host_addr)

    #port no initiating
    port_no = 12344
    exp = 0

    w_size=int(input("Enter Window Size-->"))
    new = 1
    W_S = 0
    W_E = W_S + w_size - 1
    rec = []

    # server instance
    s_socket = socket.socket()
    # bind host address and port together
    s_socket.bind((host_addr, port_no))

    # no of clients listening simultaneously
    s_socket.listen(1)
    print("\nWaiting for incoming connections...\n")

    # accepting the connection
    conn, addr = s_socket.accept()
    time.sleep(1)
    print ("\nReceived Connection from: ", addr[0], "(" ,addr[1],")\n")
    while (True):
        rec_data = conn.recv(1024).decode()
        if not rec_data:    # break when data is not received
            break
        temp = 0
```

```

cnt = 0
recieve = int(rec_data)
ackn = recieve
limit = recieve + w_size - 1

ran = random.randint(1, 3)

#received new frame
if new == 1 :
    while(cnt != ran):
        temp = random.randint(recieve, limit)
        time.sleep(1)
        if temp not in rec:
            print ("Received Frame -> ", temp)
            cnt+=1

            temp = 1
            rec.append(temp)
    else :
        time.sleep(1)

        #received a new frame of an old window
        print ("Received Frame -> ", recieve)
        rec.append(recieve)
        temp = 1
if(temp == 1):
    for i in range(recieve,limit+1):
        if i not in rec:
            ackn = i
            break
        ackn = i+1
#next expected frame
print ("Sending ACK    -> ", ackn)
time.sleep(1)

```

```

print ('*****')
rec_data = str(ackn)

# sending data to the client
conn.send(rec_data.encode())

if ackn > W_E :
    W_S = ackn
    W_E = W_S + w_size - 1
    print("----window Slided----")
    new = 1
else :
    # now received a new frame of an old window
    new = 0

# closing the connection
conn.close()
receiver()

```



## Window Size :5

### Sender Side Output:

```
Command Prompt

D:\Desktop\CN\Project\send-receive\sR>python sender.py
Initialising...

Enter Window Size -->5
MR-ANNAM-123 ( 192.168.164.1 )

Enter server address: 192.168.164.1

Trying to connect to 192.168.164.1 ( 12344 )

Connected

***** Enter "bye" to close connection *****
Hit any key to start sending frames ->
Sending frames...
Frame -> 0
Frame -> 1
Frame -> 2
Frame -> 3
Frame -> 4
*****
Received ACK server: 2
Hit any key to start sending frames ->
Sending frames...
Frame -> 2
*****
Received ACK server: 3
Hit any key to start sending frames ->
Sending frames...
Frame -> 3
-----window slided-----
Received ACK server: 5
Hit any key to start sending frames ->
Sending frames...
Frame -> 5
Frame -> 6
Frame -> 7
Frame -> 8
Frame -> 9
*****
Received ACK server: 5
Hit any key to start sending frames ->
Sending frames...
Frame -> 5
*****
Received ACK server: 6
Hit any key to start sending frames ->
Sending frames...
Frame -> 6
*****
Received ACK server: 8
Hit any key to start sending frames ->
Sending frames...
Frame -> 8
*****
Received ACK server: 9
Hit any key to start sending frames ->
Sending frames...
Frame -> 9
```

### Receiver Side Output:

```
Command Prompt

D:\Desktop\CN\Project\send-receive\sR>python receiver.py
Enter Window Size-->5

Waiting for incoming connections...

Received Connection from: 192.168.164.1 ( 54603 )

Received Frame -> 4
Received Frame -> 0
Received Frame -> 1
Sending ACK -> 2
*****
Received Frame -> 2
Sending ACK -> 3
*****
Received Frame -> 3
Sending ACK -> 5
*****
---window Slided---
Received Frame -> 7
Sending ACK -> 5
*****
Received Frame -> 5
Sending ACK -> 6
*****
Received Frame -> 6
Sending ACK -> 8
*****
Received Frame -> 8
Sending ACK -> 9
*****
Received Frame -> 9
Sending ACK -> 10
*****
---window Slided---
Received Frame -> 14
Received Frame -> 12
Received Frame -> 11
Sending ACK -> 10
*****
Received Frame -> 10
Sending ACK -> 13
*****
Received Frame -> 13
Sending ACK -> 15
*****
---window Slided---
```

```
D:\Desktop\CN\Project\send-receive\sR>
```

## Sender Side Output:

```

C:\> Command Prompt
Hit any key to start sending frames ->
Sending frames...
Frame -> 8
*****
Received ACK server: 9
Hit any key to start sending frames ->
Sending frames...
Frame -> 9
-----window slided-----
*****
Received ACK server: 10
Hit any key to start sending frames ->
Sending frames...
Frame -> 10
Frame -> 11
Frame -> 12
Frame -> 13
Frame -> 14
*****
Received ACK server: 10
Hit any key to start sending frames ->
Sending frames...
Frame -> 10
*****
Received ACK server: 13
Hit any key to start sending frames ->
Sending frames...
Frame -> 13
-----window slided-----
*****
Received ACK server: 15
Hit any key to start sending frames -> bye
D:\Desktop\CN\Project\send-receive\sR>

```

## Window Size: 7

### Sender Side Output:

```
C:\> Command Prompt

D:\Desktop\CN\Project\send-receive\sR>python send.py
Initialising....

Enter Window Size -->7
MR-ANNAM-123 ( 192.168.164.1 )

Enter server address: 192.168.164.1

Trying to connect to 192.168.164.1 ( 12344 )

Connected

***** Enter "bye" to close connection *****
Hit any key to start sending frames ->
Sending frames...
Frame -> 0
Frame -> 1
Frame -> 2
Frame -> 3
Frame -> 4
Frame -> 5
Frame -> 6
*****
Received ACK server: 0
Hit any key to start sending frames ->
Sending frames...
Frame -> 0
*****
Received ACK server: 1
Hit any key to start sending frames ->
Sending frames...
Frame -> 1
*****
Received ACK server: 5
Hit any key to start sending frames ->
Sending frames...
Frame -> 5
-----window slided-----
*****
Received ACK server: 7
Hit any key to start sending frames ->
Sending frames...
Frame -> 7
Frame -> 8
Frame -> 9
Frame -> 10
Frame -> 11
Frame -> 12
```

### Receiver Side Output:

```
C:\> Command Prompt

D:\Desktop\CN\Project\send-receive\sR>python receiver.py
Enter Window Size-->7

Waiting for incoming connections...

Received Connection from: 192.168.164.1 ( 56555 )

Received Frame -> 3
Received Frame -> 4
Received Frame -> 6
Received Frame -> 2
Sending ACK -> 0
*****
Received Frame -> 0
Sending ACK -> 1
*****
Received Frame -> 1
Sending ACK -> 5
*****
Received Frame -> 5
Sending ACK -> 7
*****
----window Slided----
Received Frame -> 12
Received Frame -> 8
Received Frame -> 7
Sending ACK -> 9
*****
Received Frame -> 9
Sending ACK -> 10
*****
Received Frame -> 10
Sending ACK -> 11
*****
Received Frame -> 11
Sending ACK -> 13
*****
Received Frame -> 13
Sending ACK -> 14
*****
----window Slided----
Received Frame -> 20
Sending ACK -> 14
*****
Received Frame -> 14
Sending ACK -> 15
*****
```

## Sender Side Output:

```
C:\> Command Prompt
Frame -> 10
Frame -> 11
Frame -> 12
Frame -> 13
*****
Received ACK server: 9
Hit any key to start sending frames ->
Sending frames...
Frame -> 9
*****
Received ACK server: 10
Hit any key to start sending frames ->
Sending frames...
Frame -> 10
*****
Received ACK server: 11
Hit any key to start sending frames ->
Sending frames...
Frame -> 11
*****
Received ACK server: 13
Hit any key to start sending frames ->
Sending frames...
Frame -> 13
-----window slided-----
*****
Received ACK server: 14
Hit any key to start sending frames ->
Sending frames...
Frame -> 14
Frame -> 15
Frame -> 16
Frame -> 17
Frame -> 18
Frame -> 19
Frame -> 20
*****
Received ACK server: 14
Hit any key to start sending frames ->
Sending frames...
Frame -> 14
*****
Received ACK server: 15
Hit any key to start sending frames ->
Sending frames...
Frame -> 15
*****
Received ACK server: 16
Hit any key to start sending frames ->
```

## Receiver Side Output:

```
C:\> Command Prompt
*****
Received Frame -> 14
Sending ACK -> 15
*****
Received Frame -> 15
Sending ACK -> 16
*****
Received Frame -> 16
Sending ACK -> 17
*****
Received Frame -> 17
Sending ACK -> 18
*****
Received Frame -> 18
Sending ACK -> 19
*****
Received Frame -> 19
Sending ACK -> 21
*****
----window Slided----
D:\Desktop\CN\Project\send-receive\sR>
```

## Sender Side Output:

```

C:\> Command Prompt
Hit any key to start sending frames ->
Sending frames...
Frame -> 15
*****
Received ACK server: 16
Hit any key to start sending frames ->
Sending frames...
Frame -> 16
*****
Received ACK server: 17
Hit any key to start sending frames ->
Sending frames...
Frame -> 17
*****
Received ACK server: 18
Hit any key to start sending frames ->
Sending frames...
Frame -> 18
*****
Received ACK server: 19
Hit any key to start sending frames ->
Sending frames...
Frame -> 19
-----window slided-----
*****
Received ACK server: 21
Hit any key to start sending frames -> bye

D:\Desktop\CN\Project\send-receive\sR>
```

## **CHAPTER 5**

### **PERFORMANCE EVALUATION**

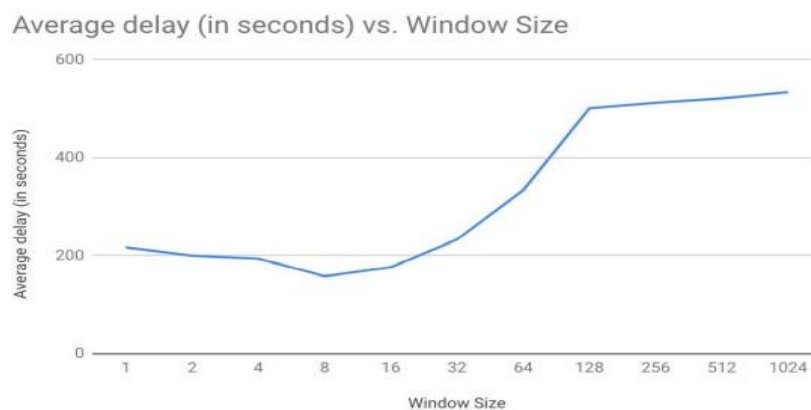
#### **Effect of Window Size:**

**In GBN:**

Window Size(N)	Average Delay(in seconds)
1	217
2	200
4	194
8	158
16	176
32	233
64	334
128	501
256	512
512	534
1024	534

**Table 1**

**Graph:**



**Graph 1**

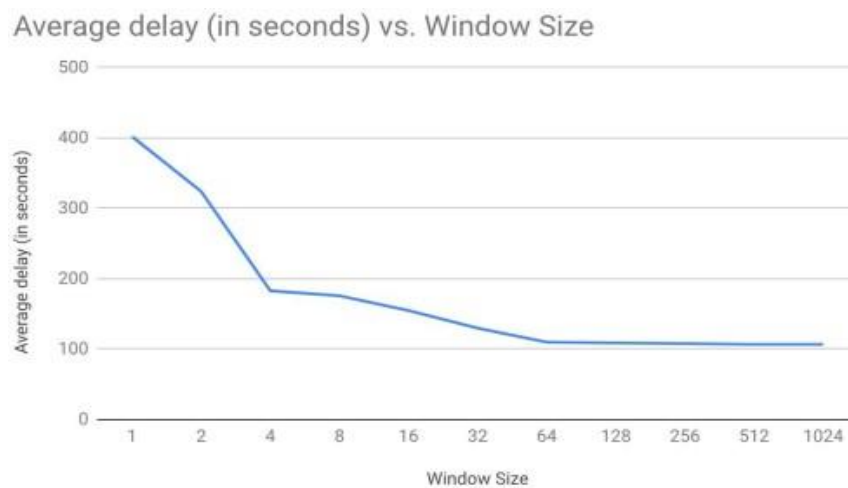
**Observation:** When  $N=1$ , average delay is the highest because client has to wait for every packet's acknowledgement before sending another packet. This is due to Stop-and-wait scenario, and therefore it wastes a lot of time and resources because typically clients and servers are capable of handling more than a single packet. As we increase  $N$ , we can see transfer time decreases as we are sending/receiving many packets per Round Trip Time. Also, towards the end we can see that average transfer time starts increasing again. This shows that beyond a certain value, if we still keep increasing  $N$ , delay will start increasing again as many packets are lost due to congestion

### In Selective Repeat:

Window Size(N)	Average Delay(in seconds)
1	402
2	324
4	183
8	176
16	155
32	130
64	110
128	109
256	108
512	107
1024	107

Table 2

### Graph:



Graph 2

**Observation:** As exemplified by the graph, the average delay is consistently decreasing with increase in window size. This graph is not exactly same as in the case of Go-Back-N where after a certain value of N delay started to increase. Due to the fact that retransmissions are selective, the decrease in the average delay is continuous

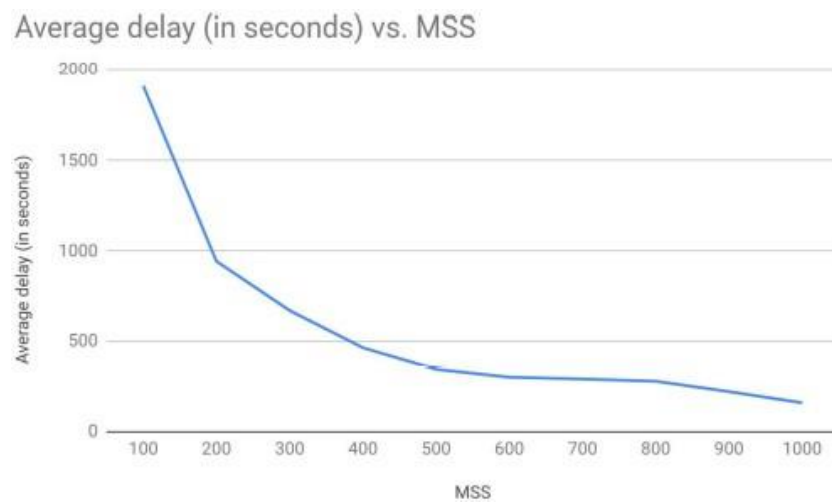
## Effect of MSS:

In GBN:

MSS	Average Delay (in seconds)
100	1914
200	943
300	670
400	465
500	345
600	302
700	291
800	279
900	222
1000	160

Table 3

Graph:



Graph 3

**Observation:** Referring to the chart, increasing maximum segment size reduces the time to transfer the file. The reason is that if  $N$  is fixed and MSS is increased, fewer packets have to be sent to transfer the file (i.e. more data sent per RTT)

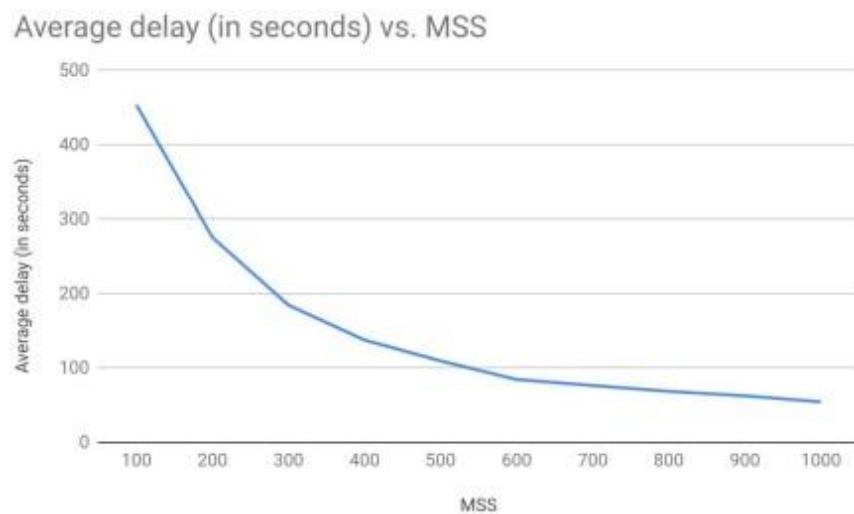


**IN SR:**

MSS	Average Delay(in sec)
100	454
200	276
300	185
400	138
500	110
600	85
700	77
800	69
900	63
1000	55

**Table 4**

**Graph:**



**Graph 4**

**Observation:** From the chart, when  $N=1$ , transfer time is the highest because client has to wait for every packet's acknowledgement before sending another packet. As  $N$  increases, the transfer time decreases because of sending/receiving many packets per Round Trip Time. Also, it is evident that for higher value of  $N$ , there is significant performance difference between Go-back- $N$  and Selective Repeat ARQ because in Selective repeat, fewer packets need to be retransmitted.

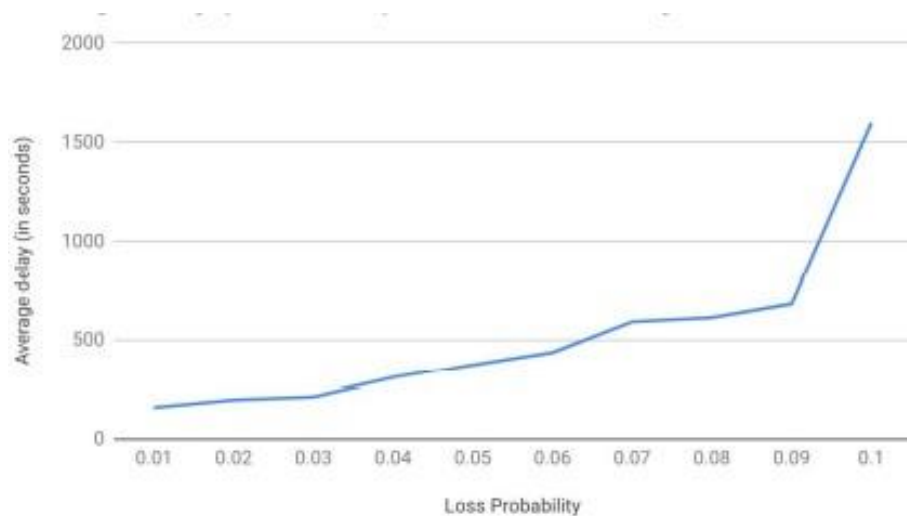
### Effect of loss probability :

#### In GBN

Loss Probability	Average Delay(in sec)
0.01	159
0.02	198
0.03	213
0.04	305
0.05	373
0.06	437
0.07	594
0.08	615
0.09	684
0.10	1598

Table 5

#### Graph:



Graph 5

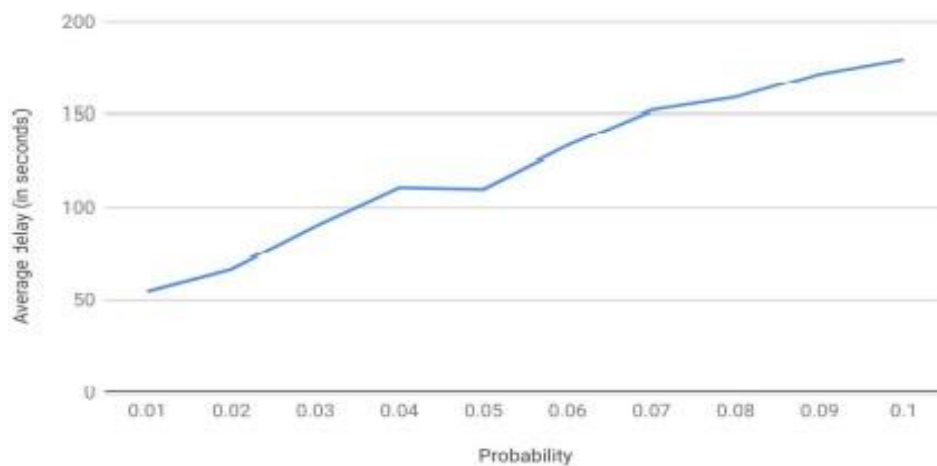
**Observation:** As depicted by the graph, increasing loss probability increases the average time to transfer the file. Naturally more packets need to be retransmitted due to packet loss. For instance, if the loss probability is increased to 0.5 while keeping N & MSS the same, 500 per 1000 packets will need to be retransmitted per RTT which will massively increase the delay in the transfer.

**In SR:**

Loss Probability p	Average Delay (in sec)
0.01	55
0.02	67
0.03	90
0.04	111
0.05	110
0.06	133
0.07	152
0.08	159
0.09	172
0.10	180

**Table 6**

**Graph:**



**Graph 6**

**Observation:** Increasing loss probability increases the average time to transfer the file because more packets need to be retransmitted due to packet loss.

## **DISADVANTAGES OF GBN:**

- A single error, in the 1st packet, will make the rest of the successive packets, transmitted as waste. And will lead to RE-TRANSMISSION, which obviously can be avoided.
- When window size is large, and there is high congestion in network, the probability of 1st packet of a retransmission getting dropped, by a intermediate router is **high**. This adds to ,number of retransmissions, in a **already congested channel**.
- The window size of 1 in receiver side, and large window size, on sender side, perform extremely bad, in a congested network (where chances of a packet, getting dropped is high).
- It does not utilize the benefit of **buffers**, that can be added to both hosts, which will **reduce**, the **discarding** of ,**rightly** received packets(which were received out of **sequence**).
- With buffers being introduced, the receiver, no longer needs to limit its window size to 1.
- ALL THESE IDEAS, ARE INCORPORATED IN SR PROTOCOL.

## **DISADVANTAGES OF SR:**

- **Hardware limitations**, play a big role in deciding the **window size**, as each sequence number, within the sender side window, **needs to be allotted a timer**, which must be capable of running simultaneously.(all N timers, running simultaneously).
- **Hardware limitations**, in another form, will be the **amount of Buffer, available especially for the RECEIVER side**, this factor will limit the window size.
- Congestion in the channel, is another factor, based on which , window size, should be determined. But Congestion changes with time, thus is un-predictable

## CHAPTER 6

### CONCLUSION

The sliding window protocol is known as the error control protocol. Both types of sliding window protocols are effective in various situations. The protocols use the method of **Error Detection and Error Correction at the Data Link Layer**. The elementary difference

between **Go Back N ARQ** and **Selective Repeat ARQ** is Go Back N ARQ retransmits all frames, and Selective Repeat ARQ only retransmits the required and damaged frame.

The users can use a protocol based on sender and receiver CPU power, Bandwidth, and buffer. However, Selective Repeat ARQ is more effective than Go Back N ARQ because it uses less Bandwidth, and all the frames will be received by the receiver without any difficulty.

#### **Future plans/Improvements Possible:**

- Throughout it was assumed that, packet cannot be reordered, in the underlying channel between sender and receiver, however it need not hold true ,always.
- When sender, receiver are connected by a single wire, than the above assumption will mostly hold, but if the underlying channel is a network, then packet reordering is possible.
- In such a scenario, the sender needs a bit more programming. The sender needs to ensure, before using a particular SEQUENCE NUMBER, that the packet, previously sent with that sequence number, no longer exists in network.
- This is achieved by adding a additional field in packet header.(this is done by network layer).
- These are:
  1. In IPv4: it is TIME TO LIVE(TTL), field
  2. In IPv6: it is HOP LIMIT field
- These fields are decremented , when the packet passes through any intermediate router, when these field becomes 0, the packet is DROPPED.
- These ensures that a packet, does not move around INDEFINITELY.
- As already mentioned, these headers are included on the segment, in network layer, thus it was not shown in GBN,SR protocol application.

## CHAPTER 7

### REFERENCES

- **Computer Networking** : Kurose, James F. Computer networking : a top-down approach / James F. Kurose, Keith W. Ross.—6th ed. p. cm(**Prescribed Textbook, for this course**)
- Tanenbaum, Andrew S., 1944- Computer networks / Andrew S. Tanenbaum, David J. Wetherall. -- 5th ed. p. cm
- Tanenbaum, A.S., Computer Networks, 3rd Edition, Prentice Hall PTR, 1996, pp. 212250.
- Henry, D., “Master’s Project: A Visualization System for Sliding Windows Protocols”, Colorado State University Technical Report, available at <http://www.cs.colostate.edu/testing/>, 2002.
- <https://www.baeldung.com/cs/networking-go-back-n-protocol>
- Chun, Wesley. Core python applications programming / Wesley J. Chun. — 3rd ed.w p. cm. (**Prescribed TextBook for Python Course**)
- Weldon, E. (March 1982). "An Improved Selective-Repeat ARQ Strategy". *IEEE Transactions on Communications*. **30** (3): 480-486.  
[Bibcode:1982ITCom..30..480W](#). [doi:10.1109/TCOM.1982.1095497](#). [ISSN 0090-6778](#).
- Python Basics: A Practical Introduction to Python 3 Revised and Updated 4th Edition  
David Amos, Dan Bader, Joanna Jablonski, Fletcher Heisl
- <https://ieeexplore.ieee.org/document/7754487>
- <https://ieeexplore.ieee.org/document/368975>
- <https://ieeexplore.ieee.org/document/1096499>