Experiment No -06

AIM: To demonstrate Go back N and Selective Repeat Modes of Sliding Window Protocol in peer to peer mode.

OBJECTIVES:

- To study sliding window protocol.
- To implement Go back N and Selective Repeat Protocol.
- To differentiate between Go back N and Selective Repeat Protocol.

PROBLEM STATMENT:
Write a program to simulate Go back N and Selective Repeat Modes of Sliding Window Protocol in peer to peer mode and demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode.

OUTCOMES:
CO2: Analyze data flow between peer to peer in an IP network using Application, Transport and Network Layer Protocols

SOFTWARE & HARDWARE REQUIREMENTS:

- Software: jdk compiler and wireshark.
- Hardware: PC-2.

THEORY:

The basic idea of sliding window protocol is that both sender and receiver keep a 'window' of acknowledgment. The sender keeps the value of expected acknowledgment; while the receiver keeps the value of expected receiving frame. When it receives an acknowledgment from the receiver, the sender advances the window. When it receives the expected frame, the receiver advances the window.

In transmit flow control, sliding window is a variable-duration window that allows a sender to transmit a specified number of data units before an acknowledgement is received or before a specified event occurs.

Flow Control is a set of procedures that tells the sender how much data it can transmit before it must wait for an acknowledgment from the receiver. The flow of data should not be allowed to

Overwhelm the receiver. Receiver should also be able to inform the transmitter before its limits (this limit may be amount of memory used to store the incoming data or the processing power at the receiver end) are reached and the sender must send fewer frames. Hence, Flow control refers to the set of procedures used to restrict the amount of data the transmitter can send before waiting for acknowledgment.

There are two methods developed for flow control namely Stop-and-wait and Sliding-window Sliding window algorithms, used by TCP, permit multiple data packets to be in simultaneous transit, making more efficient use of network bandwidth.

Sliding Window Protocol:

With the use of multiple frames for a single message, the stop-and-wait protocol does not perform well. Only one frame at a time can be in transit. 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. To keep track of the frames, sender station sends sequentially numbered frames. Since the sequence number to be used occupies a field in the frame, it should be of limited size. If the header of the frame allows k bits, the sequence numbers range from 0 to
$2^k - 1$. Sender maintains a list of sequence numbers that it is allowed to send (sender window).

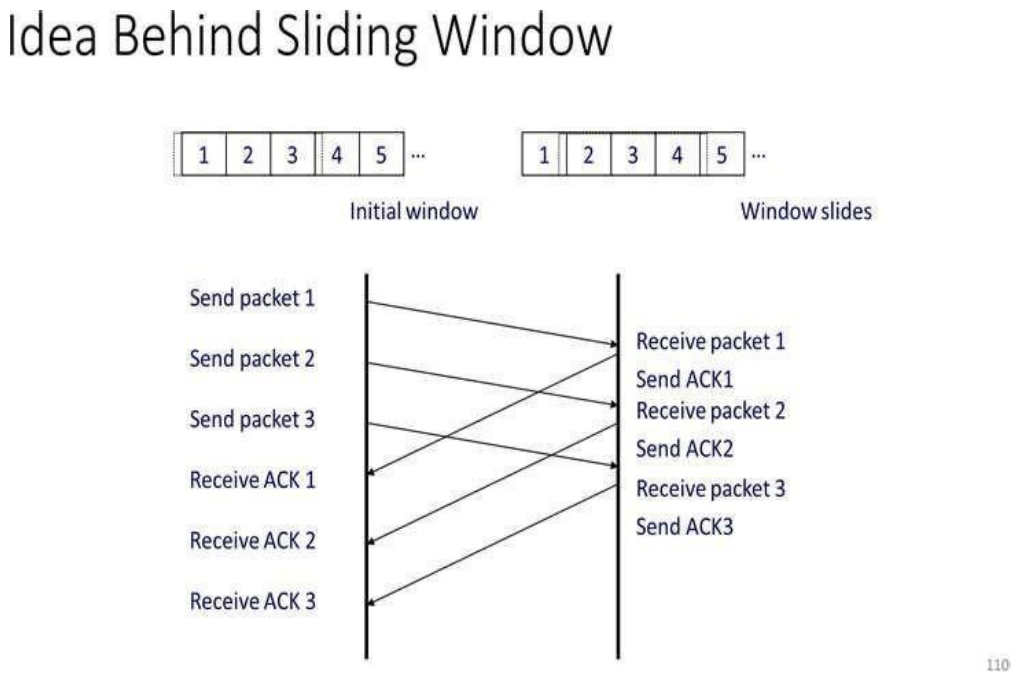
The size of the sender's window is at most $2^k - 1$. The sender is provided with a buffer equal to the window size. Receiver also maintains a window of size $2^k - 1$. The receiver acknowledges a frame by sending an ACK frame that includes the sequence number of the next frame expected. This also explicitly 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 in one go. The receiver needs a buffer of size 1.

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.

A sliding window algorithm places a buffer between the application program and the network data flow. For TCP, the buffer is typically in the operating system kernel, but this is more of an implementation detail than a hard-and-fast requirement.

Data received from the network is stored in the buffer, from where the application can read at its own pace. As the application reads data, buffer space is freed up to accept more input from the network. The window is the amount of data that can be "read ahead" - the size of the buffer, less the amount of valid data stored in it. Window announcements are used to inform the remote host of the current window size.

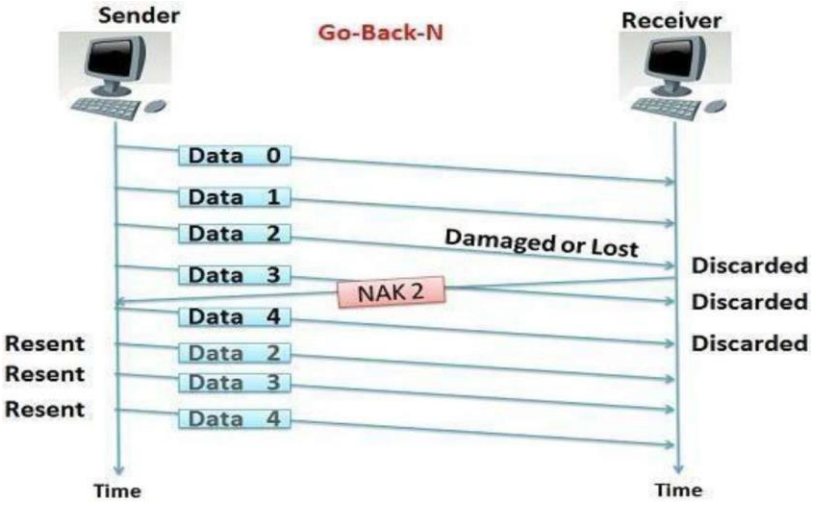
An example of a sliding window in packet transmission is one in which, after the sender fails to receive an acknowledgement for the first transmitted packet, the sender "slides" the window, i.e. resets the window, and sends a second packet. This process is repeated for the specified number of times before the sender interrupts transmission. Sliding window is sometimes (loosely) called *acknowledgement delay period*.



Go-Back-N Protocol and "Selective Repeat Protocol" are the sliding window protocols. The sliding window protocol is primarily an error control protocol, i.e. it is a method of error detection and error correction. The basic difference between go-back-n protocol and selective repeat protocol is that the "go-back-n protocol" retransmits all the frames that lie after the frame which is damaged or lost. The "selective repeat protocol" retransmits only that frame which is damaged or lost.

Go back N ARQ

In the Go-Back-N Protocol, the sequence numbers are modulo 1!", Where *m* is the size of the sequence number field in bits.



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, this protocol 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.

Key Differences between Go-Back-N and Selective Repeat

1. Go-Back-N protocol is design to retransmit all the frames that are arrived after the damaged or a lost frame. On the other hand, Selective Repeat protocol retransmits only that frame that is damaged or lost.
2. If the error rate is high i.e. more frames are being damaged and then retransmitting all the frames that arrived after a damaged frame waste the lots of bandwidth. On the other hand, selective repeat protocol re-transmits only damaged frame hence, minimum bandwidth is wasted.
3. All the frames after the damaged frame are discarded and the retransmitted frames arrive in a sequence from a damaged frame onwards, so, there is less headache of sorting the frames hence it is less complex. On the other hand only damaged or suspected frame is retransmitted so, extra logic has to be applied for sorting hence, it is more complicated.
4. Go-Back-N has a window size of N-1 and selective repeat have a window size <=(N+1)/2.
5. Neither sender nor receiver need the sorting algorithm in Go-Back-N whereas, receiver must be able to sort the as it has to maintain the sequence.
6. In Go-Back-N receiver discards all the frames after the damaged frame hence, it don't need to store any frames. Selective repeat protocol does not discard the frames arrived after the damaged frame instead it stores those frames till the damaged frame arrives successfully and is sorted in a proper sequence.
7. In selective repeat NAK frame refers to the damaged frame number and in Go-Back-N, NAK frame refers to the next frame expected.
8. Generally the Go-Back-N is more is use due to its less complex nature instead of Selective Repeat protocol.

Write a program to simulate Go back N and Selective Repeat Modes of Sliding Window Protocol in Peer-to-Peer mode

```
#include <stdio.h>

#include <conio.h>
#include <math.h>
int n, r;
struct frame
{
        char ack;
        int data;

} frm[10]; int sender(void); void
recvack(void); void resend_sr(void);
void resend_gb(void);
void selective(void);
```

```
void goback()
{
        sender();       recvack();
         sender();
            printf("\n all frames sent succesfully \n");
}
void selective()
{
        sender();       recvack();
resend_sr();
            printf("\n all frames sent succesfully \n");
}
int sender()
{
         int i,data;
        printf("\nEnter the no.of frames to be sent:");          scanf("%d", &n);
          for (i = 1; i <= n; i++)
        {
                  printf("\nEnter the data for frames [%d]", i);
                 scanf("%d", &frm[i], data);
                frm[i].ack = 'y';
        }
        return 0;
}

void recvack()
{
        int i;      rand();
          r = rand() % n;

          frm[r].ack = 'n';
           for (i =1;i <= n; i++)
        {
                if (frm[i].ack == 'n')
                        printf("\nThe frame number %d is not recieved\n", r);
        }
}
void resend_sr()
{
        printf("\nresending frame %d", r);    sleep(2);
          frm[r].ack = 'y';
            printf("\nThe received frame is %d", frm[r].data);
}
/*int resend_gb()
{
         int i,data;
        printf("\nEnter the no.of frames to be sent:");          scanf("%d", &n);
          for (i = 1; i <= n; i++)
        {
 printf("\nEnter the data for frames [%d]", i); scanf("%d", &frm[i], data);
                frm[i].ack = 'y';
```

```
}
```

```
        return 0;
}
/*void resend_gb()
{
        int i;
        printf("\nAgain send frame %d", r);  for (i = r;i <= n; i++)
        {
                sleep(2);
                frm[i].ack = 'y';
                        printf("\n received data of frame %d is %d", i, frm[r].data);
        }
}*/ int main()
{

        int c;
        do
        {
                        printf("\n\n1.Selective repeat ARQ\n2.GOback ARQ\n3.exit");

                printf("\nEnter your choice:");
 scanf("%d", &c);               switch (c)
                {
                case 1:
selective();
                        break;              case
2:                      goback();
        break;          case 3:
exit(0);
                         break;
                }
          } while (c >= 4);
}
```

Example: Python Simulation (Peer-to-Peer)

```
import socket
import threading
import random
import time

# Parameters
WINDOW_SIZE = 4
TOTAL_FRAMES = 10
LOSS_PROBABILITY = 0.2   # 20% chance of frame loss

# --- Sender Side ---
```

```
def sender(conn, mode="GBN"):
    base = 0
    next_seq = 0
    acked = [False] * TOTAL_FRAMES

    while base < TOTAL_FRAMES:
        while next_seq < base + WINDOW_SIZE and next_seq < TOTAL_FRAMES:
            # Simulate frame send
            if random.random() > LOSS_PROBABILITY:
                print(f"Sender: Sending Frame {next_seq}")
                conn.send(str(next_seq).encode())
            else:
                print(f"Sender: Frame {next_seq} lost in transmission")
            next_seq += 1

        time.sleep(2)  # wait for ACKs

        if mode == "GBN":
            # In Go-Back-N, move base only if ACK received for base
            if acked[base]:
                while base < TOTAL_FRAMES and acked[base]:
                    base += 1
        else:
            # In Selective Repeat, move base to next unacked
            while base < TOTAL_FRAMES and acked[base]:
                base += 1

    print("Sender: Transmission completed.")
    conn.send(b"END")

# --- Receiver Side ---
def receiver(conn, mode="GBN"):
    expected = 0
    received = [False] * TOTAL_FRAMES

    while True:
        data = conn.recv(1024).decode()
        if data == "END":
            break
        frame = int(data)
```

```
        if mode == "GBN":
            if frame == expected:
                print(f"Receiver: Received Frame {frame}, sending ACK")
                conn.send(f"ACK{frame}".encode())
                expected += 1
            else:
                print(f"Receiver: Out of order frame {frame}, discarding")
        else:  # Selective Repeat
            print(f"Receiver: Received Frame {frame}, sending ACK")
            conn.send(f"ACK{frame}".encode())
            received[frame] = True

# --- ACK Listener ---
def ack_listener(conn, acked):
    while True:
        ack = conn.recv(1024).decode()
        if ack.startswith("ACK"):
            frame = int(ack[3:])
            print(f"Sender: Received {ack}")
            acked[frame] = True
```

CONCLUSION: - Hence we have implemented of sliding window protocol (Go back N and Selective Repeat).