Selective Repeat Protocol

The go-back-n protocol works well if errors are rare, but if the line is poor it wastes a lot of bandwidth on retransmitted frames. An alternative strategy, the selective repeat protocol.

In this protocol, both sender and receiver maintain a window of outstanding and acceptable sequence numbers, respectively. The sender's window size starts out at 0 and grows to some predefined maximum.The receiver's window, in contrast, is always fixed in size and equal to the predetermined maximum. The receiver has a buffer reserved for each sequence number within its fixed window.

Code for Selective Repeat Protocol

```
#include<cstdlib>
#include<iostream>
#define Pipeline_Size 16
#define Max_No_of_Frames 32
using namespace std;
class Head
{
   private:

   int
Data_Set[Max_No_of_Frames]={23,24,35,335,46,416,464,53,524,37,76,7,6,787,8,78,67,64
,
                              65,75,742,727,37,27,272,72,727,466,174,321,62,22};
   int Data[Max_No_of_Frames], Pipeline[Pipeline_Size], Bits_Sent,
      Bits_Received, No_of_bits_to_send;

   public:

   int Negative_Ack;
   Head()
   {
      Bits_Sent=Bits_Received=0;
   }
   void Scan()
   {
      int No_of_Bits,No_of_Frames;
      cout<<"Enter the no of bits : ";
      cin>>No_of_Bits;
      No_of_Frames = 2<<(No_of_Bits-1);
      No_of_bits_to_send = Pipeline_Size;
      for(int j=0, Frame_No=0 ; j<Max_No_of_Frames ; Frame_No%=No_of_Frames)
         Data[j++]=Frame_No++;
      Event_Send(No_of_Frames);
```

```
    }

    void Event_Send(int Frames_Send)
    {
       cout <<"\nSender Message :\n";
       int k=0,i;
       if (Negative_Ack!=-1)
       {
          Pipeline[k]=Negative_Ack;
          cout<<"\nFrame No. "<<Pipeline[k++]<<" with data "<<Data_Set[i]<<" is resent.";
       }
       for (i=Bits_Sent ; i<Bits_Sent+No_of_bits_to_send ; )
       {
          Pipeline[k]=Data[i++];
          cout<<"\nFrame No. "<<Pipeline[k++]<<" with data "<<Data_Set[i]<<" is sent.";
       }
       Event_Receive(Frames_Send);
    }
    void  Event_Receive(int Frames_Received)
    {
        int Trans_Error,Lost_Index,Ack_Lost_Prob,Damage;
        bool cont;
        cout <<"\n\nReceiver Message :\n";
        Trans_Error = rand() % 5; // probability of transmission error = 1/5

        /* We have assumed if Trans_Error=0 then transmission error happened
        otherwise we received data correctly. */

        if(Trans_Error!=0)
        {
          for (int i=0 ; i<No_of_bits_to_send ; i++)
          {
             if (Bits_Received == Pipeline[i] || Bits_Received == Negative_Ack)
             {
                cout <<"\nFrame No. "<<Pipeline[i]<<" with data "<<Data_Set[i]<<" is
received correctly.";
                ++Bits_Received;
                //Bits_Received%=Frames_Received;
             }
             else
                cout << "\nDuplicate Frame No. "<<Pipeline[i]<<" with data
"<<Data_Set[i]<<" is received thus discarded.";
          }
          Ack_Lost_Prob = rand() % 5;

          /* We have assumed if Ack_Lost_Prob is between 0 and 5 then acknowledgments
          are lost otherwise we received all acknowledgments correctly. */
```

```
        if (Ack_Lost_Prob>=0 && Ack_Lost_Prob<5)
        {
           Ack_Lost_Prob = rand() % Pipeline_Size;
           cout << "\nAcknowledgment "<<Pipeline[Ack_Lost_Prob]<<" is lost.";
           //Bits_Sent = Bits_Received = Pipeline[Ack_Lost_Prob];
           Negative_Ack=Pipeline[Ack_Lost_Prob];
        }
        //else
         //   Bits_Sent=(Bits_Sent+No_of_bits_to_send)%Frames_Received;
      }
      else
      {
        Lost_Index=rand()%No_of_bits_to_send;

        // Lost_Index is the index of the frame being lost.

        for (int i = 0 ; i < Lost_Index ; i++)
        {
           if (Bits_Received == Pipeline[i])
           {
              cout<<"\nFrame No. "<<Pipeline[i]<<" with data "<<Data_Set[i]<<" is
recieved correctly.";
              ++Bits_Received;
              Bits_Received%=Frames_Received;
           }
           else
              cout<<"\nDuplicate frame "<<Pipeline[i]<<" is received thus discarded.";
        }

        Damage = rand() % 2;
        // If Damage == 0 Frame damaged otherwise Frame lost.
        if(!Damage)
           cout <<"\nFrame No. "<<Pipeline[Lost_Index]<<" is damaged.";
        else
           cout <<"\nFrame No. "<<Pipeline[Lost_Index]<<" is lost.";
        for (int i=Lost_Index+1;i<No_of_bits_to_send; i++)
           cout << "\nFrame  No. "<<Pipeline[i]<<" is received thus discarded.";
        cout<<"\nSender's Timeout thus Resend the Frame.";
        //Bits_Sent = Pipeline[Lost_Index];
      }
      cout << "\nEnter 1 to continue or 0 to abort : ";
      cin >> cont;
      if (cont == 1)
        Event_Send(Frames_Received);
      else
        exit(0);
   }
};
```

```
int  main()
{
   Head Selective_Repeat;
   Selective_Repeat.Negative_Ack=-1;
   Selective_Repeat.Scan();
}
```

Results :-