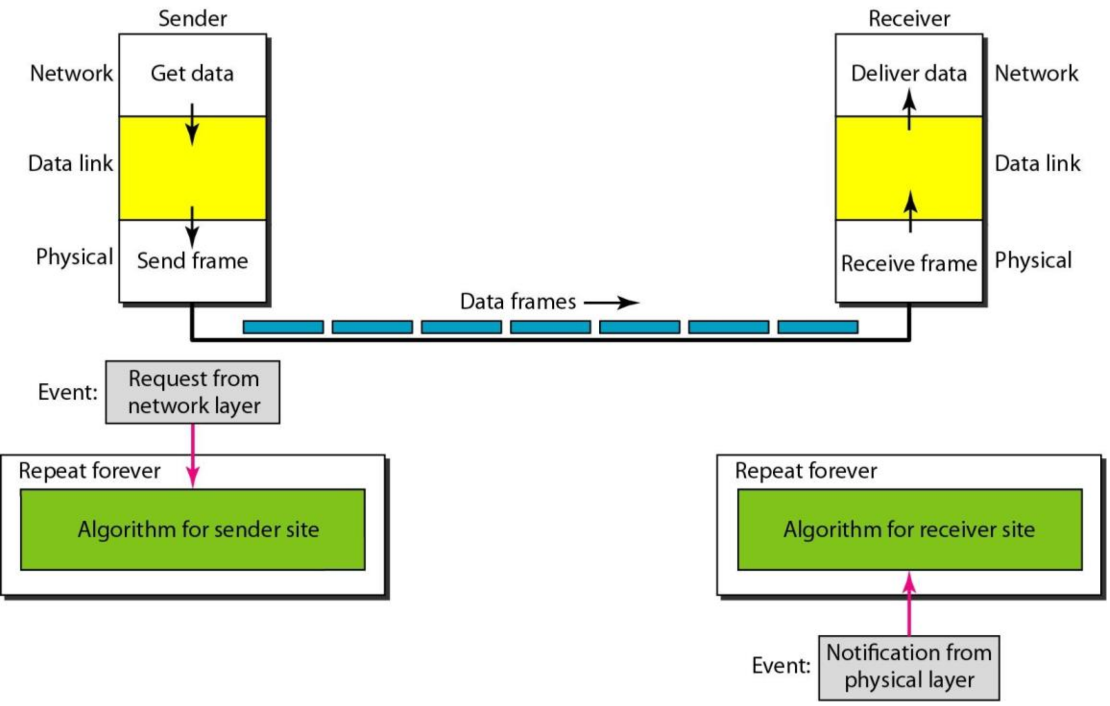
I. SIMPLEST PROTOCOL

1. *Introduction*

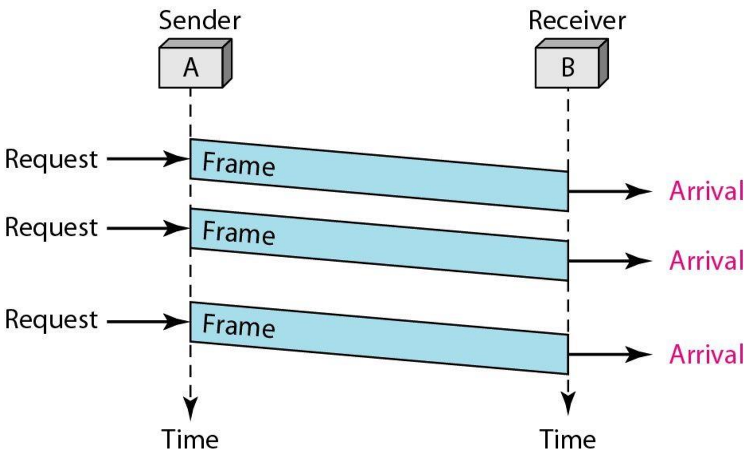
- Noiseless: Error free
- No Error Control
- No Flow Control
- **Assumption:**
  Receiver can immediately handle a frame
  Receiver cannot be overwhelmed
- Unidirectional

2. *Design Diagram*



3. *Flow diagram*

Figure shows an example of communication using this protocol.It is very simple. The sender sends a sequence of frames without even thinking about the receiver. To send three frames, three events occur at the sender site and three events at the receiver site. Note that the data frames are shown by tilted boxes; the height of the box defines the transmission time difference between the first bit and the last bit in the frame



4. Protocol

Algorithm 11.1 Sender-site algorithm for the simplest protocol

```
while(true)                         // Repeat forever
{
  WaitForEvent();                   // Sleep until an event occurs
  if(Event(RequestToSend))          //There is a packet to send
  {
     GetData();
     MakeFrame();
     SendFrame();                   //Send the frame
  }
}
```

Algorithm 11.2 Receiver-site algorithm for the simplest protocol

```
while(true)                         // Repeat forever
{
  WaitForEvent();                   // Sleep until an event occurs
  if(Event(ArrivalNotification))    //Data frame arrived
  {
     ReceiveFrame();
     ExtractData();
     DeliverData();                 //Deliver data to network layer
  }
}
```

II. STOP AND wait PROTOCOL

1. Introduction

1. It is provided by the data link layer of the OSI suite.
2. It uses a link between sender and receiver as a half-duplex link.
3. The flow control protocols ensure that the sender sends the data only at a rate that the receiver can receive and process it.
4. This is a flow control protocol that works in a noiseless channel.
5. Noiseless channel is an idealistic channel in which no data frames are lost, corrupted, or duplicated.
6. After sending the data, the sender will stop and waits until he receives an acknowledgment from the receiver.

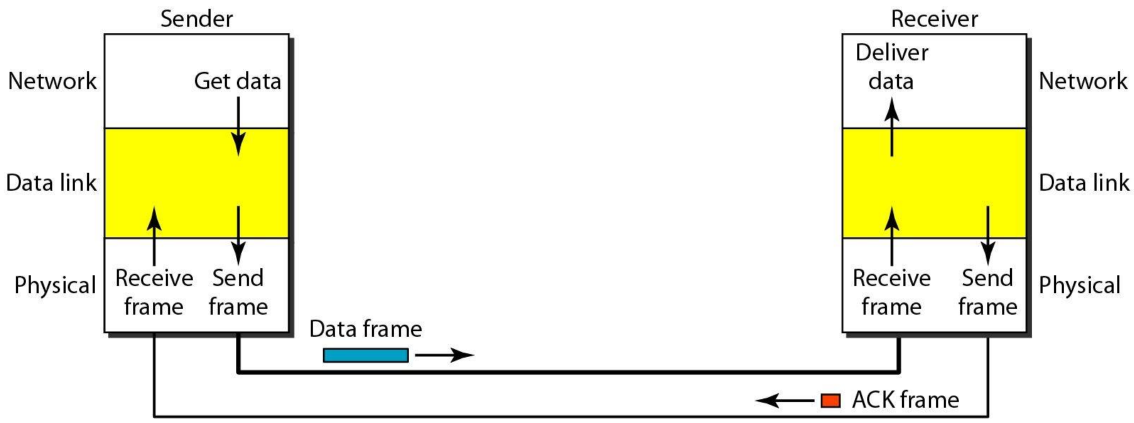
2. Design Diagram

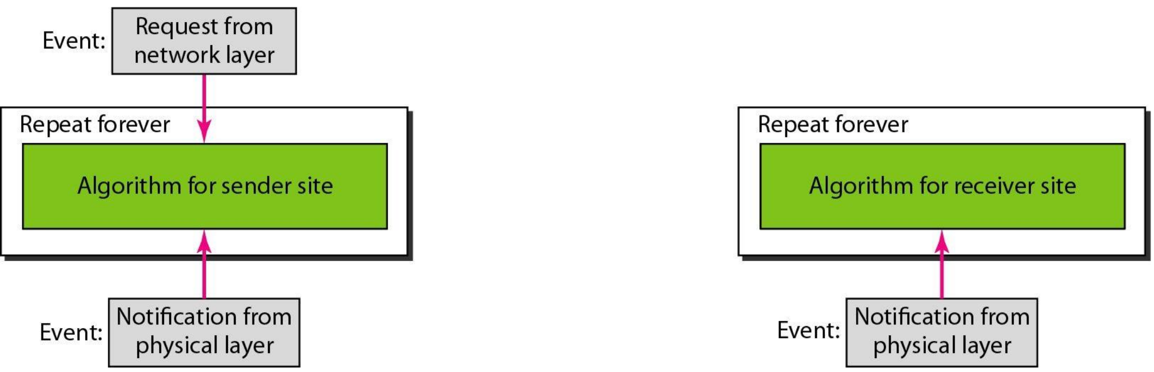
7. Sender's Side

1. **Rule 1:** The sender sends one data packet at a time.
2. **Rule 2:** The sender only sends the subsequent packet after getting the preceding packet's acknowledgement.
3. Therefore, the concept behind the stop and wait protocol on the sender's end is relatively straightforward: Send one packet at a time and refrain from sending any additional packets until you have received an acknowledgement.

8. Receiver's Side

1. **Rule 1:** Receive the data packet, then consume it.
2. **Rule 2:** The receiver provides the sender with an acknowledgement after consuming the data packet.
3. As a result, the stop and wait protocol's basic tenet on the receiver's end is similarly extremely straightforward: Ingest the packet, and after it has been consumed, send the acknowledgement. This is a mechanism for flow control

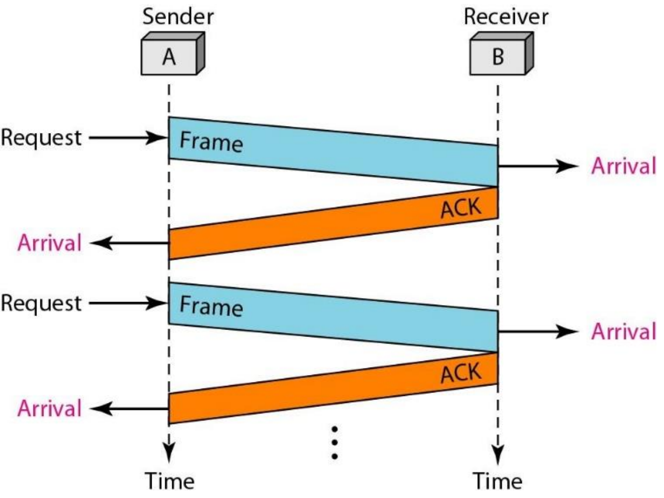




3. *Flow Diagram*

9. Working of Stop and Wait Protocol

1. The sender will not send the second packet to the receiver until acknowledgment of the first packet is received.
2. The receiver will send the acknowledgment for the packet that the receiver has received.
3. When the sender receives the acknowledgment, it sends the next packet to the receiver.
4. This process of sending data and receiving acknowledgment continues until all the packets are not sent.



4. *PROTOCOL*

Algorithm 11.3 Sender-site algorithm for Stop-and-Wait Protocol

```
while(true)                          //Repeat forever
canSend = true                       //Allow the first frame to go
{
  WaitForEvent();                    // Sleep until an event occurs
  if(Event(RequestToSend) AND canSend)
  {
     GetData();
     MakeFrame();
     SendFrame();                    //Send the data frame
     canSend = false;                //Cannot send until ACK arrives
  }
  WaitForEvent();                    // Sleep until an event occurs
  if(Event(ArrivalNotification)  // An ACK has arrived
   {
     ReceiveFrame();                 //Receive the ACK frame
     canSend = true;
   }
}
```

Algorithm 11.4 Receiver-site algorithm for Stop-and-Wait Protoco

```
while(true)                          //Repeat forever
{
  WaitForEvent();                    // Sleep until an event occurs
  if(Event(ArrivalNotification)) //Data frame arrives
  {
     ReceiveFrame();
     ExtractData();
     Deliver(data);                  //Deliver data to network layer
     SendFrame();                    //Send an ACK frame
  }
}
```

III. STOP AND wait PROTOCOL ARQ

1. Introduction

1. The stop and wait ARQ is a connection-oriented protocol.
2. It uses sequence numbers to number the frames the sequence numbers are based on modulo-2 arithmetic.
3. The acknowledgment number always announces in modulo-2 arithmetic the sequence number of the next frame expected.
4. the stop and wait ARQ, the sender needs to maintain a time tracker.
5. The stop and wait ARQ is a sliding window protocol with a window size equal to 1.
6. Error correction in Stop-and-Wait ARQ is done by keeping a copy of the sent frame and retransmitting of the frame when the timer expires.
7. The sender can send only one frame at a time and the receiver can also receive only one frame at a time.

2. SEQUENCE NUMBER

10. DATA FRAME

Sequence Numbers

- Sequence Numbers: 0,1

Data frame

- 0 or 1

S_n

0	1	0	1	0	1

11. ACK FRAME

Acknowledgement frame

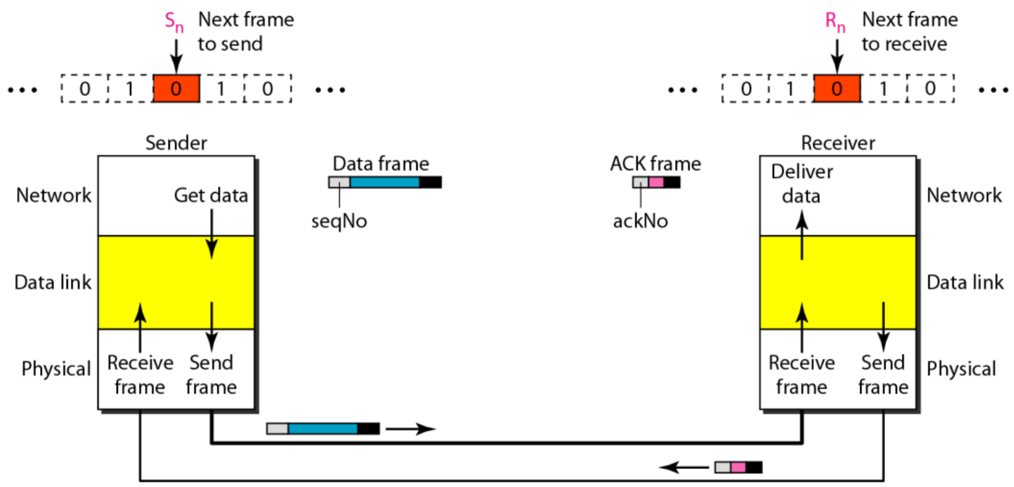
- 0 or 1
- Send Acknowledgement for expected frame
- Received : Frame 0 , ACK : Frame 1

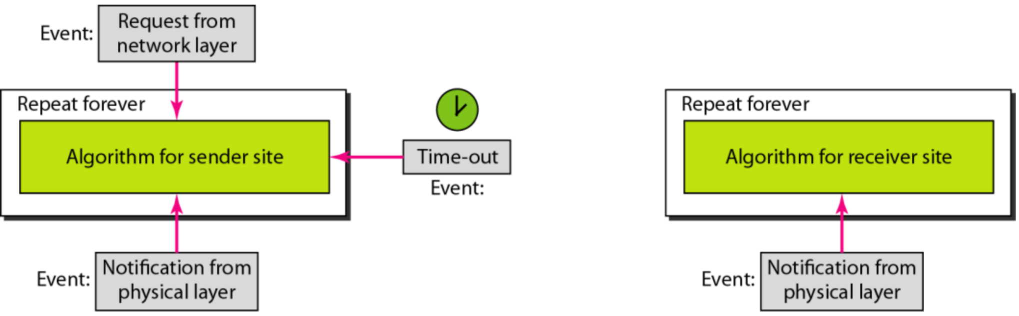
R_n

0	1	0	1	0	1

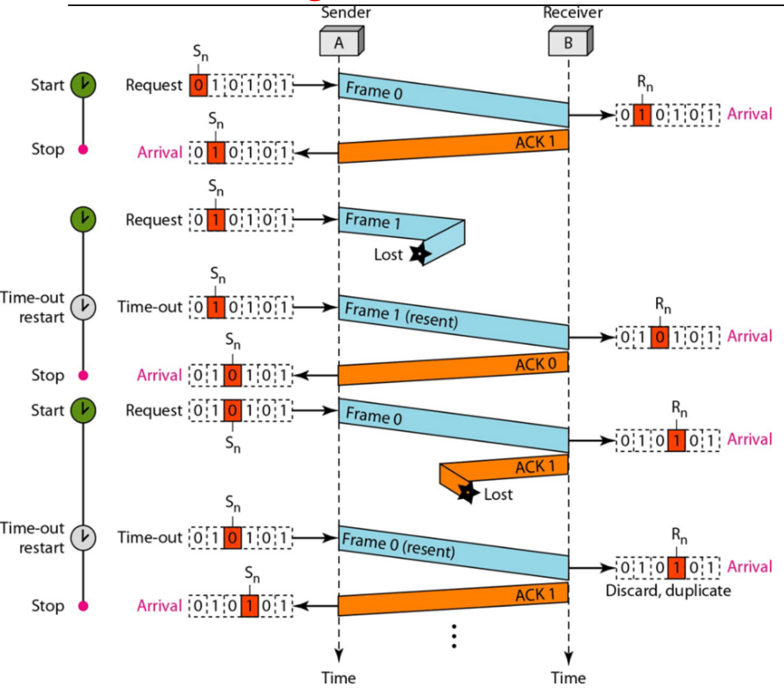
3. Design Diagram

1. The sending device keeps a copy of the last frame transmitted until it receives an acknowledgment for that frame.
2. A data frames uses a seqNo (sequence number); an ACK frame uses an ackNo (acknowledgment number).
3. The sender has a control variable, which we call Sn (sender, next frame to send), that holds the sequence number for the next frame to be sent (0 or 1).
4. The receiver has a control variable, which we call Rn (receiver, next frame expected), that holds the number of the next frame expected.
5. When a frame is sent, the value of Sn is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa.
6. When a frame is received, the value of Rn is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa.
7. Three events can happen at the sender site; one event can happen at the receiver site





4. Flow Diagram



Assume we have used x as a sequence number;

1. The frame arrives safe and sound at the receiver site; the receiver sends an acknowledgment. The acknowledgment arrives at the sender site, causing the sender to send the next frame numbered x + 1.
2. The frame arrives safe and sound at the receiver site; the receiver sends an acknowledgment, but the acknowledgment is corrupted or lost.
   The sender resends the frame (numbered x) after the time-out. Note that the frame here is a duplicate. The receiver can recognize this fact because it expects frame x + I but frame x was received.
3. The frame is corrupted or never arrives at the receiver site; the sender resends the frame (numbered x) after the time-out.

5. PROTOCOL

Sender-site algorithm

```
Sn = 0;                          // Frame 0 should be sent first
canSend = true;                  // Allow the first request to go
while(true)                      // Repeat forever
{
  WaitForEvent();                // Sleep until an event occurs
  if(Event(RequestToSend) AND canSend)
  {
     GetData();
     MakeFrame(Sn);                       //The seqNo is Sn
     StoreFrame(Sn);                      //Keep copy
     SendFrame(Sn);
     StartTimer();
     Sn = Sn + 1;
     canSend = false;
  }
  WaitForEvent();                         // Sleep
    if(Event(ArrivalNotification)       // An ACK has arrived
    {
      ReceiveFrame(ackNo);              //Receive the ACK frame
      if(not corrupted AND ackNo == Sn) //Valid ACK
        {
          Stoptimer();
          PurgeFrame(Sn-1);             //Copy is not needed
          canSend = true;
        }
     }

    if(Event(TimeOut)                   // The timer expired
    {
     StartTimer();
     ResendFrame(Sn-1);                 //Resend a copy check
    }
}
```

Receiver-site algorithm

```
Rn = 0;                      // Frame 0 expected to arrive first
while(true)
{
  WaitForEvent();            // Sleep until an event occurs
  if(Event(ArrivalNotification))  //Data frame arrives
  {
     ReceiveFrame();
     if(corrupted(frame));
        sleep();
     if(seqNo == Rn)              //Valid data frame
     {
      ExtractData();
       DeliverData();             //Deliver data
       Rn = Rn + 1;
     }
      SendFrame(Rn);              //Send an ACK
  }
}
```

IV. GO-BACK-N ARQ

1. Introduction

1. Go-Back-N Automatic Repeat Request protocol for Noisey channels in the data link layer.
2. Go Back N ARQ which stands for Go Back N Automatic Repeat Request (ARQ) is a data link layer protocol that is used for data flow control purposes.
3. It is a sliding window protocol in which multiple frames are sent from sender to receiver at once.
4. The number of frames that are sent at one depends upon the size of the window that is taken.
5. The size of the sender window in Go Back N ARQ is equal to N.
6. The size of the receiver window in Go Back N ARQ is equal to 1.
7. When the acknowledgement for one frame is not received by the sender, the entire window is retransmitted, starting with the corrupted frame
8. The copy of sent data is maintained in the sent buffer of the sender until all the sent packets are acknowledged.
9. If the timeout timer runs out then the sender will resend all the packets.
10. Go-Back-N ARQ is a more efficient use of a connection than Stop-and-wait ARQ, since unlike waiting for an acknowledgement for each packet, the connection is still being utilized as packets are being sent. In other words, during the time that would otherwise be spent waiting, more packets are being sent.

2. SEQUENCE NUMBERS

- m bits
- Range: 0 to 2^m -1
- m = 2
- 0 to 2^2 -1 (0 to 3)
- 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3

- m = 4
- Range :0 to 2^4 -1 (0 to 15)
- 0, 1, 2.....15 , 0, 1, 2.....15 , 0, 1, 2.....15

m =4

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

3. SLIDING WINDOW

4. SEND SW

m =4,size =15

Range of sequence numbers=0 to 2^m -1

= 0 to 15(m=4)

4 regions

Region1 :

- Acknowledged

Region 2:

- Sent, Not Acknowledged
- Outstanding

Region 3:

- Can be sent
- Not received

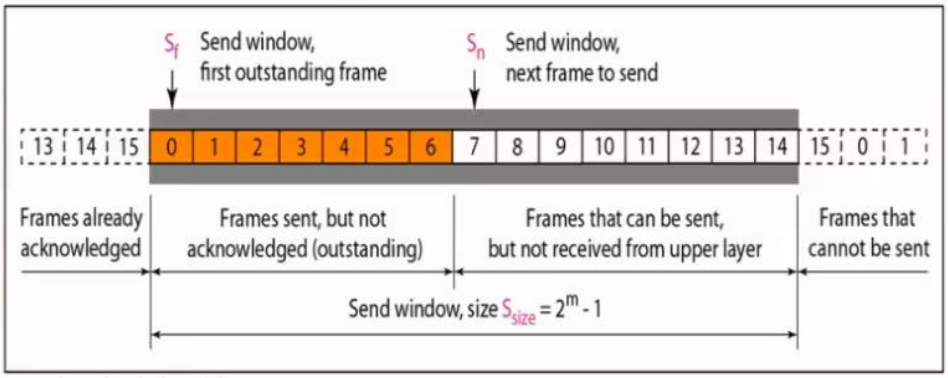
Region 4:

- Cannot be sent

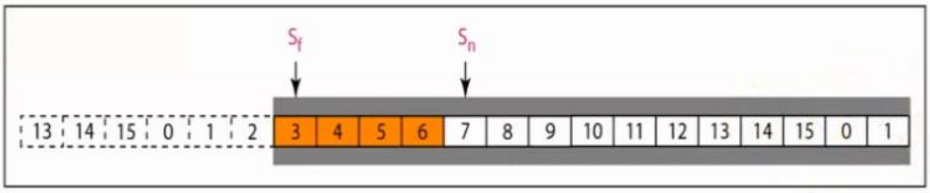
S_f : First outstanding frame

S_n: Sequence number assigned to next frame

Send window before sliding



Send window after sliding



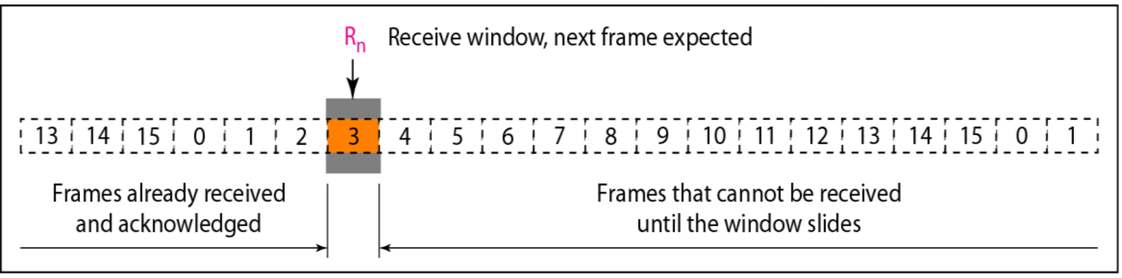
Window slides : After Acknowledgement is received

Frames 0,1, 2 : Acknowledged

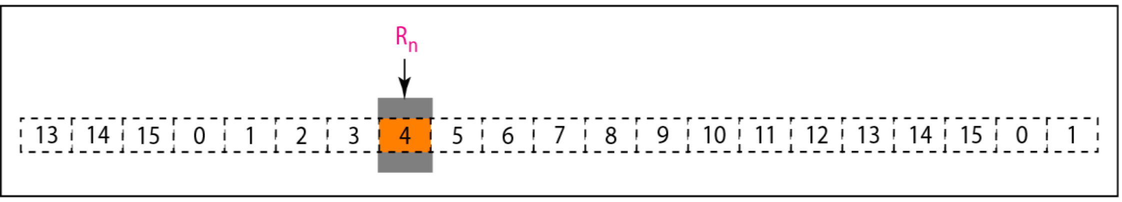
Cumulative Acknowledgement, Individual Acknowledgment

5. RECEIVE SW

- Window: Expected frame sequence number
- Left (window) : Received and Acknowledged
- Right (window) : Cannot be received
- Seq no == R_n : Accepted and Acknowledged

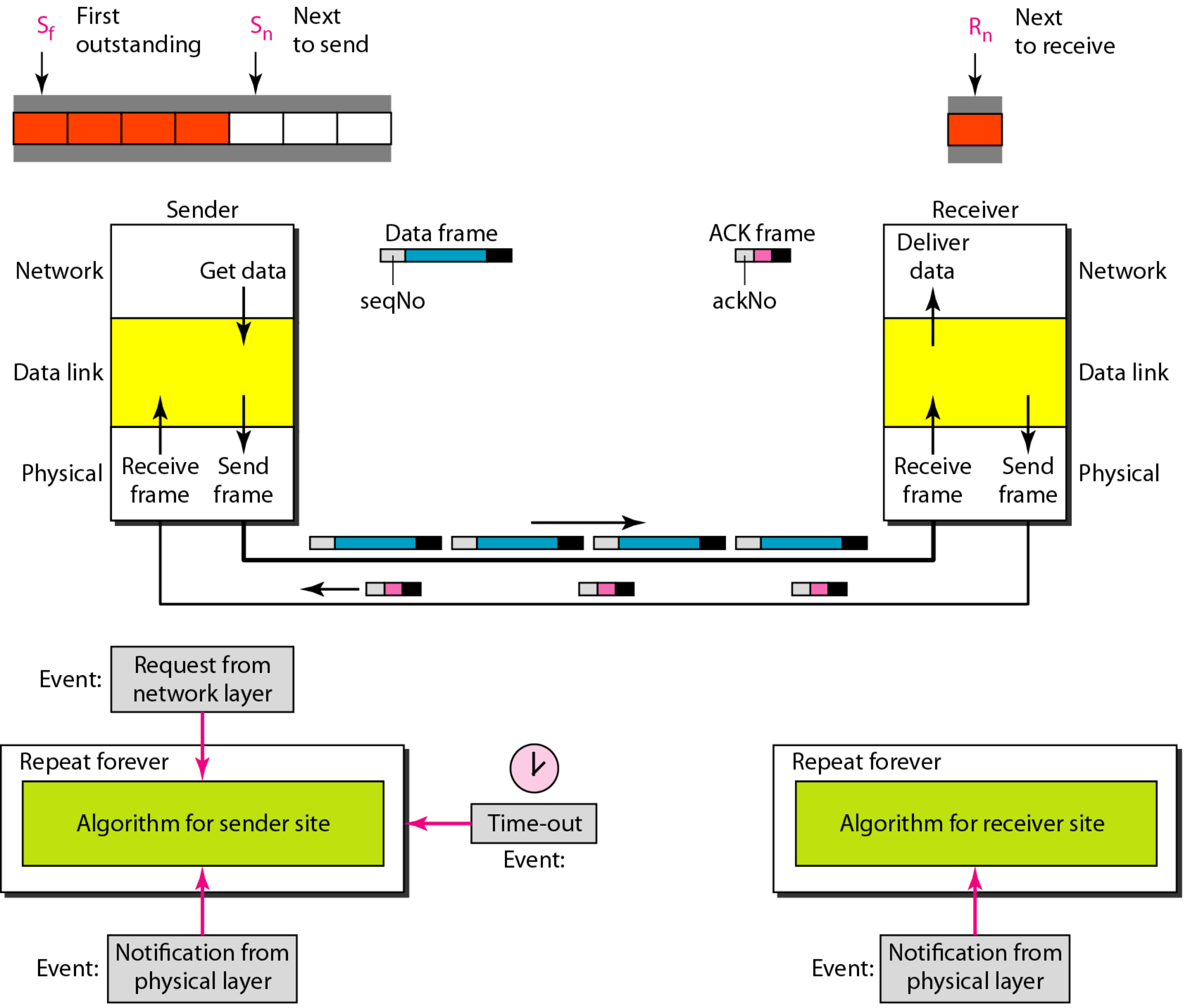


a. Receive window

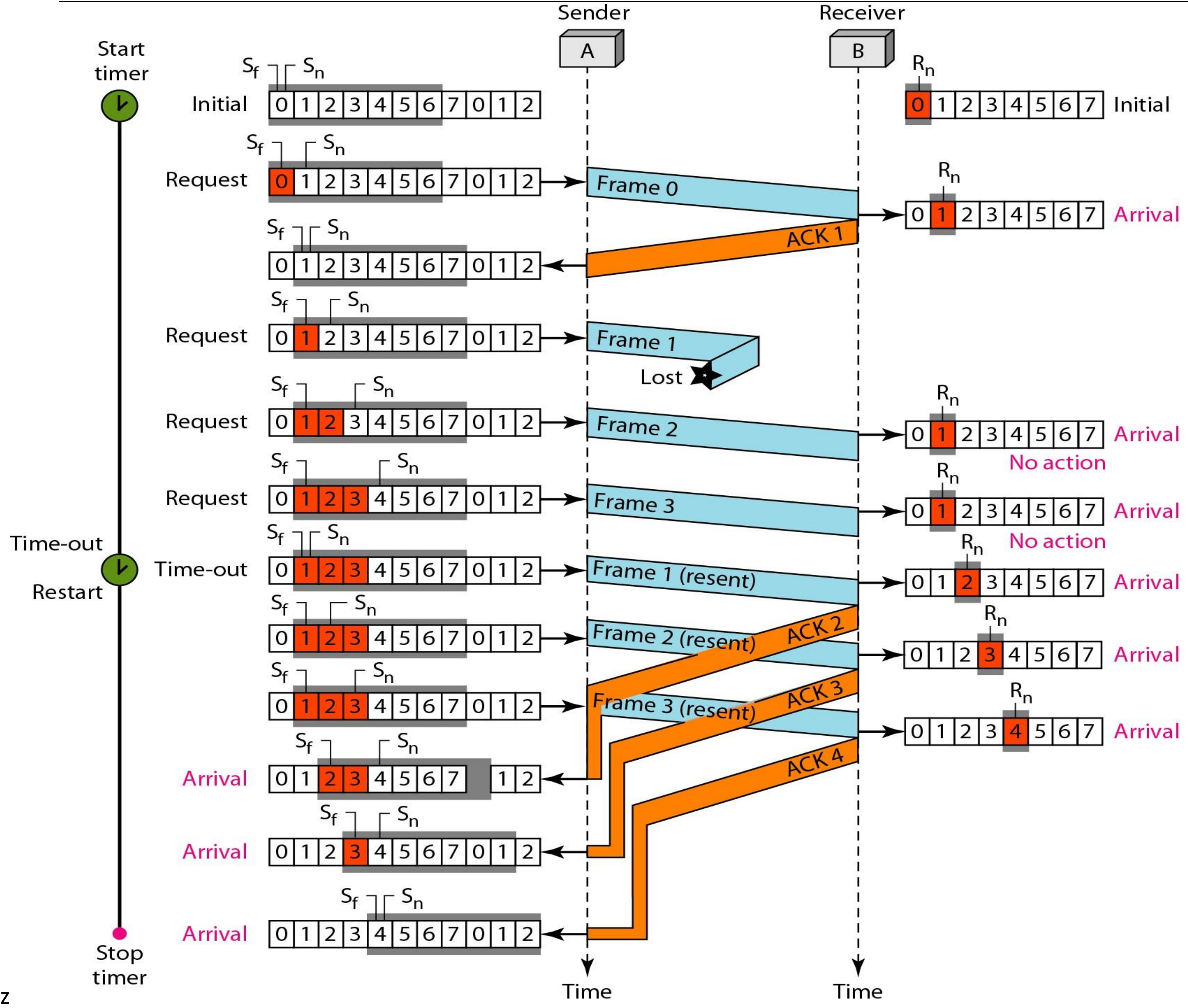


b. Window after sliding

6. Design Diagram



1. Flow Diagram



z

z

Go-Back-N sender

```
Sw = 2^m - 1;
Sf = 0;
Sn = 0;

while (true)                            //Repeat forever
{
 WaitForEvent();
  if(Event(RequestToSend))              //A packet to send
  {
     if(Sn-Sf >= Sw)                    //If window is full
           Sleep();
     GetData();
     MakeFrame(Sn);
     StoreFrame(Sn);
     SendFrame(Sn);
     Sn = Sn + 1;
     if(timer not running)
          StartTimer();
  }

  if(Event(ArrivalNotification))        //ACK arrives
  {
     Receive(ACK);
     if(corrupted(ACK))
          Sleep();
     if((ackNo>Sf)&&(ackNo<=Sn))        //If a valid ACK
     While(Sf <= ackNo)
      {
       PurgeFrame(Sf);
       Sf = Sf + 1;
      }
      StopTimer();
  }

  if(Event(TimeOut))                    //The timer expires
  {
   StartTimer();
   Temp = Sf;
   while(Temp < Sn);
    {
     SendFrame(Sf);
     Sf = Sf + 1;
    }
  }
}
```

Go-Back-N receiver algorithm

```
Rn = 0;

while (true)                            //Repeat forever
{
  WaitForEvent();

  if(Event(ArrivalNotification)) /Data frame arrives
  {
     Receive(Frame);
     if(corrupted(Frame))
          Sleep();
     if(seqNo == Rn)                    //If expected frame
     {
       DeliverData();                   //Deliver data
       Rn = Rn + 1;                     //Slide window
       SendACK(Rn);
     }
  }
}
```

V. SELECTIVE REPEAT ARQ

1. *Introduction*

2. The selective repeat ARQ is one of the Sliding Window Protocol strategies that is used where reliable in-order delivery of the data packets is required.
3. ARQ (Automatic Repeat Request) is an error-control strategy that ensures that a sequence of information is delivered in order and without any error or duplications despite transmission errors and losses.
4. The selective repeat ARQ is used for noisy channels or links and it manages the flow and error control between the sender and the receiver.
5. In the selective repeat ARQ, we only resend the data frames that are damaged or lost.
6. If any frame is lost or damaged then the receiver sends a negative acknowledgment (NACK) to the sender and if the frame is correctly received, it sends back an acknowledgment (ACK).
7. The sender sets a timer for each frame so whenever the timer is over and the sender has not received any acknowledgment, then the sender knows that the particular frame is either lost or damaged.
8. As the sender needs to wait for the timer to expire before retransmission. So, we use negative acknowledgment or NACK.
9. The ACK and the NACK have the sequence number of the frame that helps the sender to identify the lost frame.
10. The receiver has the capability of sorting the frames present in the memory buffer using the sequence numbers.
11. The sender must be capable enough to search for the lost frame for which the NACK has been received.

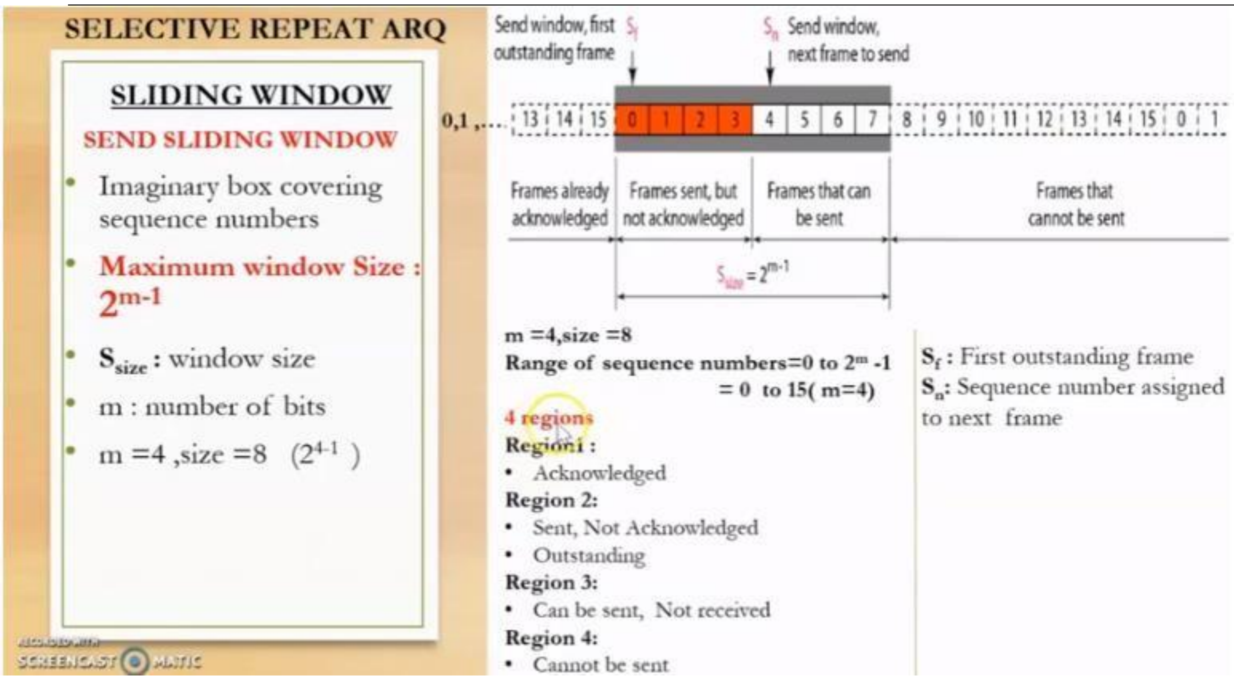
12. *Windows*

The Selective Repeat Protocol also uses two windows: a send window a receive window

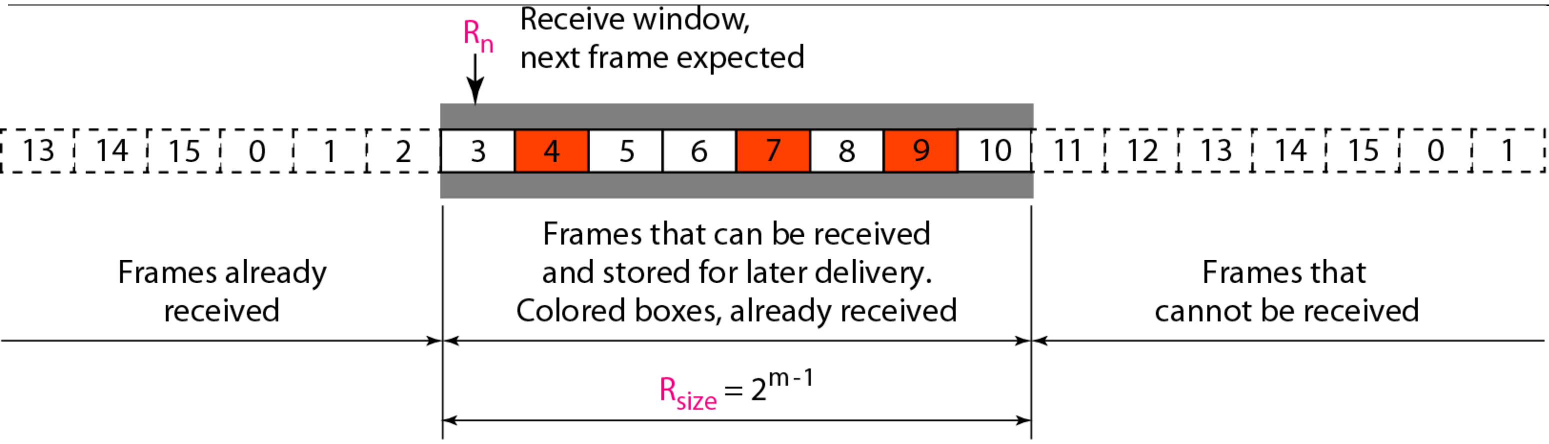
The send window maximum size can be   For example, if m = 4, the sequence numbers go from 0 to 15, but the size of the window is just 8

The receive window in Selective Repeat is totally different from the one in GoBack-N. First, the size of the receive window is the same as the size of the send window

1. *Send Window*

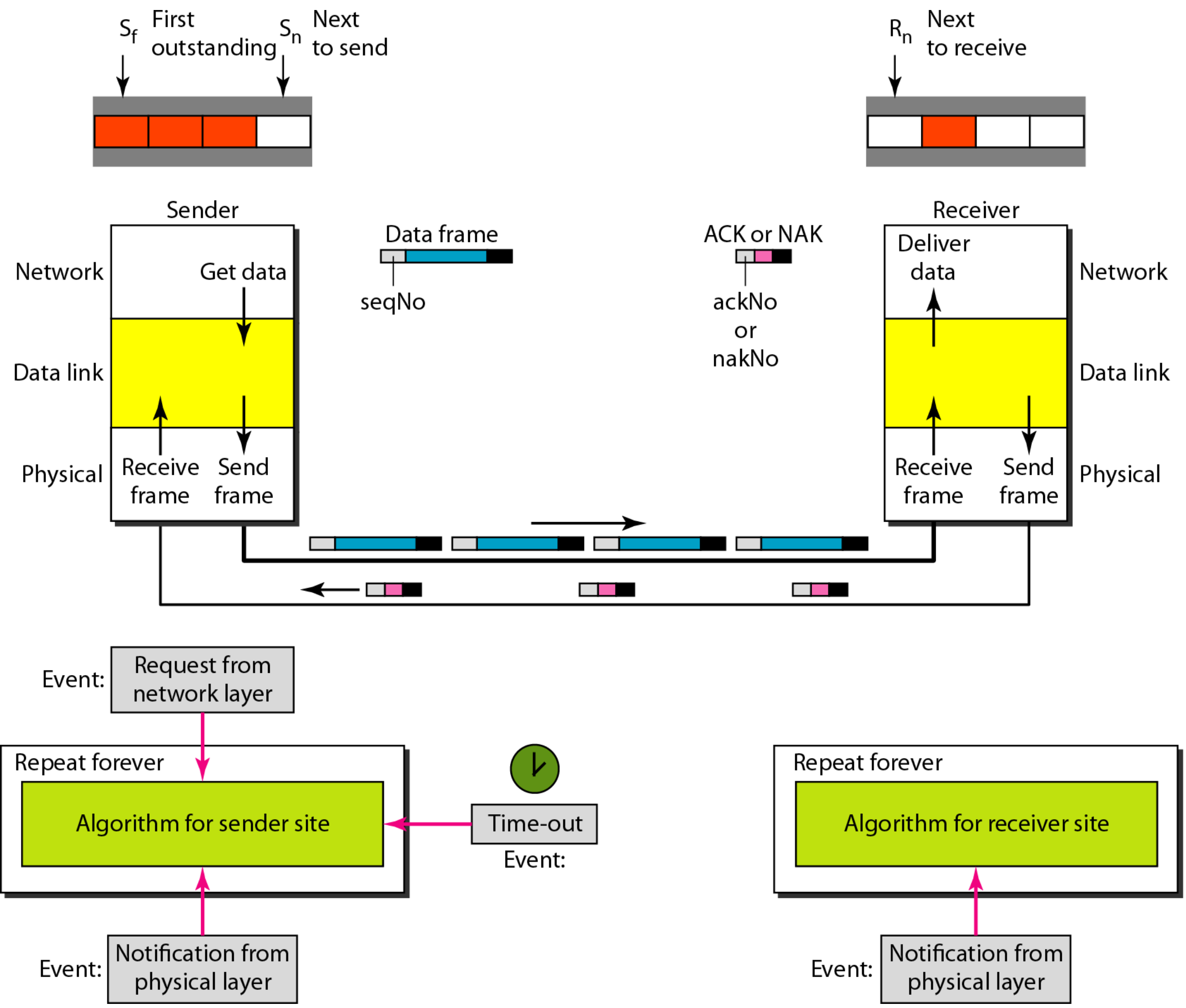


2. Receive Window



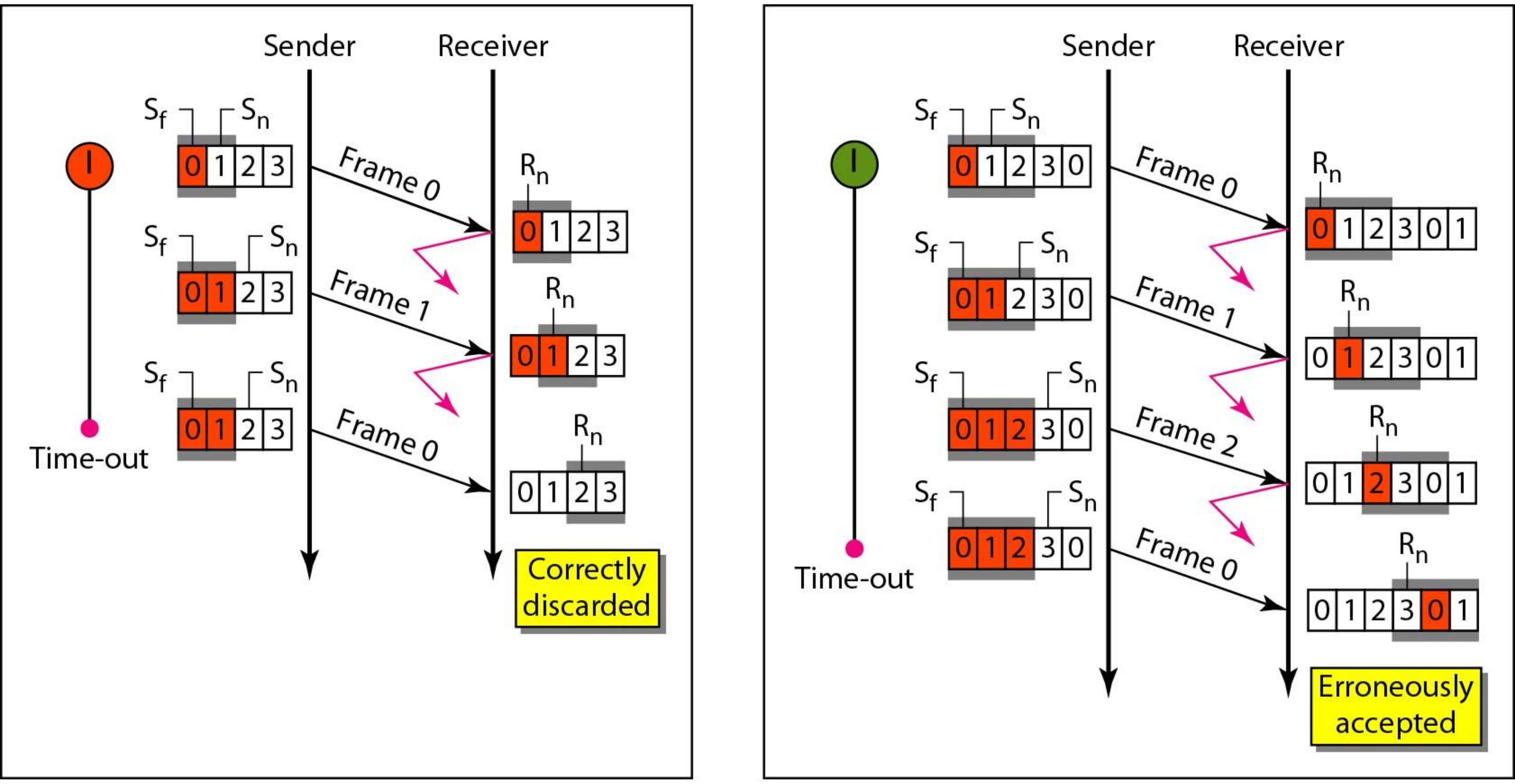
3. Design Diagram

z



4. *Flow Diagram*

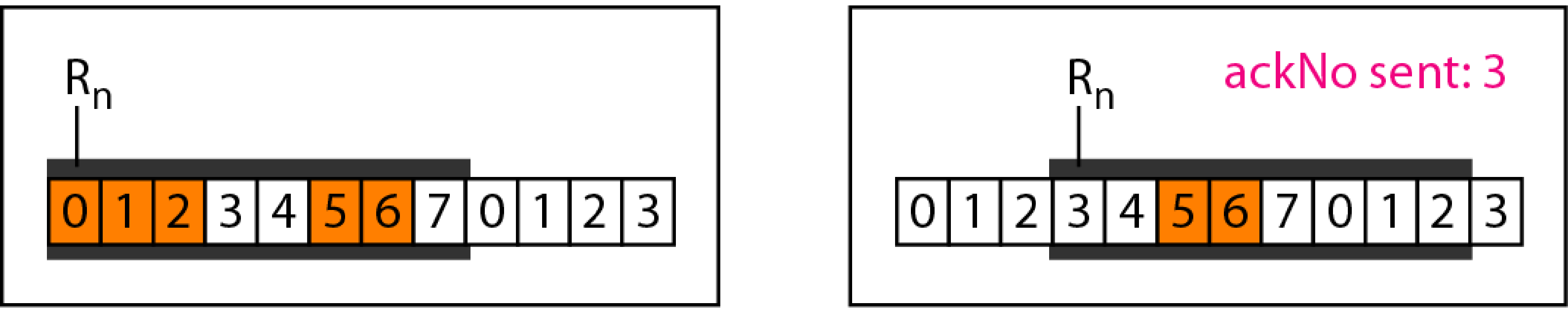
z



a. Window size = 2^{m-1} b. Window size > 2^{m-1}

The steps of data transmission can be:

1. The sender sends frames 0 and 1.
2. The receiver receives the frames and sends back ACK 0 and ACK 1.
3. Again the sender sends the frames 2 and 3.
4. The receiver only receives the frame 3. So it sends back NACK 2 which means that the 2nd frame is lost and needs to be re-transmitted.
5. So, the sender sends back the frame 2 and this process is continued till all the frames have been received by the receiver.



a. Before delivery   b. After delivery

5. PROTOCOL

```
      if(Sn-Sf >= Sw)                 //If window is full
            Sleep();
      GetData();
      MakeFrame(Sn);
      StoreFrame(Sn);
      SendFrame(Sn);
      Sn = Sn + 1;
      StartTimer(Sn);
   }

   if(Event(ArrivalNotification)) //ACK arrives
   {
      Receive(frame);                 //Receive ACK or NAK
      if(corrupted(frame))
           Sleep();
      if (FrameType == NAK)
         if (nakNo between Sf and Sn)
         {
          resend(nakNo);
          StartTimer(nakNo);
         }
      if (FrameType == ACK)
         if (ackNo between Sf and Sn)
         {
           while(sf < ackNo)
           {
            Purge(sf);
            StopTimer(sf);
            Sf = Sf + 1;
           }
         }
   }

   if(Event(TimeOut(t)))              //The timer expires
   {
    StartTimer(t);
    SendFrame(t);
   }
}
```

Receiver-site Selective Repeat algorithm

```
Rn = 0;
NakSent = false;
AckNeeded = false;
Repeat(for all slots)
    Marked(slot) = false;

while (true)                                    //Repeat forever
{
  WaitForEvent();

  if(Event(ArrivalNotification))                /Data frame arrives
  {
     Receive(Frame);
     if(corrupted(Frame))&& (NOT NakSent)
     {
      SendNAK(Rn);
      NakSent = true;
      Sleep();
     }
     if(seqNo <> Rn)&& (NOT NakSent)
     {
      SendNAK(Rn);
```

```
      NakSent = 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;
        NakSent = false;
        }
      }
     }
   }
}
```