CSE / T / 315A
Data Communications
Topic 8- Data Link Control

Sarbani Roy

sarbani.roy@jadavpuruniversity.in

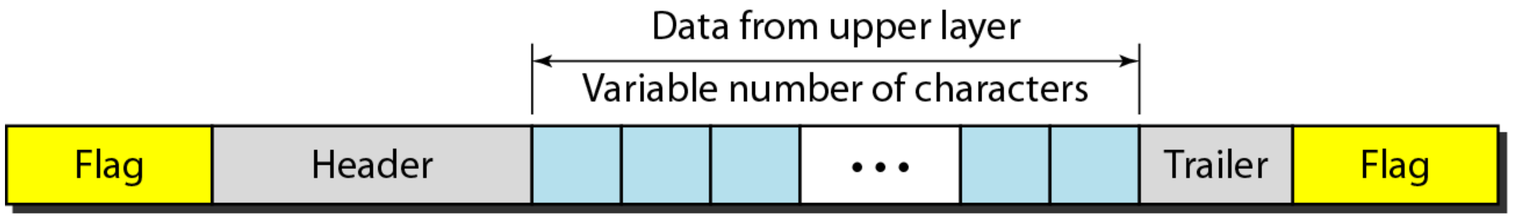
Office: CC-5-7

Cell: 9051639328

Framing

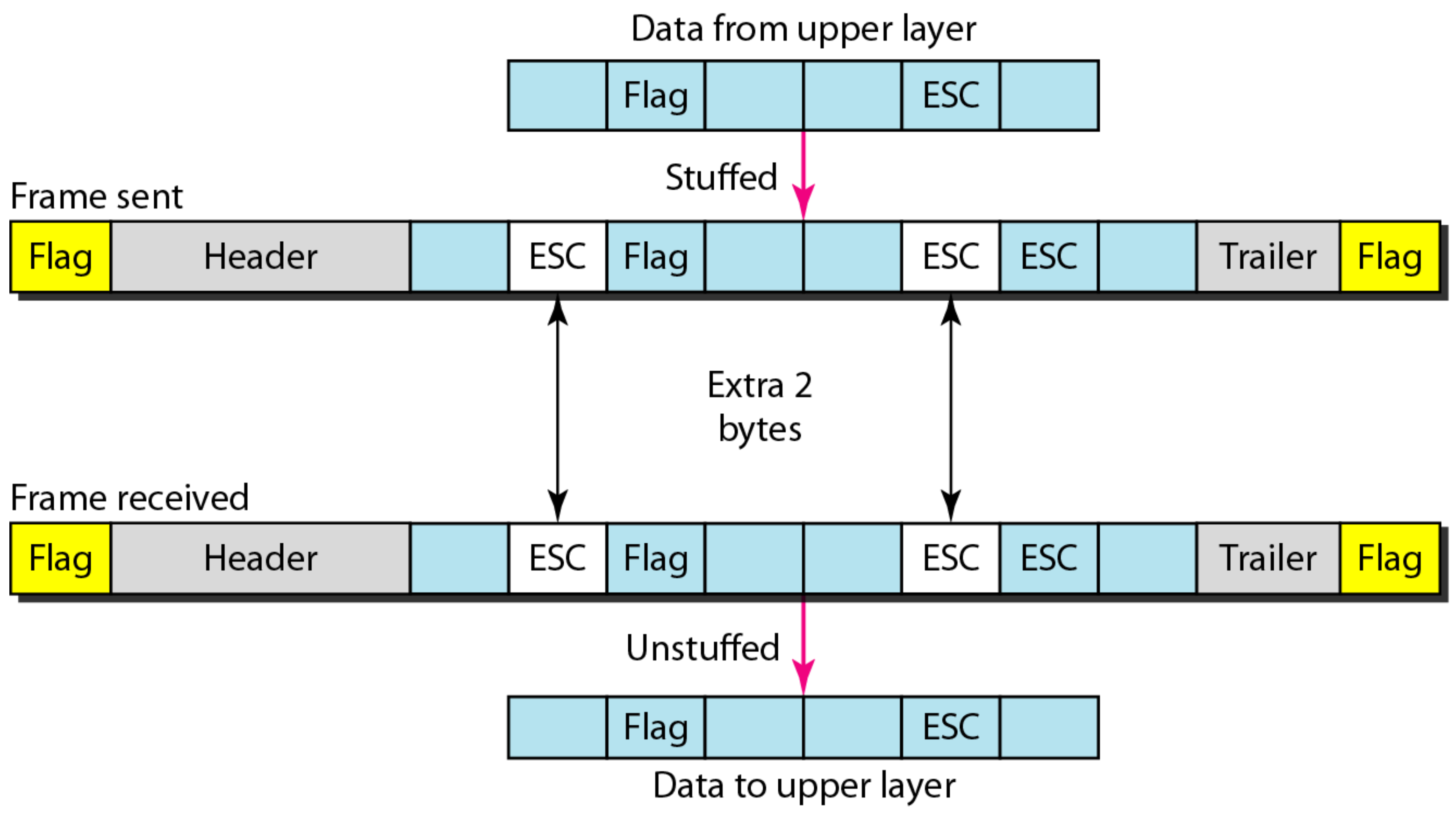
- The data link layer needs to pack bits into frames, so that each frame is distinguishable from another. Our postal system practices a type of framing. The simple act of inserting a letter into an envelope separates one piece of information from another; the envelope serves as the delimiter.
  - Fixed size framing
  - Variable size framing

A frame in a character-oriented protocol

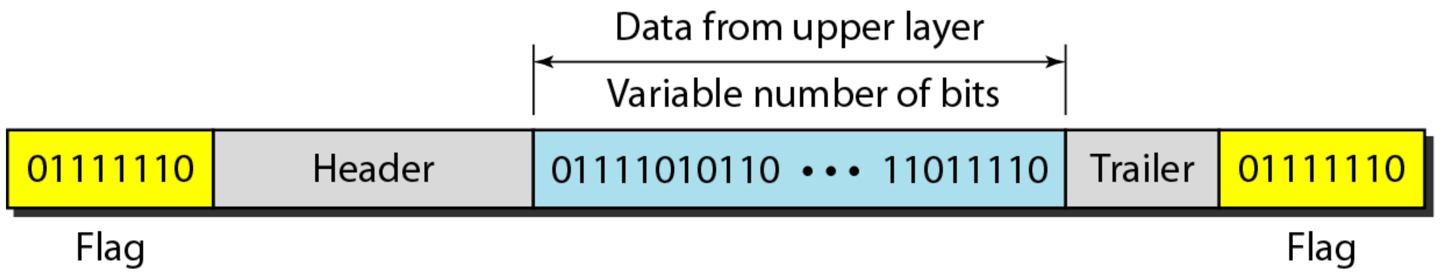


Byte stuffing and unstuffing

- Byte stuffing is the process of adding 1 extra byte whenever there is a flag or escape character in the text.

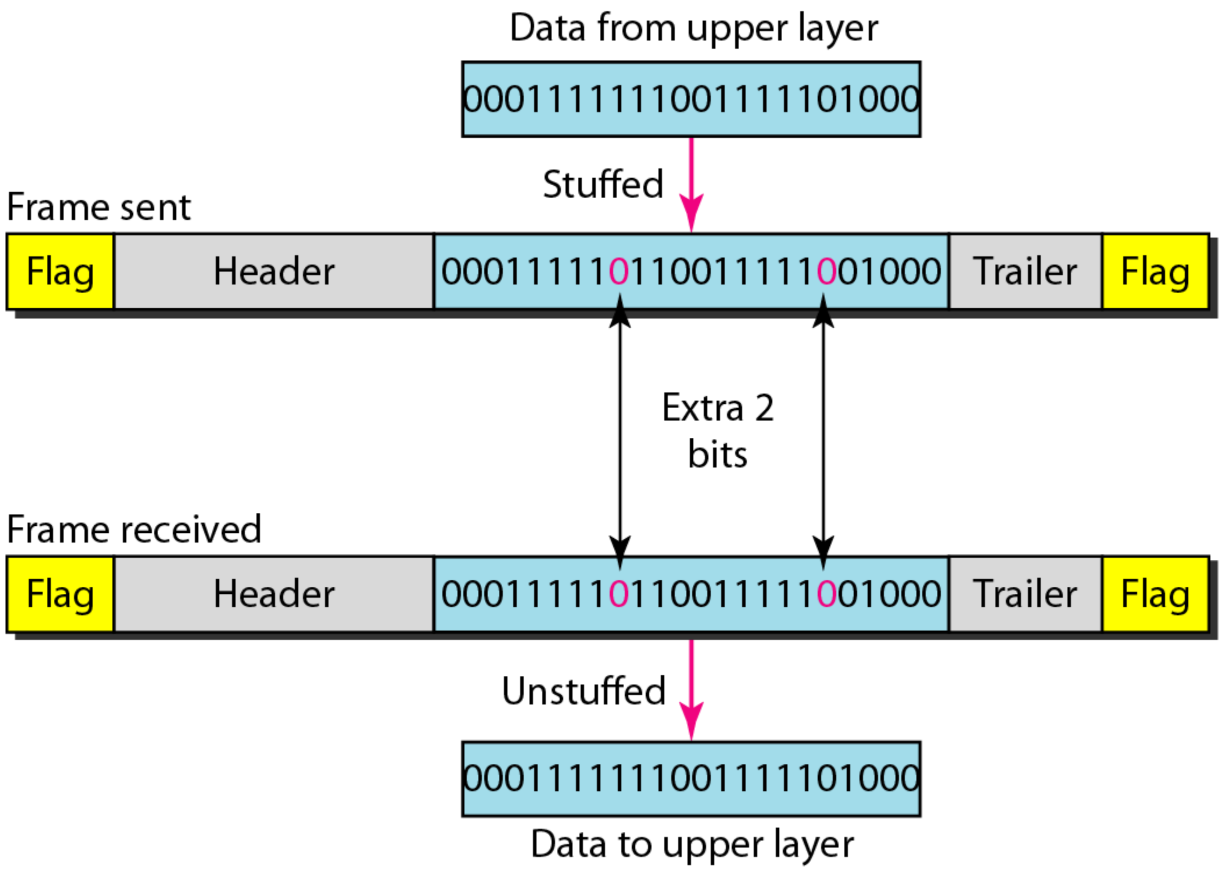


A frame in a bit-oriented protocol



Bit stuffing and unstuffing

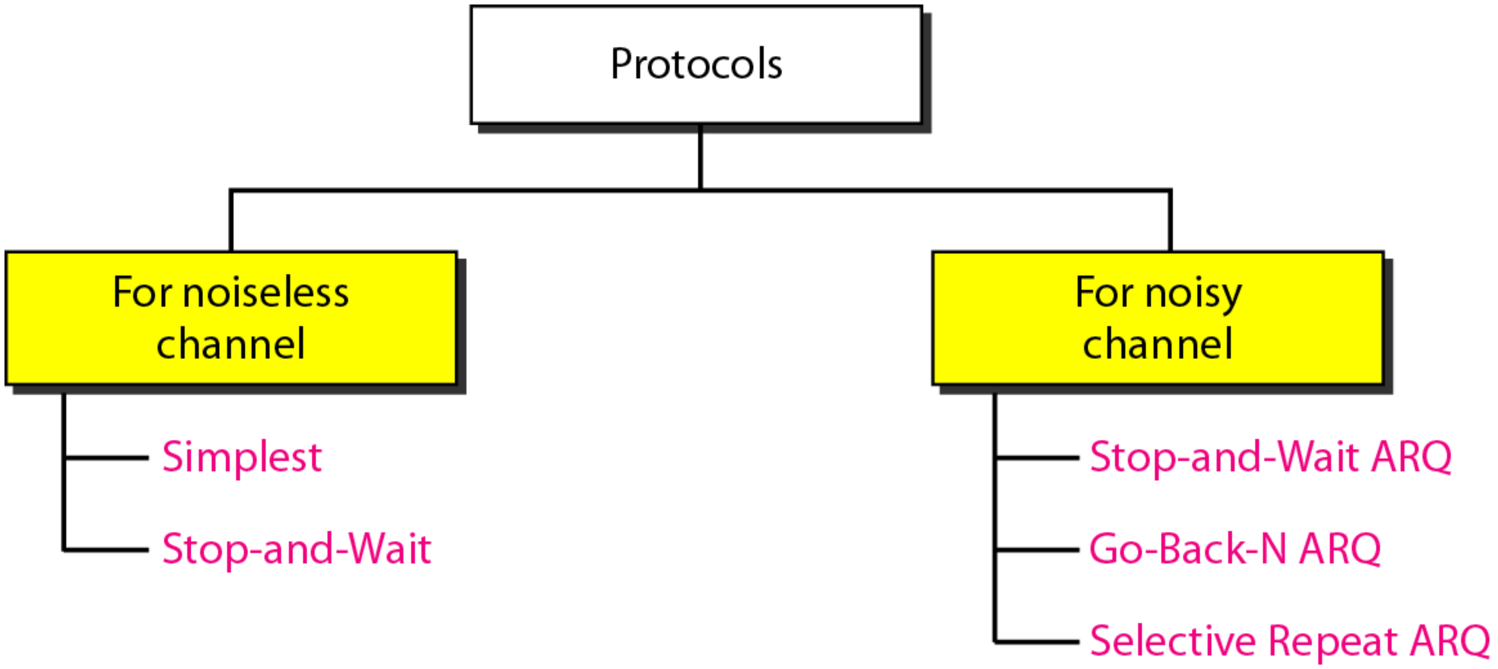
- Bit stuffing is the process of adding one extra 0 whenever five consecutive 1s follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag.



Flow and Error Control

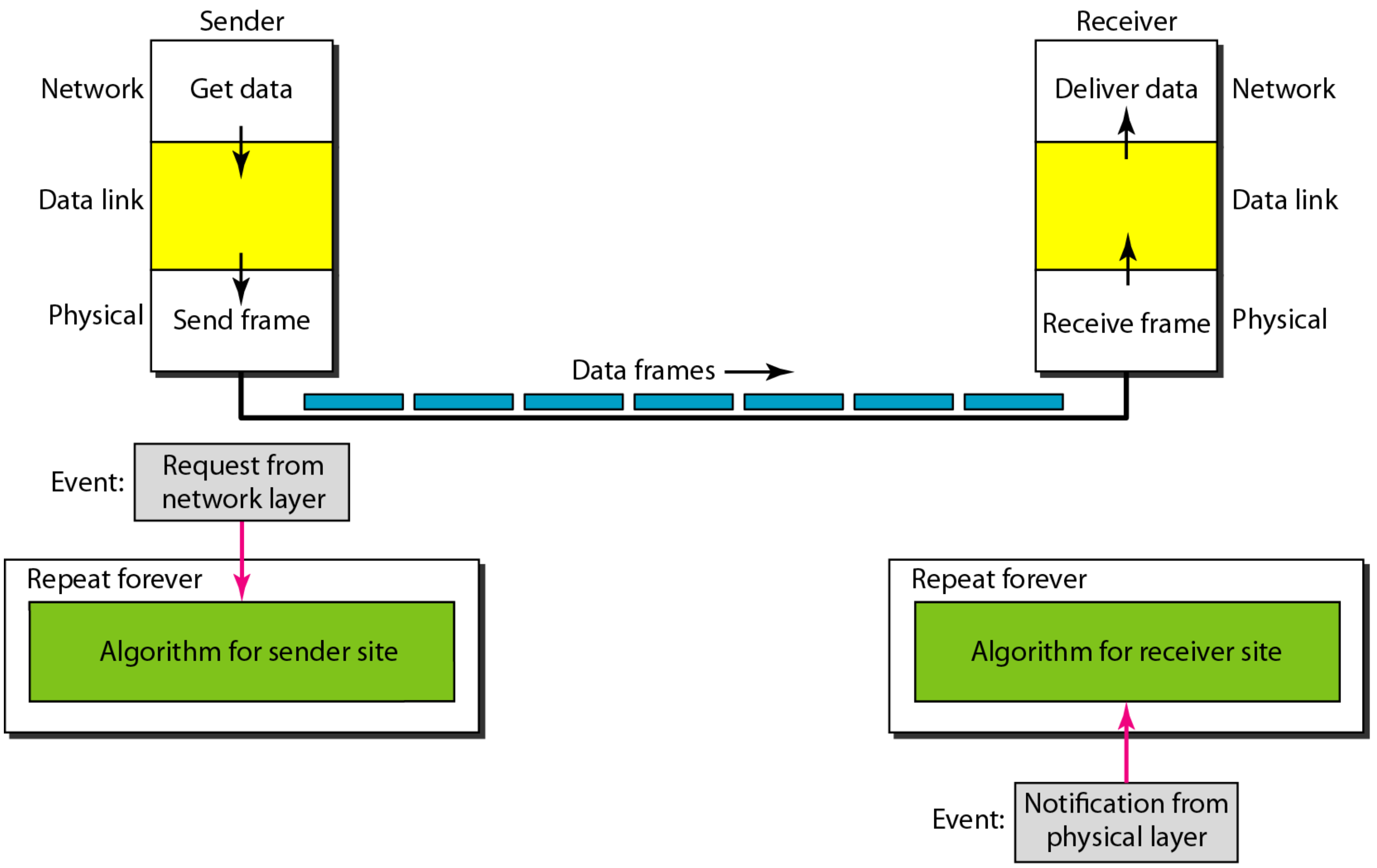
- The most important responsibilities of the data link layer are flow control and error control. Collectively, these functions are known as data link control.
- Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment.
- Error control in the data link layer is based on automatic repeat request, which is the retransmission of data.

Taxonomy of protocols



- Noiseless channel: an ideal channel in which no frames are lost, duplicated, or corrupted
- Noiseless channels are nonexistent

The design of the simplest protocol with no flow or error control



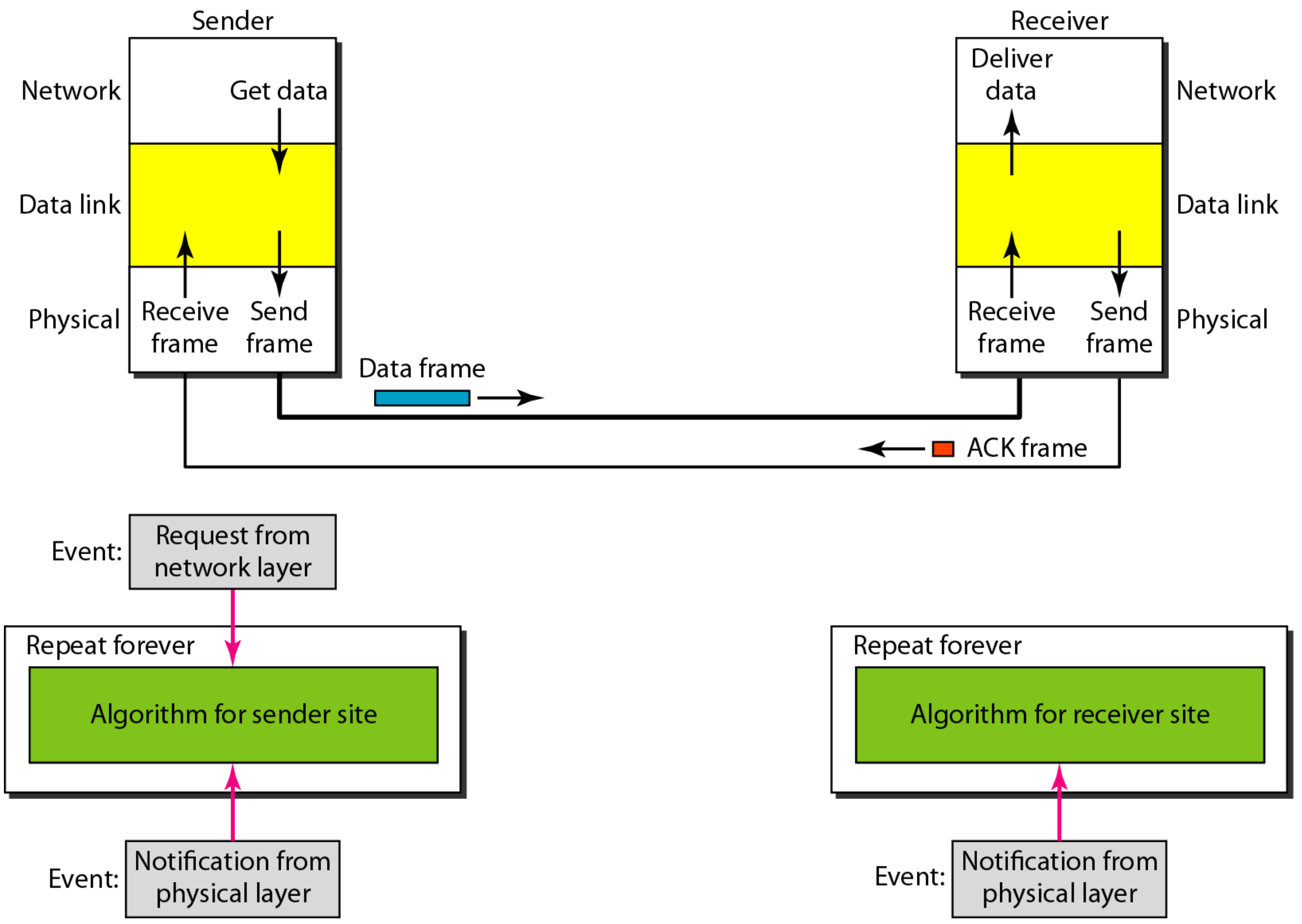
Sender-site algorithm for the simplest protocol

```
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
   }
}
```

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
  }
}
```

Design of Stop-and-Wait Protocol



Sender-site algorithm for Stop-and-Wait Protocol

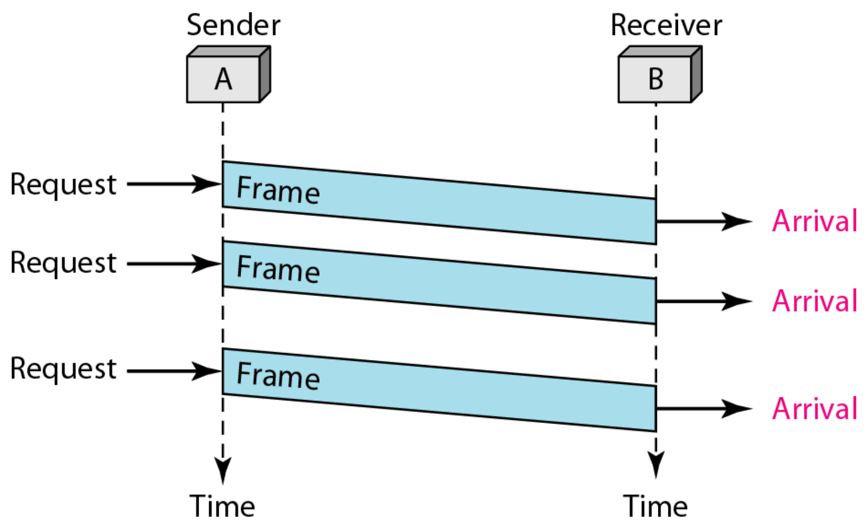
```
while(true)                          //Repeat forever
canSend = true                       //Allow the first frame to go
{
  WaitForEvent();                    // Sleep until an event occurs
  if(Event(RequestToSend) AND canSend)
  {
     GetData();
     MakeFrame();
     SendFrame();                    //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;
   }
}
```

Receiver-site algorithm for Stop-and-Wait Protocol

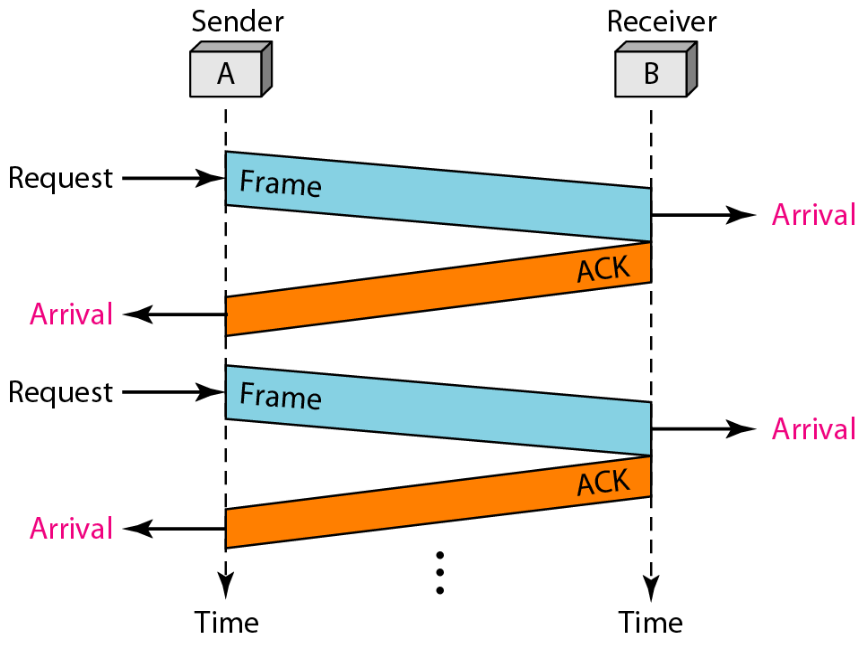
```
while(true)                             //Repeat forever
{
  WaitForEvent();                       // Sleep until an event occurs
  if(Event(ArrivalNotification))        //Data frame arrives
  {
     ReceiveFrame();
     ExtractData();
     Deliver(data);                     //Deliver data to network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

Flow Diagram

Simplest



Stop and Wait

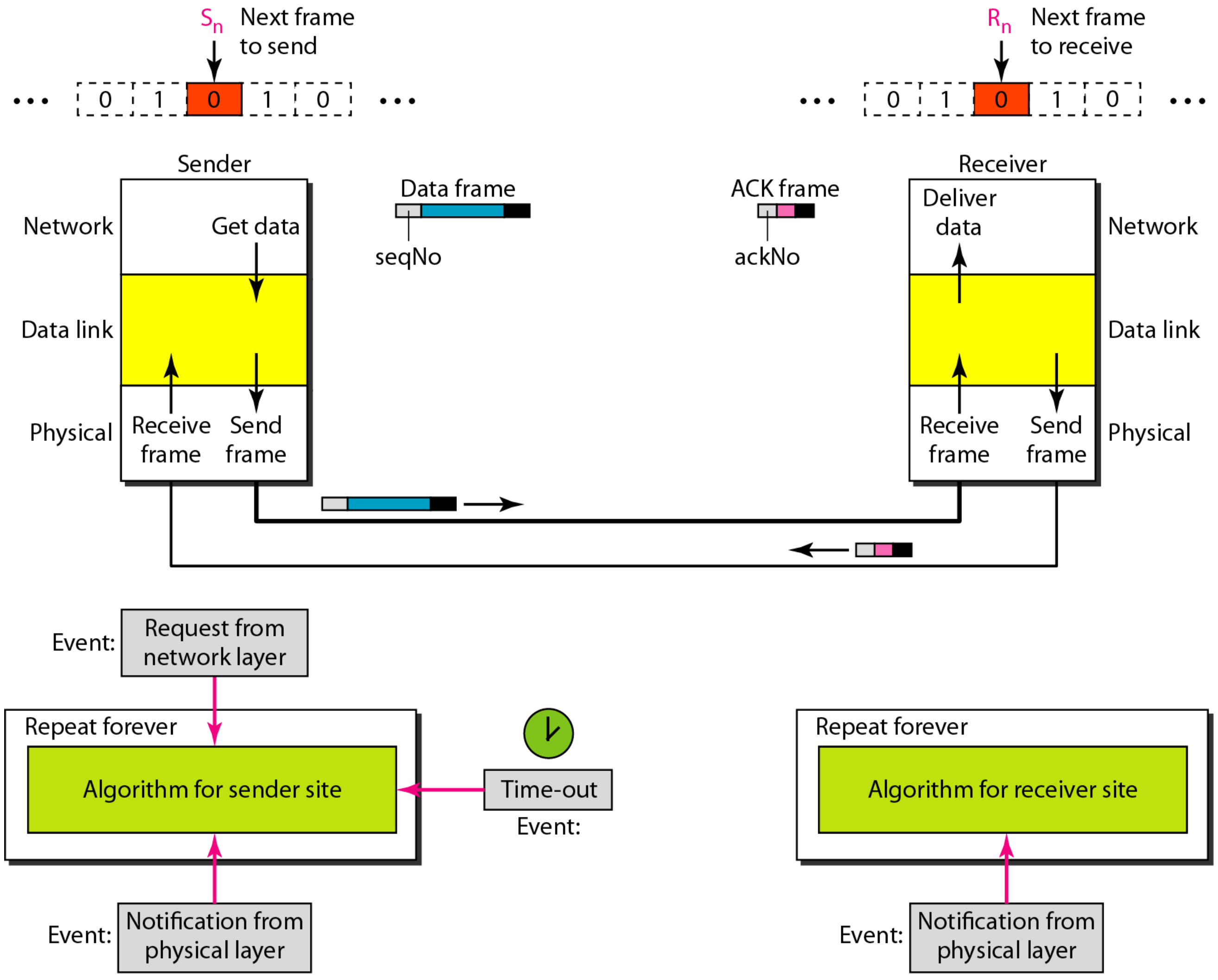


- *But, noiseless channels are nonexistent.*

Stop-and-Wait Automatic Repeat ReQuest

- 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.
- In Stop-and-Wait ARQ, we use sequence numbers to number the frames. The sequence numbers are based on modulo-2 arithmetic.

Design of the Stop-and-Wait ARQ Protocol



Sender-site algorithm for Stop-and-Wait ARQ

```
S_n = 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(S_n);                     //The seqNo is S_n
     StoreFrame(S_n);                    //Keep copy
     SendFrame(S_n);
     StartTimer();
     S_n = S_n + 1;
     canSend = false;
  }
  WaitForEvent();                        // Sleep
```

Continued

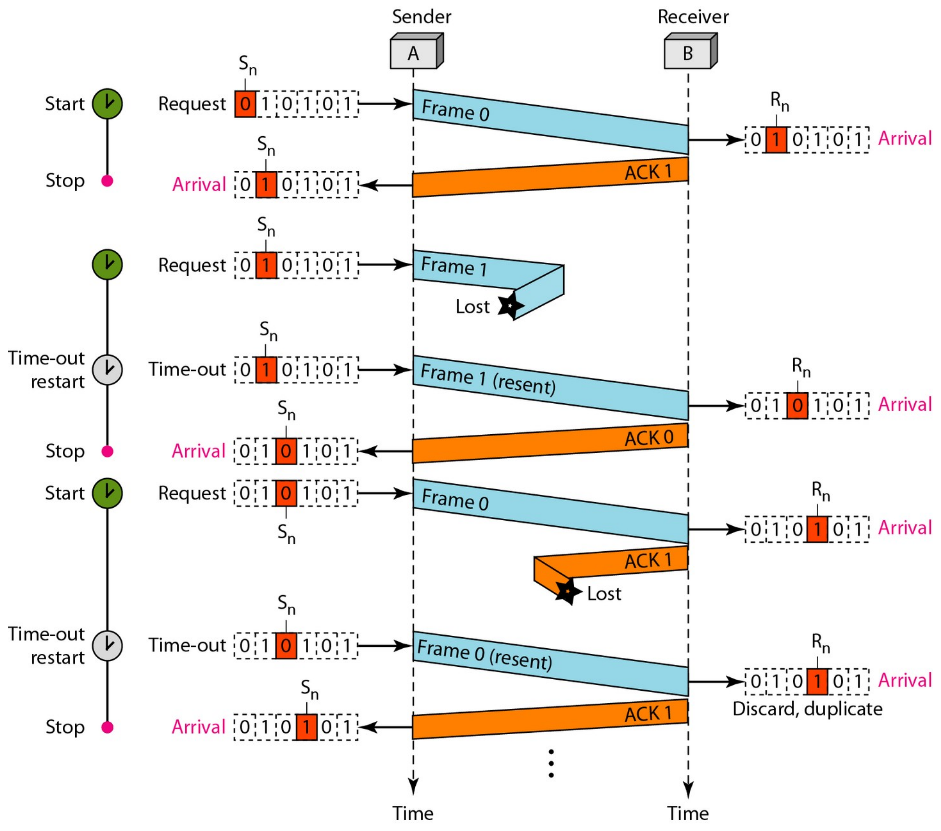
```
    if(Event(ArrivalNotification)      // An ACK has arrived
    {
      ReceiveFrame(ackNo);             //Receive the ACK frame
      if(not corrupted AND ackNo == $S_n$) //Valid ACK
        {
          Stoptimer();
          PurgeFrame($S_{n-1}$);          //Copy is not needed
          canSend = true;
        }
     }

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

Receiver-site algorithm for Stop-and-Wait ARQ Protocol

```
R_n = 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 == R_n)              //Valid data frame
     {
      ExtractData();
       DeliverData();              //Deliver data
       R_n = R_n + 1;
     }
      SendFrame(R_n);              //Send an ACK
  }
}
```

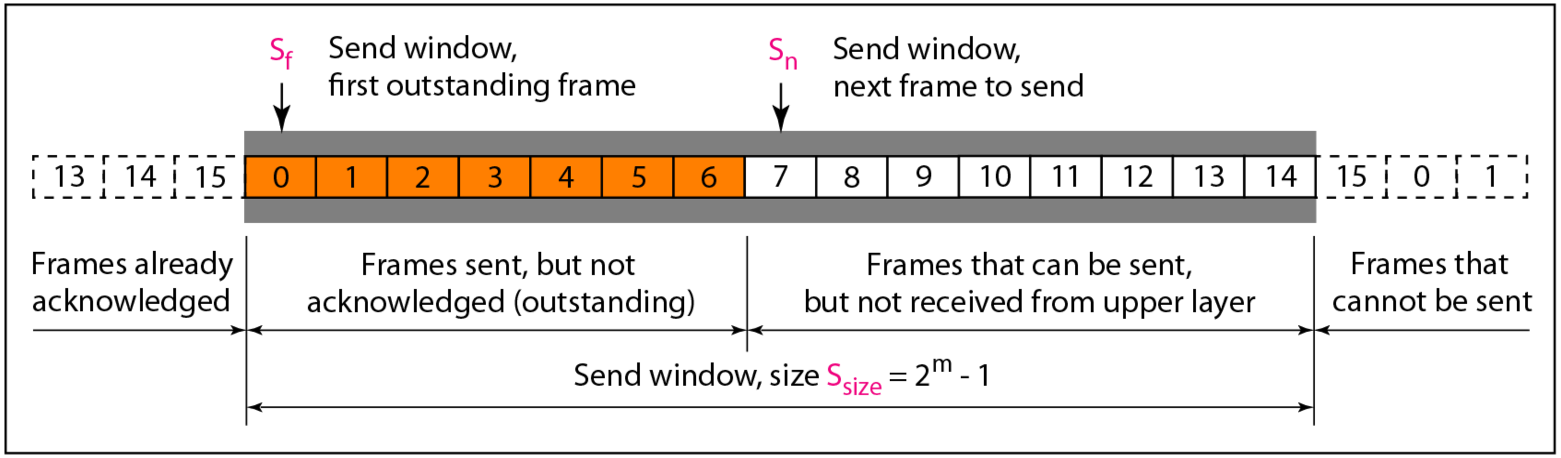
Flow diagram

- *Frame 0 is sent and acknowledged.*
- *Frame 1 is lost and resent after the time-out.*
- *The resent frame 1 is acknowledged and the timer stops.*
- *Frame 0 is sent and acknowledged, but the acknowledgment is lost.*
- *The sender has no idea if the frame or the acknowledgment is lost, so after the time-out, it resends frame 0, which is acknowledged.*



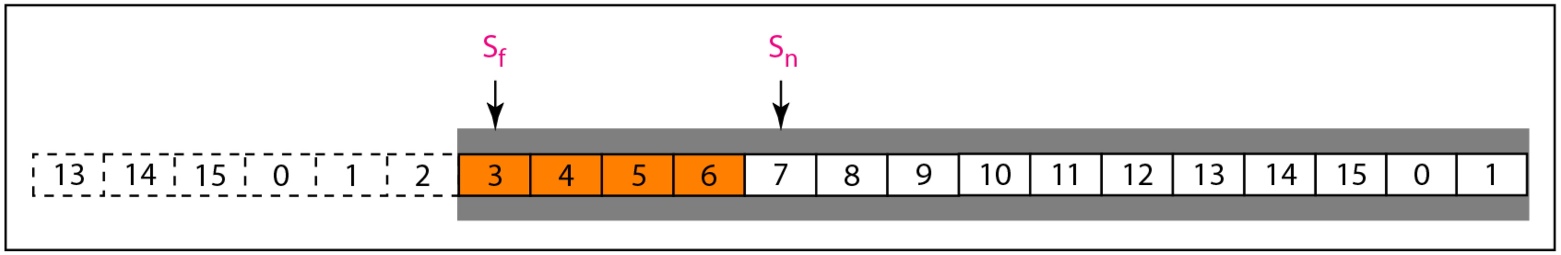
Go-Back-N Protocol

- In the Go-Back-N Protocol, the sequence numbers are modulo 2^m, where m is the size of the sequence number field in bits.
- The send window is an abstract concept defining an imaginary box of size $2^m - 1$ with three variables: Sf, Sn, and S_{size}.
- The send window can slide one or more slots when a valid acknowledgment arrives.



S_f
Send window, first outstanding frame
S_n
Send window, next frame to send
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already acknowledged
Frames sent, but not acknowledged (outstanding)
Frames that can be sent, but not received from upper layer
Frames that cannot be sent
Send window, size $S_{size} = 2^m - 1$

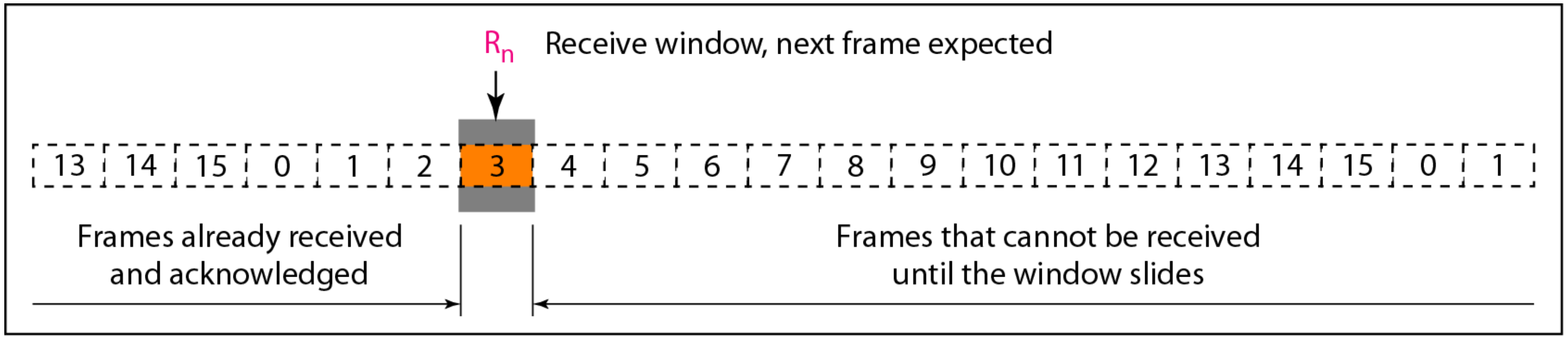
a. Send window before sliding

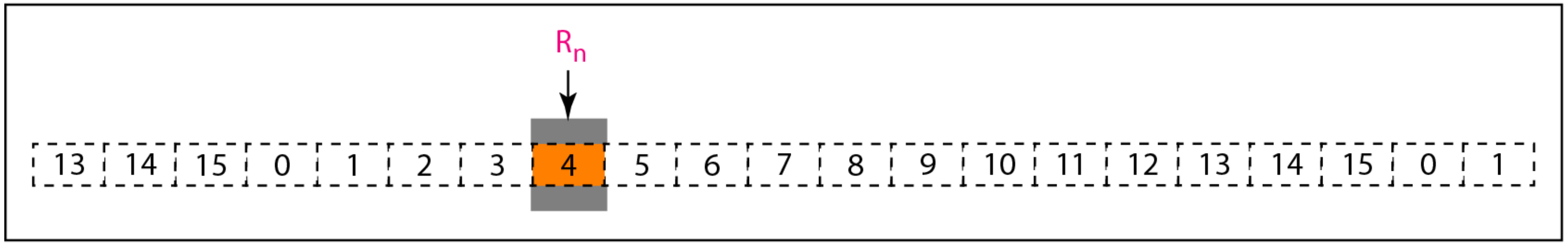


b. Send window after sliding

Receive window for Go-Back-N ARQ

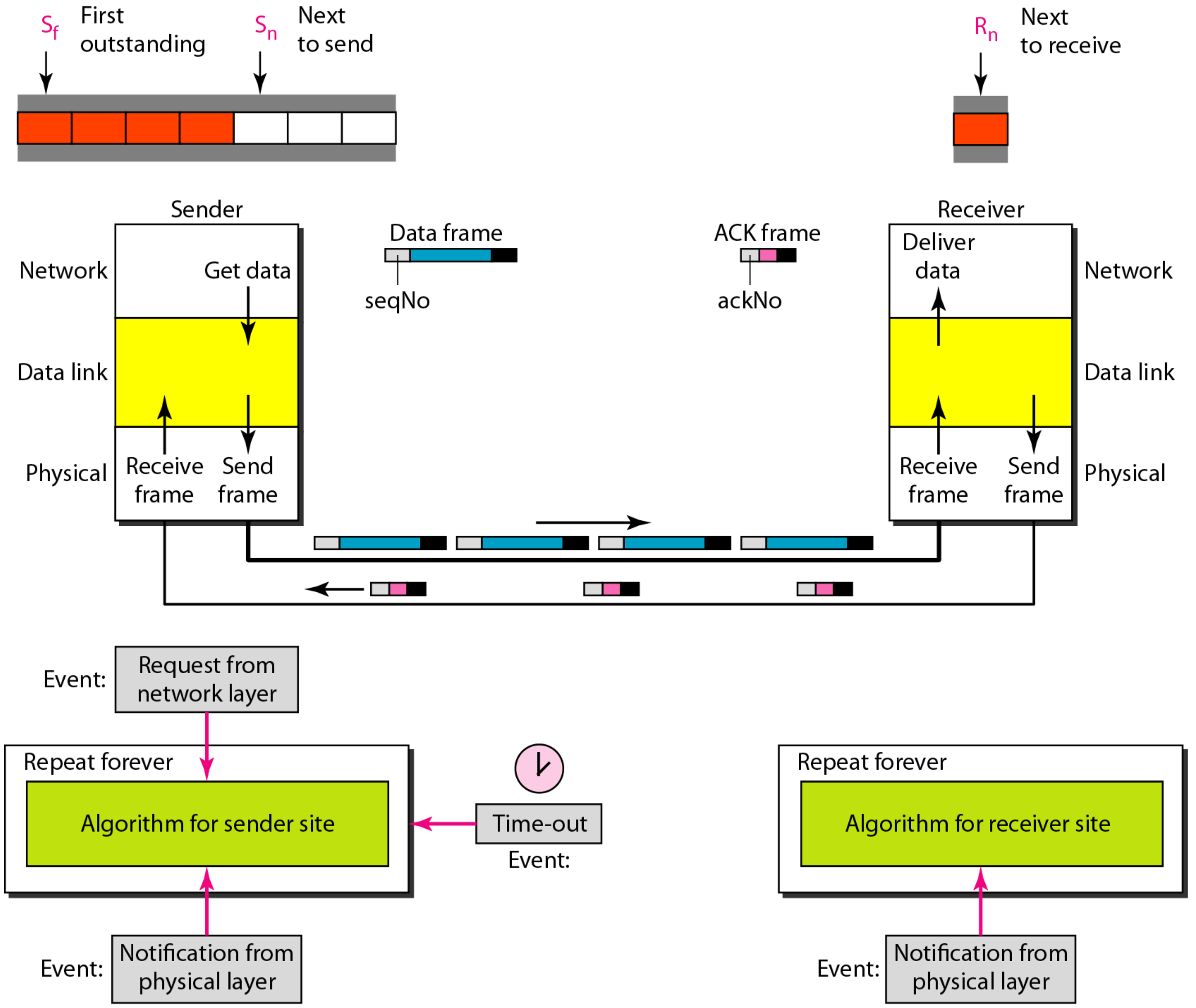
- The receive window is an abstract concept defining an imaginary box of size 1 with one single variable R_n. The window slides when a correct frame has arrived; sliding occurs one slot at a time.


R_n
Receive window, next frame expected
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already received and acknowledged
Frames that cannot be received until the window slides
a. Receive window

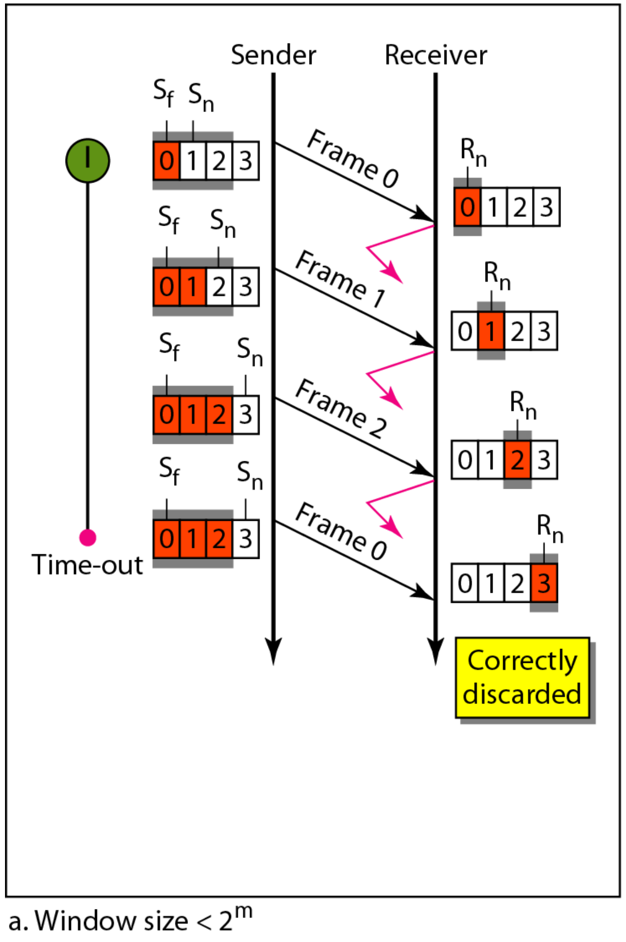


b. Window after sliding

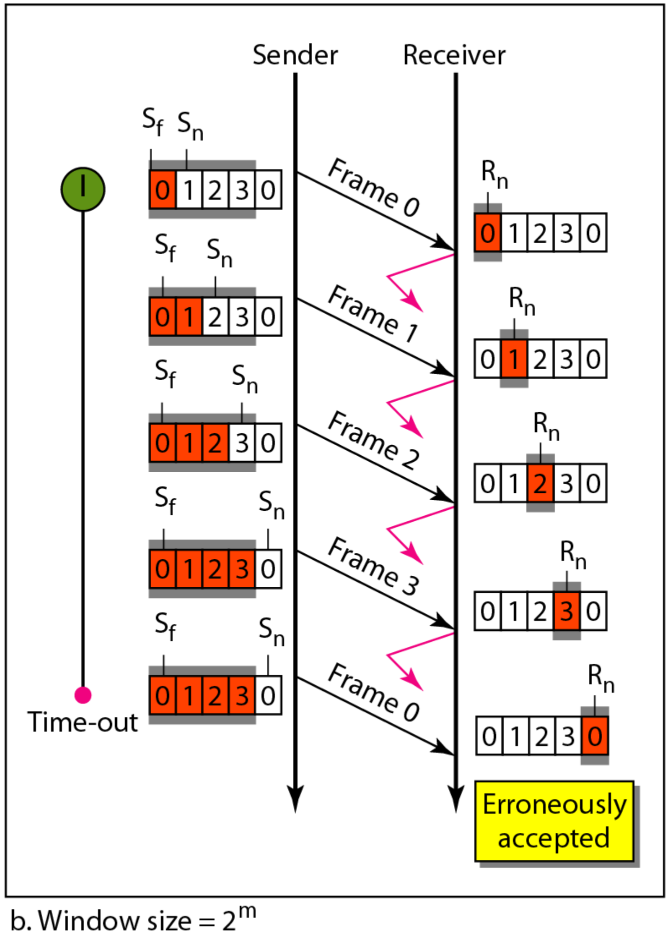
Design of Go-Back-N ARQ



Window size for Go-Back-N ARQ



a. Window size $< 2^m$



b. Window size $= 2^m$

- In Go-Back-N ARQ, the size of the send window must be less than 2^m; the size of the receiver window is always 1.

Go-Back-N sender algorithm

```
$S_w$ = $2^m$ - 1;
$S_f$ = 0;
$S_n$ = 0;

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

```

Continued

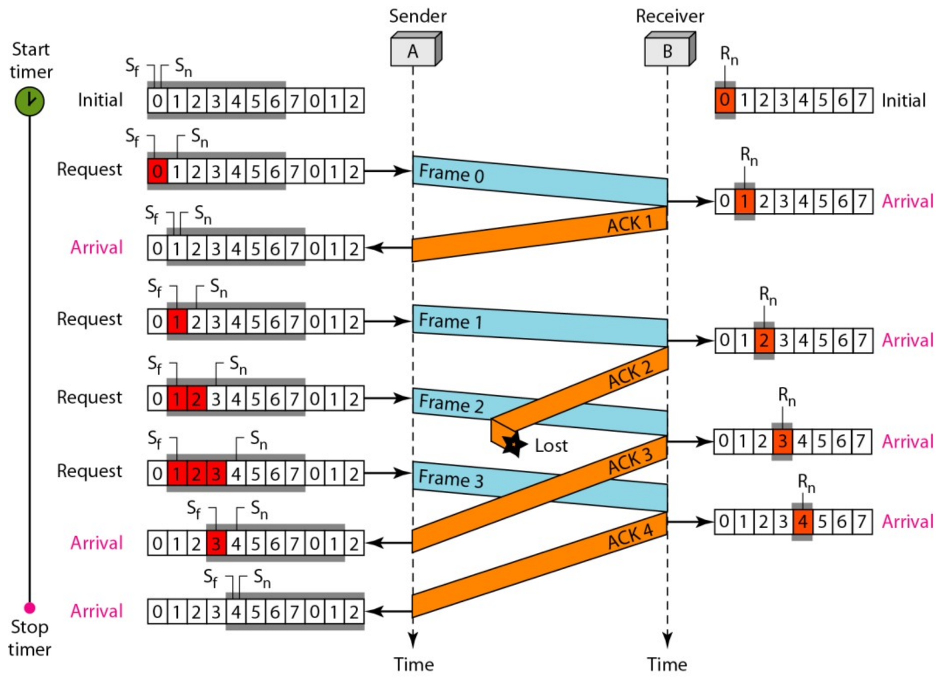
```
  if(Event(ArrivalNotification)) //ACK arrives
  {
     Receive(ACK);
     if(corrupted(ACK))
          Sleep();
     if((ackNo>Sf)&&(ackNo<=Sn))  //If a valid ACK
     While(Sf <= ackNo)
      {
       PurgeFrame(Sf);
       Sf = Sf + 1;
      }
      StopTimer();
  }

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

Go-Back-N receiver algorithm

```
R_n = 0;

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

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

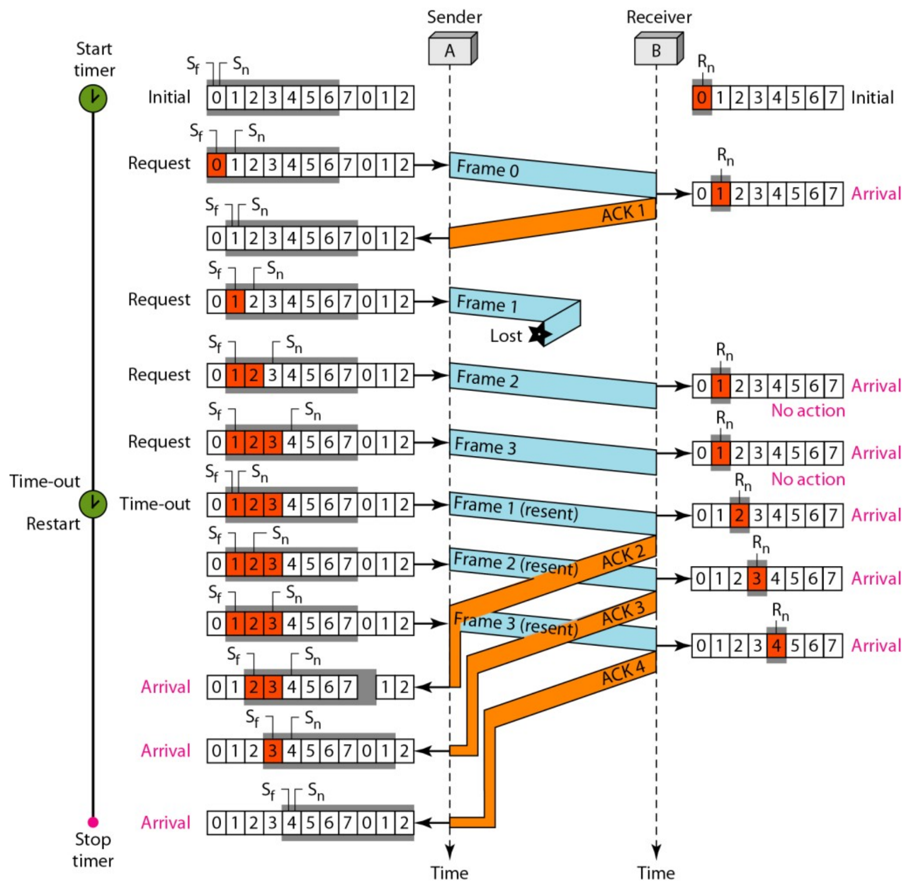
Example: Go-Back-N flow diagram

- This is an example of a case where the forward channel is reliable, but the reverse is not.
- No data frames are lost, but some ACKs are delayed and one is lost. The example also shows how cumulative acknowledgments can help if acknowledgments are delayed or lost.
- After initialization, there are seven sender events. Request events are triggered by data from the network layer; arrival events are triggered by acknowledgments from the physical layer.
- There is no time-out event here because all outstanding frames are acknowledged before the timer expires.
- Note that although ACK 2 is lost, ACK 3 serves as both ACK 2 and ACK 3.



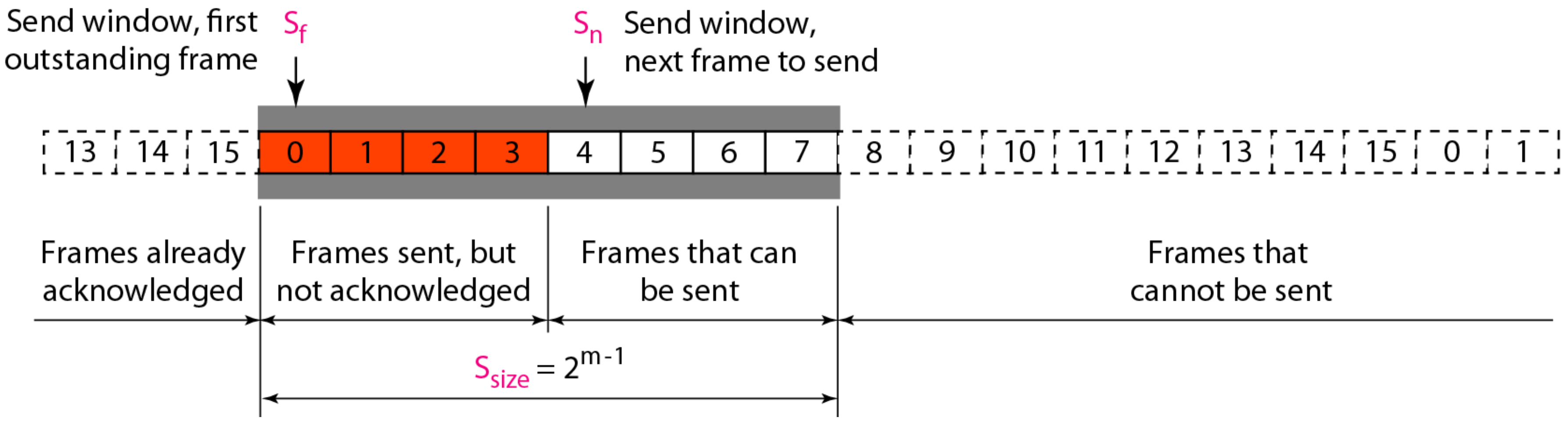
What happens when a frame is lost?

- *Frames 0, 1, 2, and 3 are sent. However, frame 1 is lost. The receiver receives frames 2 and 3, but they are discarded because they are received out of order. The sender receives no acknowledgment about frames 1, 2, or 3. Its timer finally expires. The sender sends all outstanding frames (1, 2, and 3) because it does not know what is wrong. Note that the resending of frames 1, 2, and 3 is the response to one single event. When the sender is responding to this event, it cannot accept the triggering of other events. This means that when ACK 2 arrives, the sender is still busy with sending frame 3.*

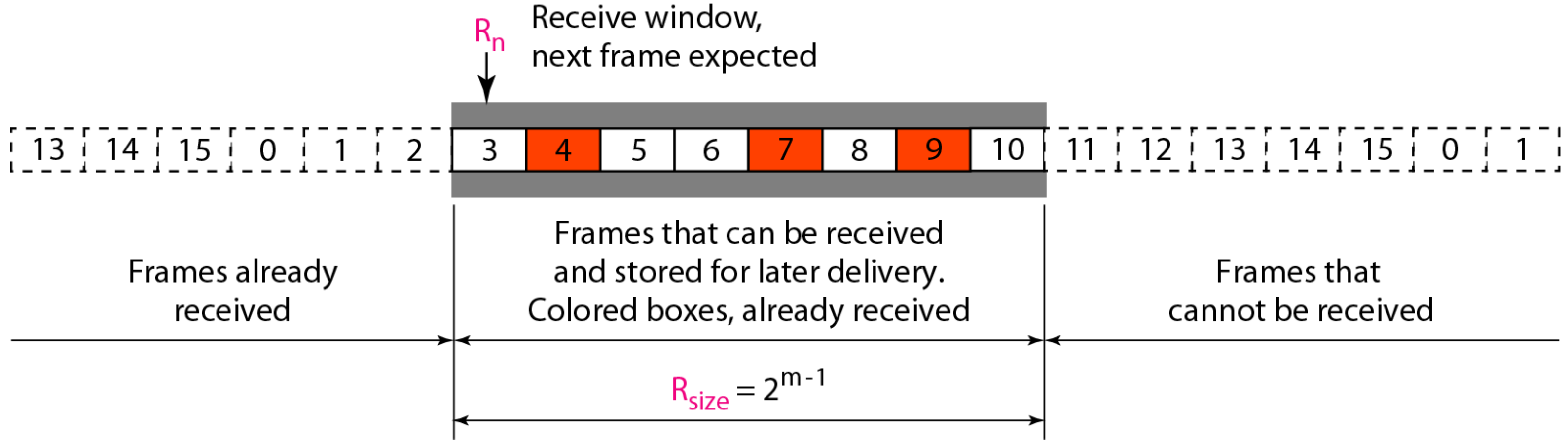


- Stop-and-Wait ARQ is a special case of Go-Back-N ARQ in which the size of the send window is 1.
- Next protocol is Selective ARQ…

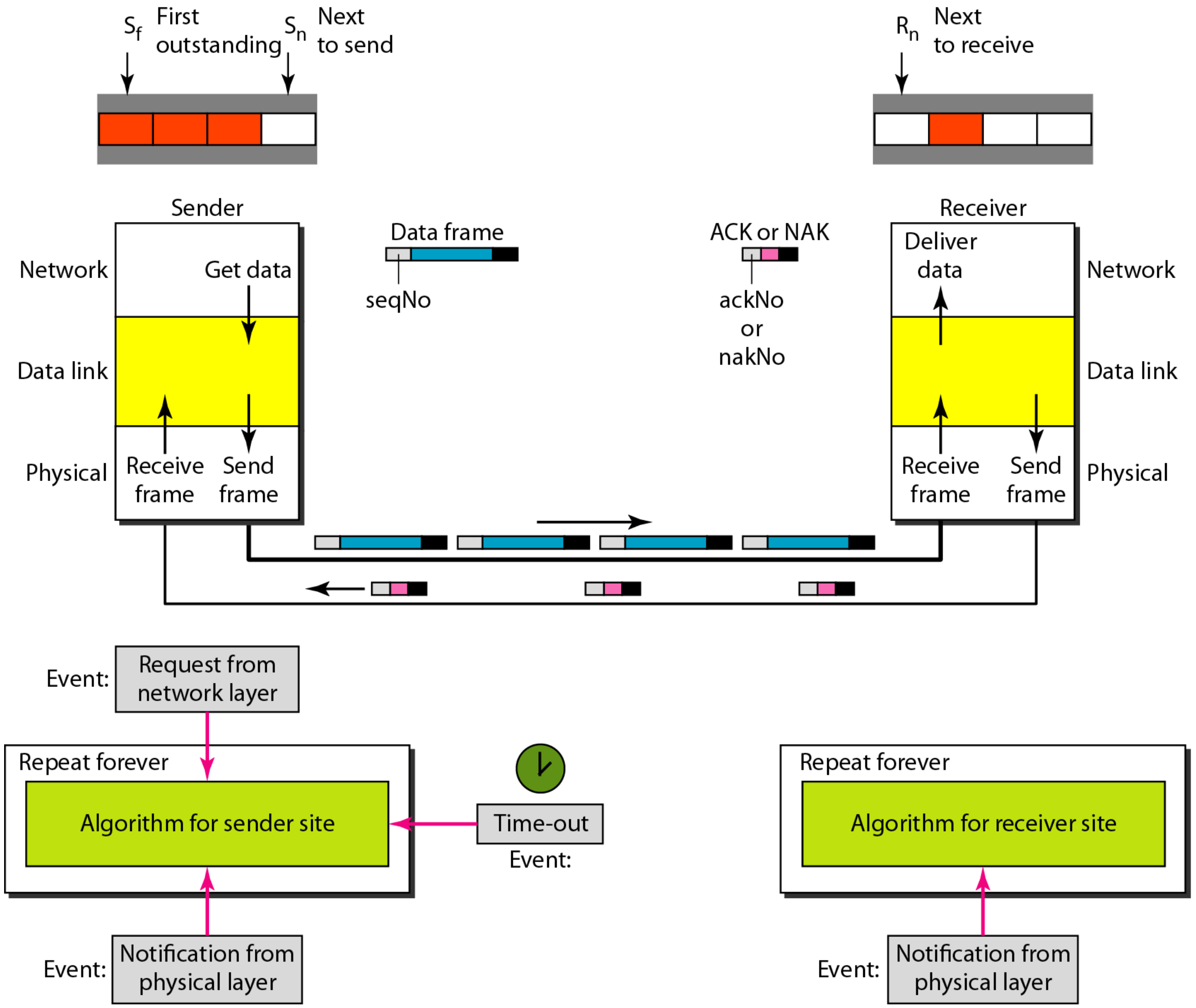
Send window for Selective Repeat ARQ


Send window, first outstanding frame
S_f
S_n
Send window, next frame to send
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already acknowledged
Frames sent, but not acknowledged
Frames that can be sent
Frames that cannot be sent
$S_{size} = 2^{m-1}$

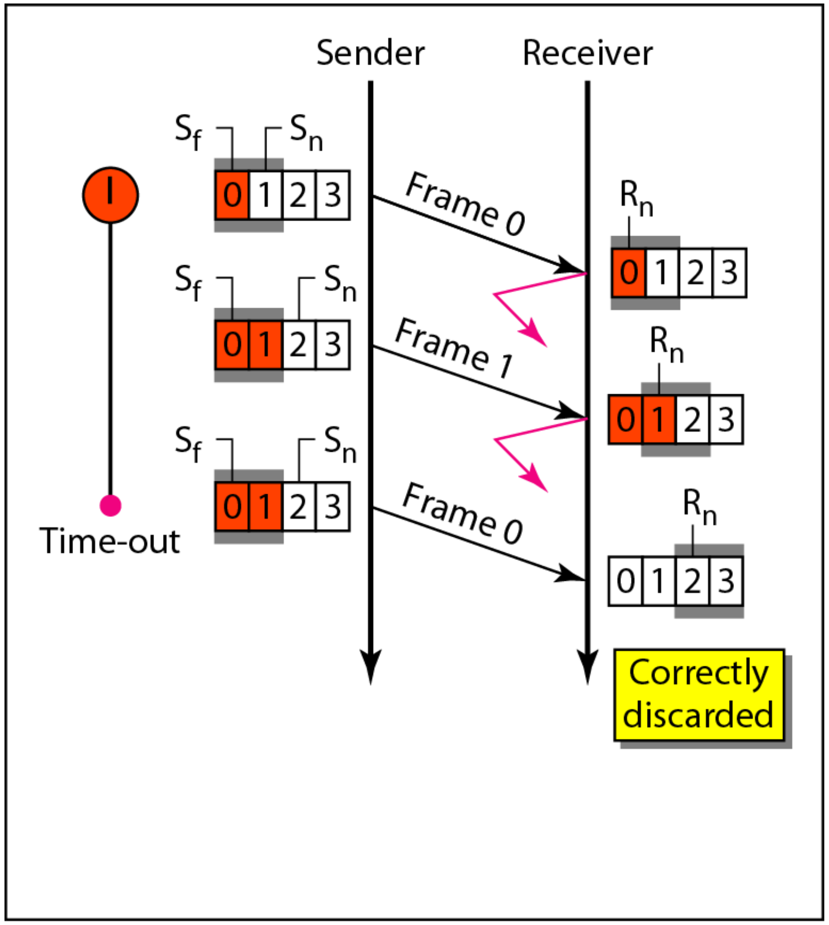
Receive window for Selective Repeat ARQ


R_n
Receive window,
next frame expected
13
14
15
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
0
1
Frames already received
Frames that can be received and stored for later delivery. Colored boxes, already received
Frames that cannot be received
$R_{size} = 2^{m-1}$

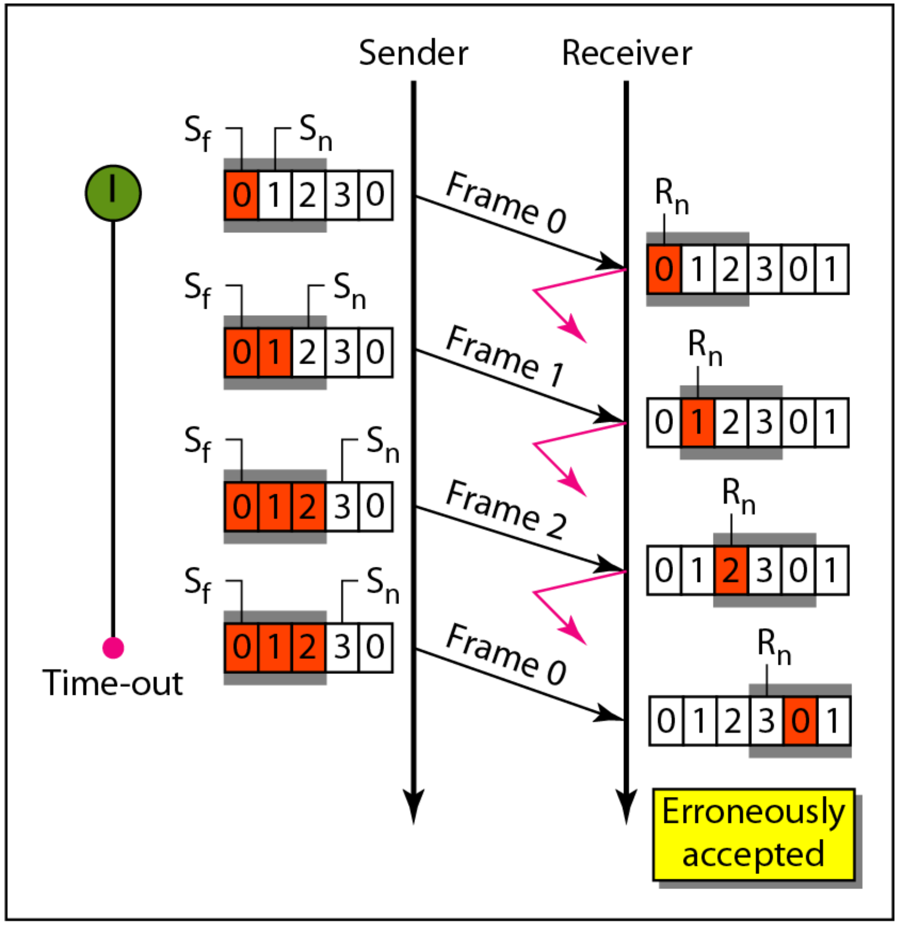
Design of Selective Repeat ARQ



Selective Repeat ARQ, window size



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



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

In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of 2^m.

Sender-site Selective Repeat algorithm

```
S_w = 2^{m-1} ;
S_f = 0;
S_n = 0;

while (true)                                //Repeat forever
{
  WaitForEvent();
  if(Event(RequestToSend))                  //There is a packet to send
  {
     if(S_n-S_f >= S_w)                     //If window is full
           Sleep();
     GetData();
     MakeFrame(S_n);
     StoreFrame(S_n);
     SendFrame(S_n);
     S_n = S_n + 1;
     StartTimer(S_n);
  }

```

Continued

```
   if(Event(ArrivalNotification)) //ACK arrives
   {
      Receive(frame);             //Receive ACK or NAK
      if(corrupted(frame))
           Sleep();
      if (FrameType == NAK)
         if (nakNo between $S_f$ and $S_n$)
         {
          resend(nakNo);
          StartTimer(nakNo);
         }
      if (FrameType == ACK)
         if (ackNo between $S_f$ and $S_n$)
         {
           while($s_f$ < ackNo)
           {
            Purge($s_f$);
            StopTimer($s_f$);
            $S_f$ = $S_f$ + 1;
           }
         }
   }

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

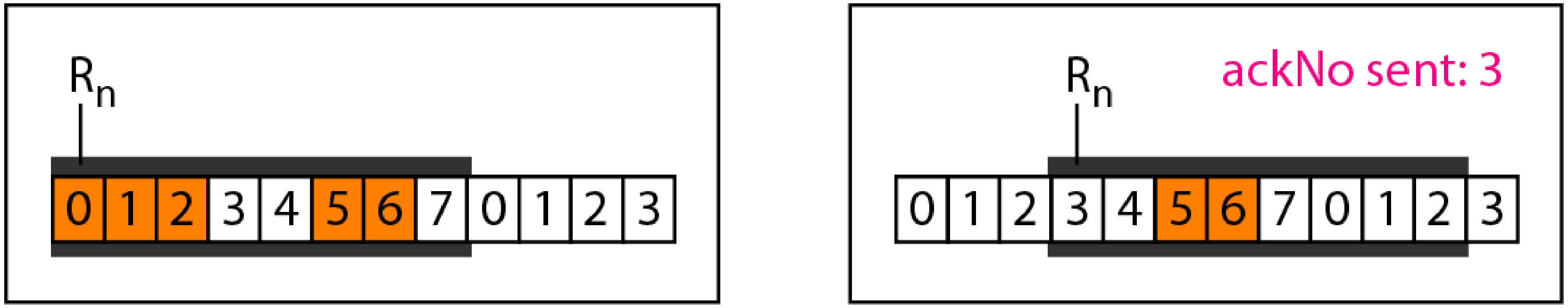
Receiver-site Selective Repeat algorithm

```
R_n = 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(R_n);
      NakSent = true;
      Sleep();
     }
     if(seqNo <> R_n)&& (NOT NakSent)
     {
      SendNAK(R_n);
```

Continued

```
        NakSent = true;
        if ((seqNo in window)&&(!Marked(seqNo))
        {
         StoreFrame(seqNo)
         Marked(seqNo)= true;
         while(Marked(R_n))
         {
          DeliverData(R_n);
          Purge(R_n);
          R_n = R_n + 1;
          AckNeeded = true;
         }
          if(AckNeeded);
          {
          SendAck(R_n);
          AckNeeded = false;
          NakSent = false;
          }
        }
      }
   }
}
```

Delivery of data in Selective Repeat ARQ

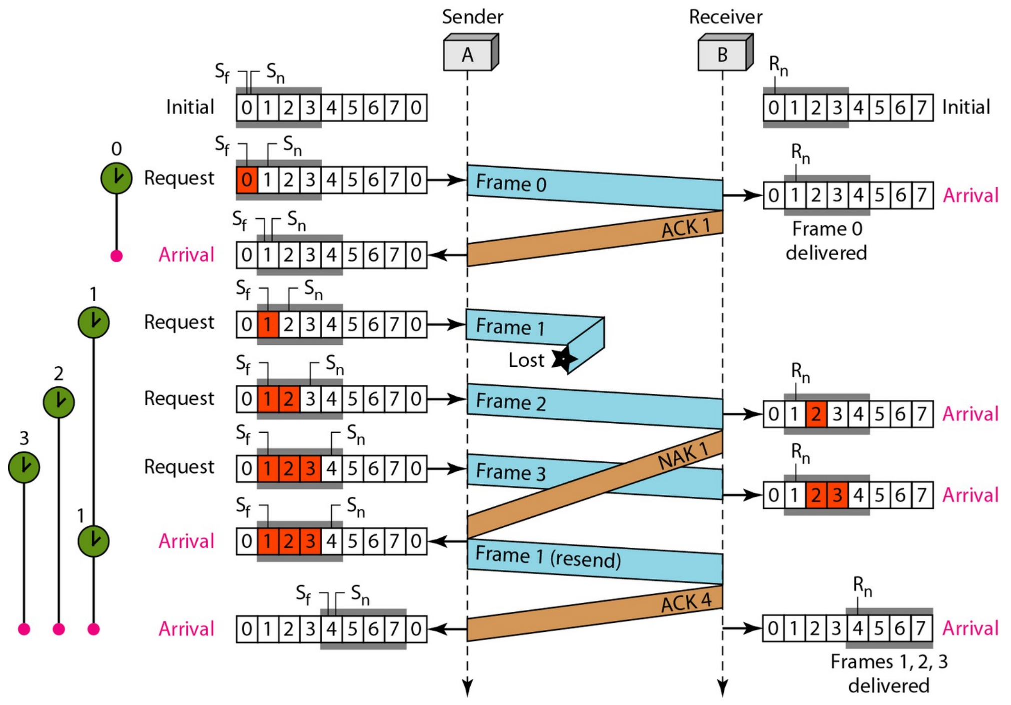


a. Before delivery

b. After delivery

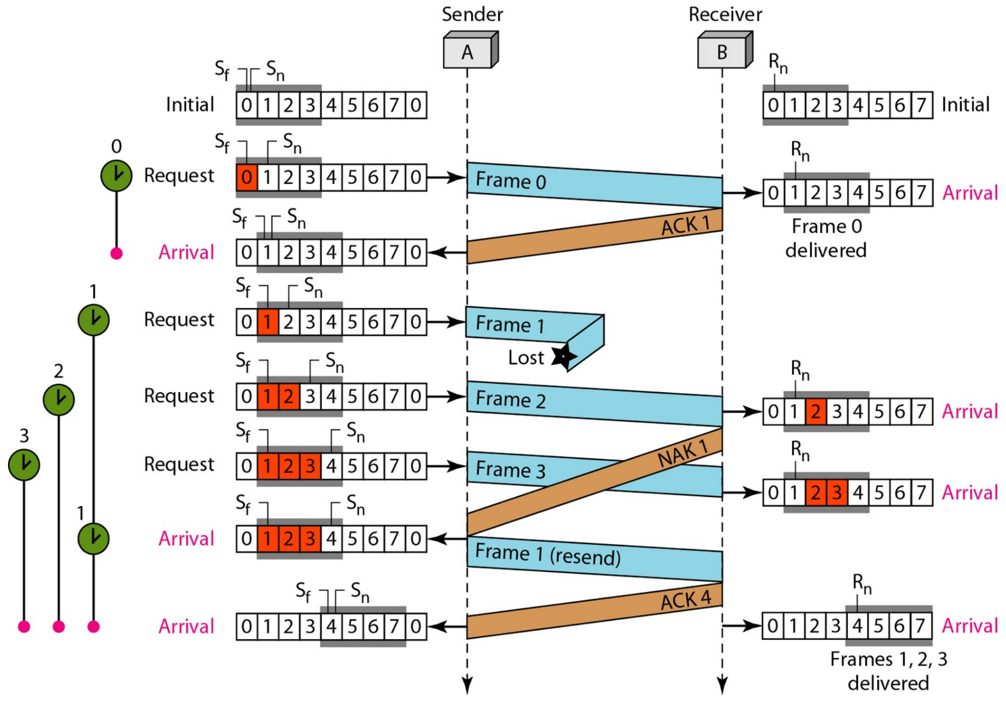
How Selective Repeat behaves when frame 1 is lost

- *One main difference is the number of timers. Here, each frame sent or resent needs a timer, which means that the timers need to be numbered (0, 1, 2, and 3). The timer for frame 0 starts at the first request, but stops when the ACK for this frame arrives. The timer for frame 1 starts at the second request, restarts when a NAK arrives, and finally stops when the last ACK arrives. The other two timers start when the corresponding frames are sent and stop at the last arrival event.*



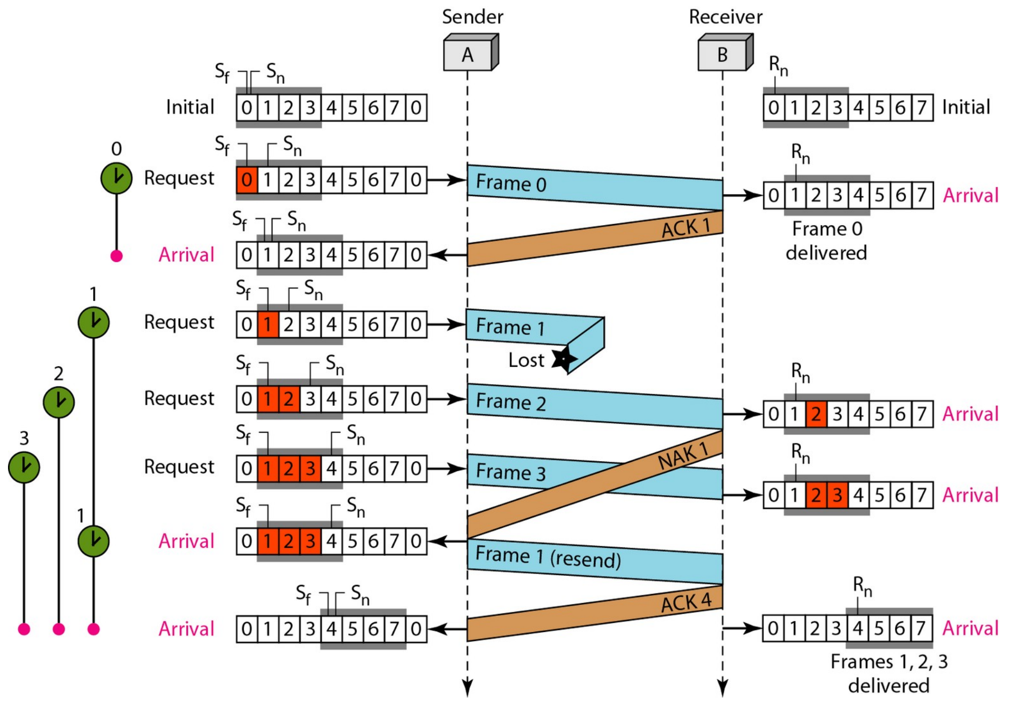
Receiver Site

- *At the receiver site we need to distinguish between the acceptance of a frame and its delivery to the network layer. At the second arrival, frame 2 arrives and is stored and marked, but it cannot be delivered because frame 1 is missing. At the next arrival, frame 3 arrives and is marked and stored, but still none of the frames can be delivered. Only at the last arrival, when finally a copy of frame 1 arrives, can frames 1, 2, and 3 be delivered to the network layer. There are two conditions for the delivery of frames to the network layer: First, a set of consecutive frames must have arrived. Second, the set starts from the beginning of the window.*



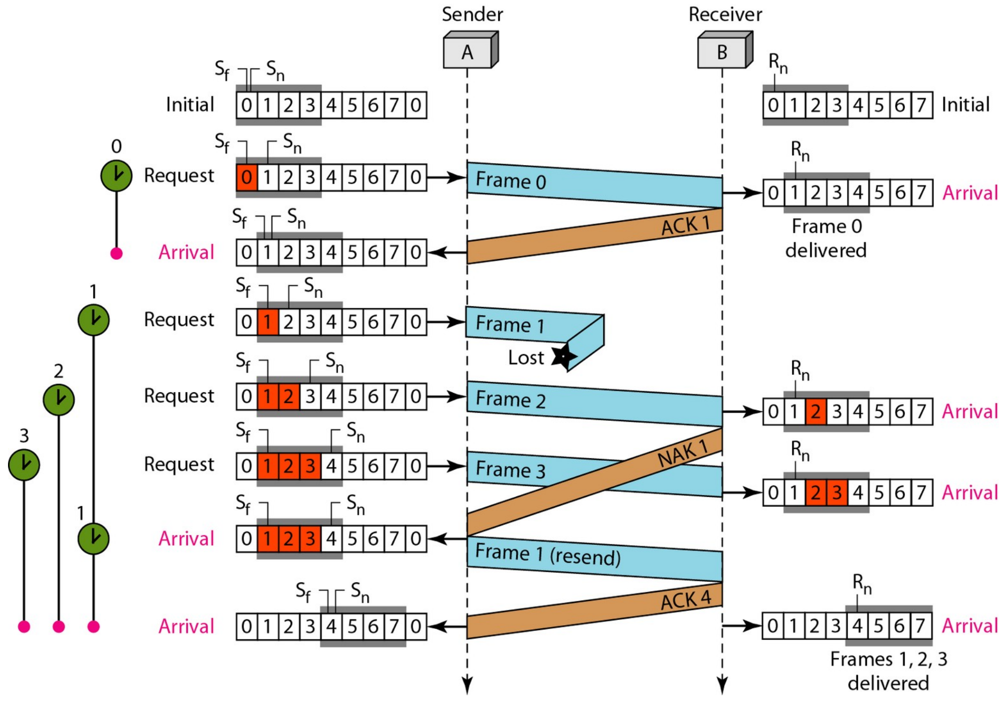
NAK

- *Another important point is that a NAK is sent after the second arrival, but not after the third, although both situations look the same. The reason is that the protocol does not want to crowd the network with unnecessary NAKs and unnecessary resent frames. The second NAK would still be NAK1 to inform the sender to resend frame 1 again; this has already been done. The first NAK sent is remembered (using the nakSent variable) and is not sent again until the frame slides. A NAK is sent once for each window position and defines the first slot in the window.*

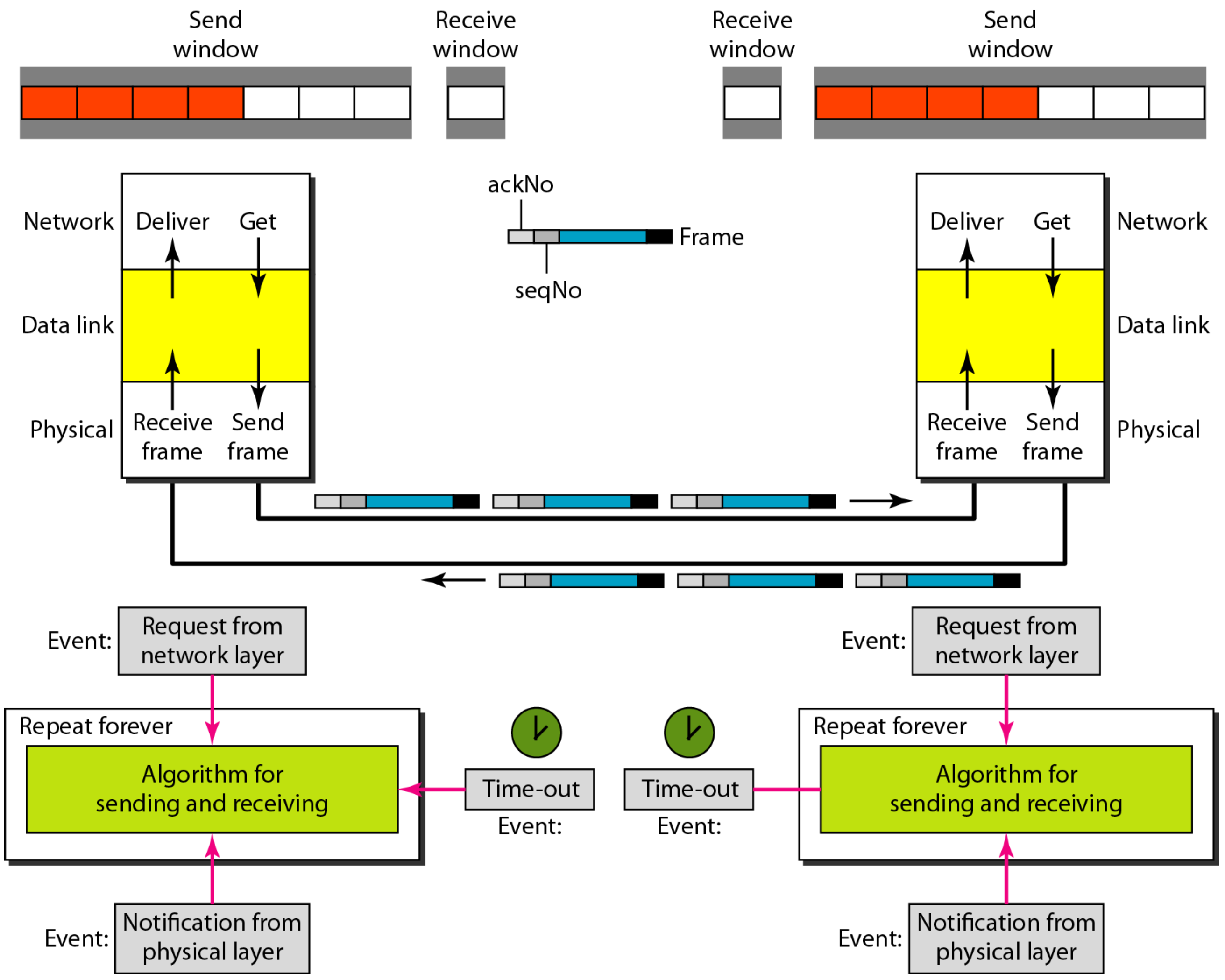


ACK

- *The next point is about the ACKs. Notice that only two ACKs are sent here. The first one acknowledges only the first frame; the second one acknowledges three frames. In Selective Repeat, ACKs are sent when data are delivered to the network layer. If the data belonging to n frames are delivered in one shot, only one ACK is sent for all of them.*

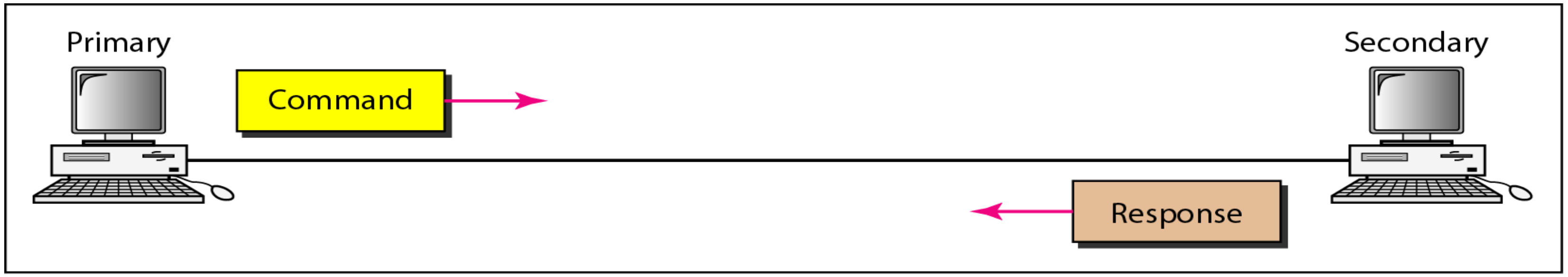


Design of piggybacking in Go-Back-N ARQ

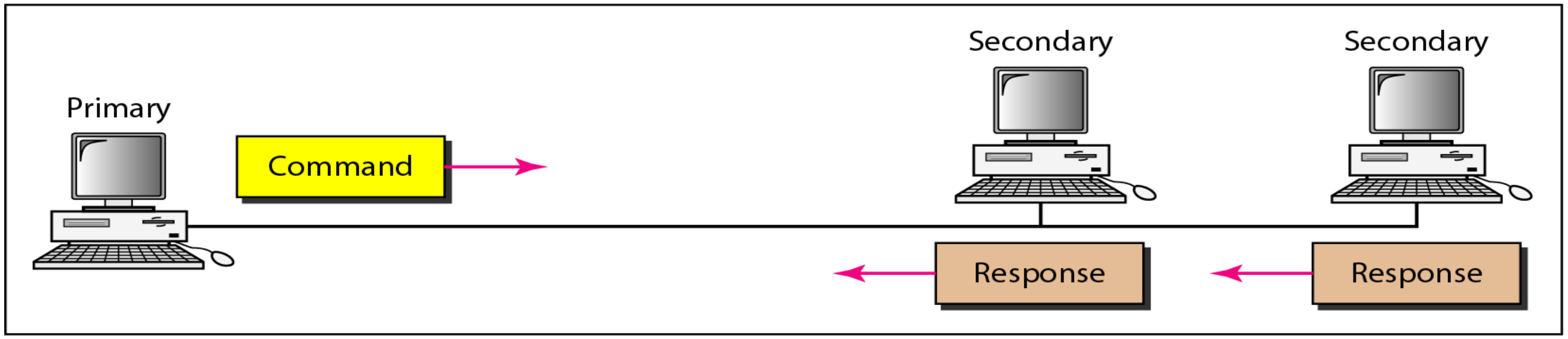


HDLC

- High-level Data Link Control (HDLC) is a bit-oriented protocol for communication over point-to-point and multipoint links. It implements the ARQ mechanisms.

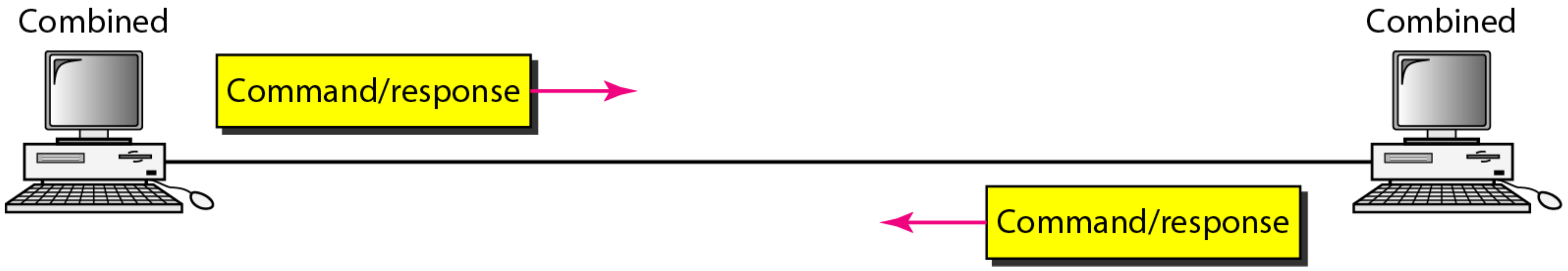


a. Point-to-point

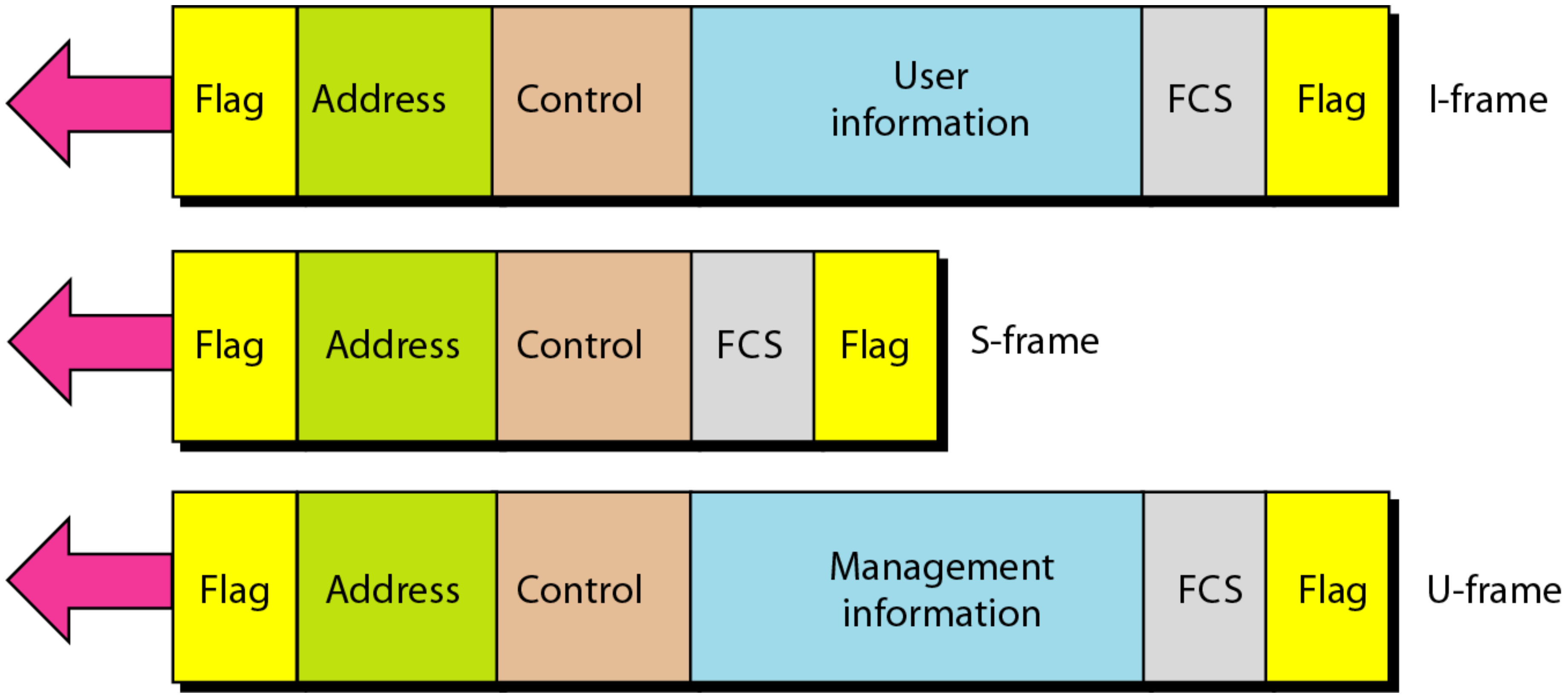


b. Multipoint

Asynchronous balanced mode



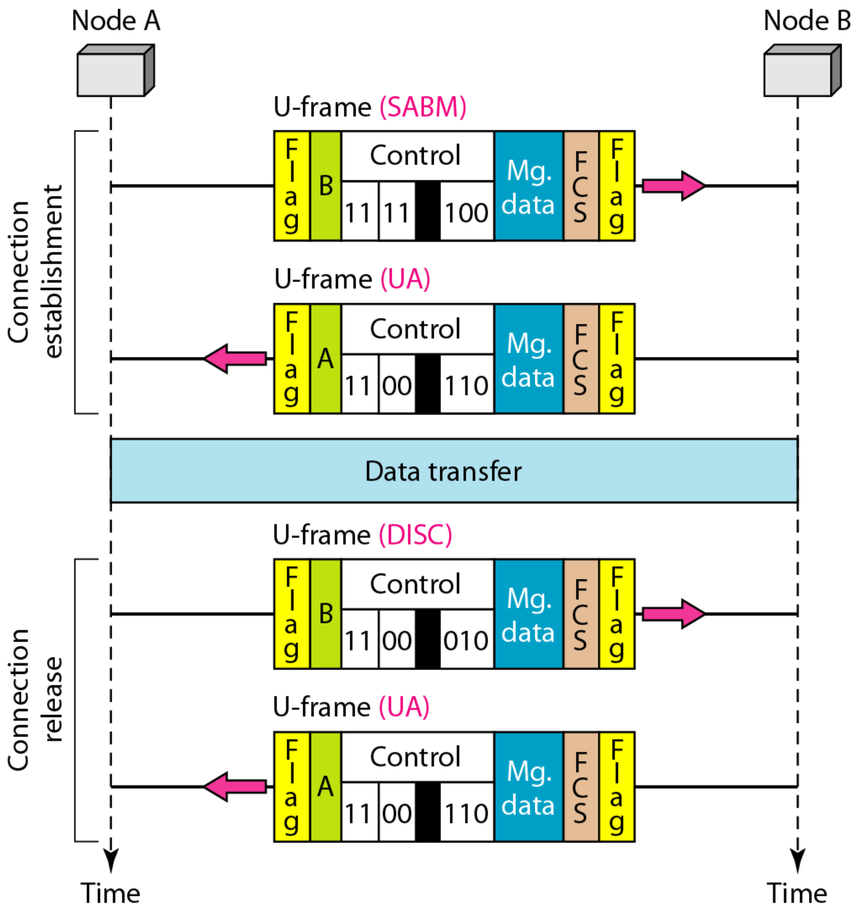
HDLC frames


Flag
Address
Control
User information
FCS
Flag
I-frame
Flag
Address
Control
FCS
Flag
S-frame
Flag
Address
Control
Management information
FCS
Flag
U-frame

U-frame control command and response

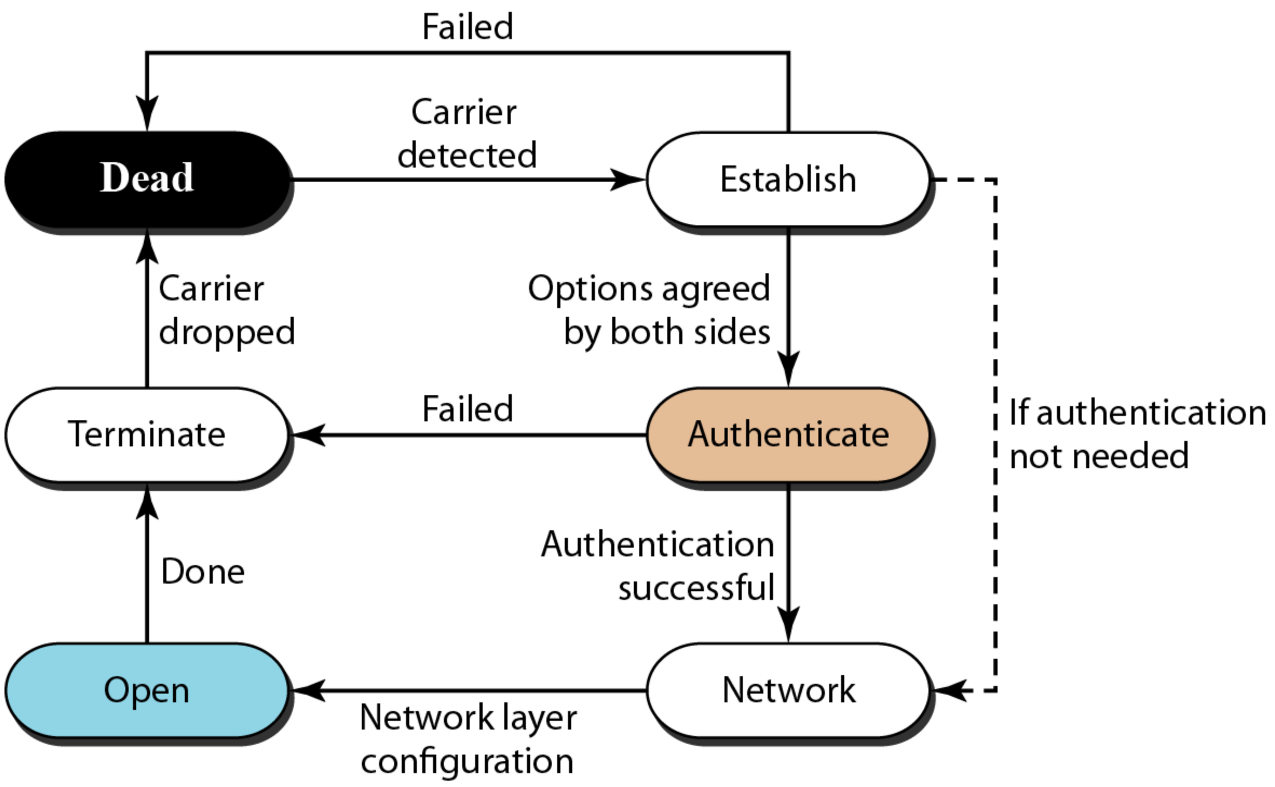
Code	*Command*	*Response*	*Meaning*
00 001	SNRM		Set normal response mode
11 011	SNRME		Set normal response mode, extended
11 100	SABM	**DM**	Set asynchronous balanced mode or **disconnect mode**
11 110	SABME		Set asynchronous balanced mode, extended
00 000	UI	**UI**	Unnumbered information
00 110		**UA**	**Unnumbered acknowledgment**
00 010	DISC	**RD**	Disconnect or **request disconnect**
10 000	SIM	**RIM**	Set initialization mode or **request information mode**
00 100	UP		Unnumbered poll
11 001	RSET		Reset
11 101	XID	**XID**	Exchange ID
10 001	FRMR	**FRMR**	Frame reject

- *Node A asks for a connection with a set asynchronous balanced mode (SABM) frame; node B gives a positive response with an unnumbered acknowledgment (UA) frame. After these two exchanges, data can be transferred between the two nodes (not shown in the figure). After data transfer, node A sends a DISC (disconnect) frame to release the connection; it is confirmed by node B responding with a UA (unnumbered acknowledgment).*



PPP

- Although HDLC is a general protocol that can be used for both point-to-point and multipoint configurations, one of the most common protocols for point-to-point access is the Point-to-Point Protocol (PPP). PPP is a byte-oriented protocol.


Failed
Carrier detected
Dead
Establish
Carrier dropped
Options agreed by both sides
Terminate
Failed
Authenticate
If authentication not needed
Done
Authentication successful
Open
Network
Network layer configuration