Data Communications and Networking Fourth Edition

upto page 71

Chapter 11

Data Link Control

11.1 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.

Topics discussed in this section:

Fixed-Size Framing
Variable-Size Framing

Framing

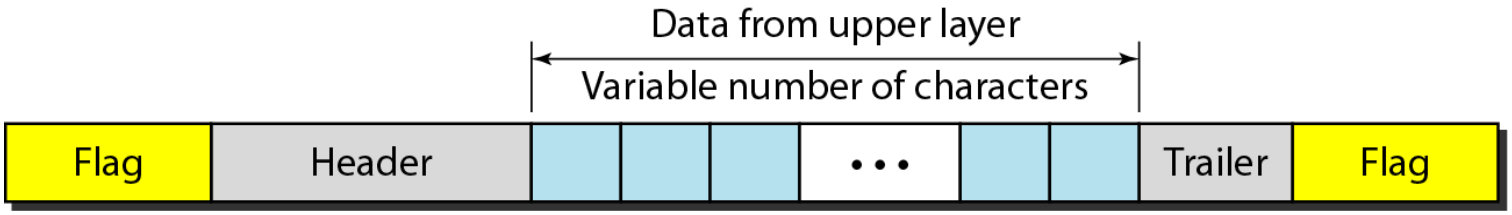
- Fixed-Size Framing
  - There is no need for defining the boundaries of the frame; the size itself can be used as a delimiter.
- Variable-Size Framing
  - Character-Oriented Protocols
    - Data to be carried are 8bit characters from a coding system.
  - Bit-Oriented Protocols
    - The data section of a frame is a sequence of bits to be interpreted by the upper layer as text, graphic, audio, video, and so on.

Framing

Character-Oriented Protocols

- To separate one frame from the next, an 8-bit(1byte) flag, composed of protocol-dependent special characters, is added at the beginning and the end of a frame.

Figure 11.1 *A frame in a character-oriented protocol*



Framing

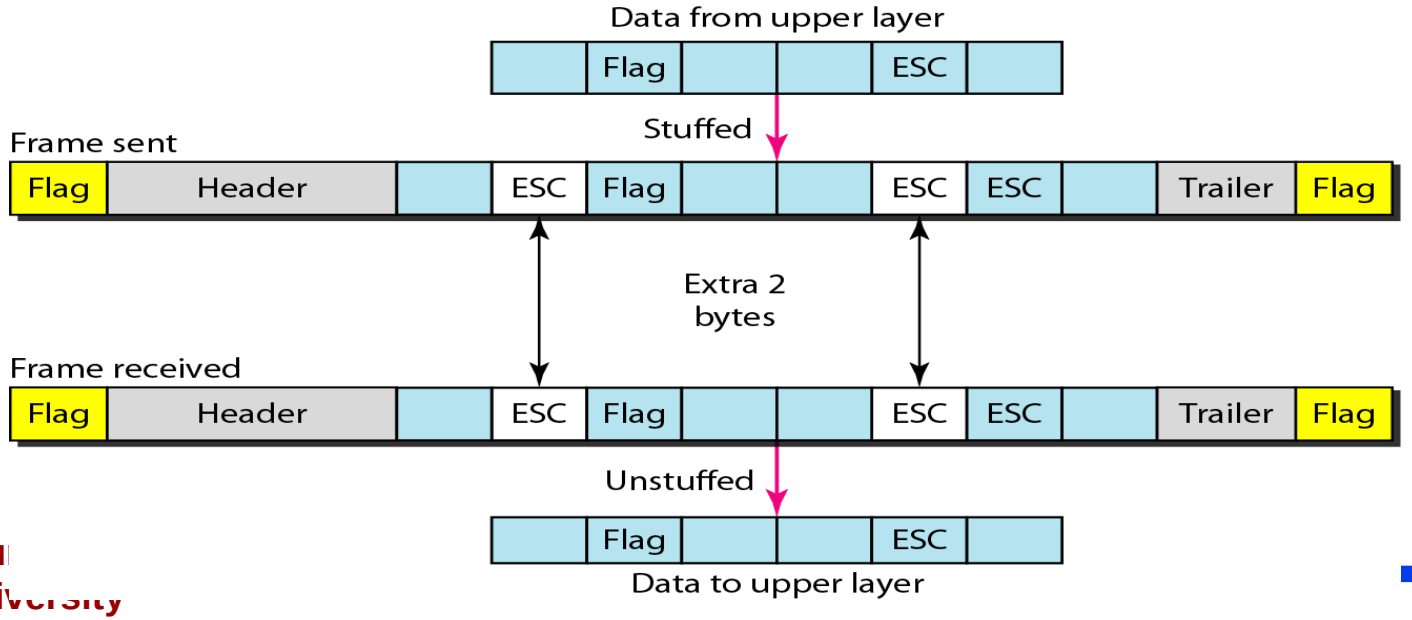
- **Character-Oriented Protocols**
  - **Any pattern used for the flag could also be part of the information.**
  - **To fix this problem, a byte-stuffing strategy was added to character-oriented framings.**

Byte stuffing is the process of adding 1 extra byte (ESC-predefined bit pattern) whenever there is a flag or escape character in the text.

Character-Oriented Protocols

- Whenever the receiver encounters the ESC character, receiver removes it from the data section and treats the next character as data, not a delimiting flag.
- The escape characters (ESC) that are part of the text must also be marked by another escape character (ESC).

Figure 11.2 *Byte stuffing and unstuffing*

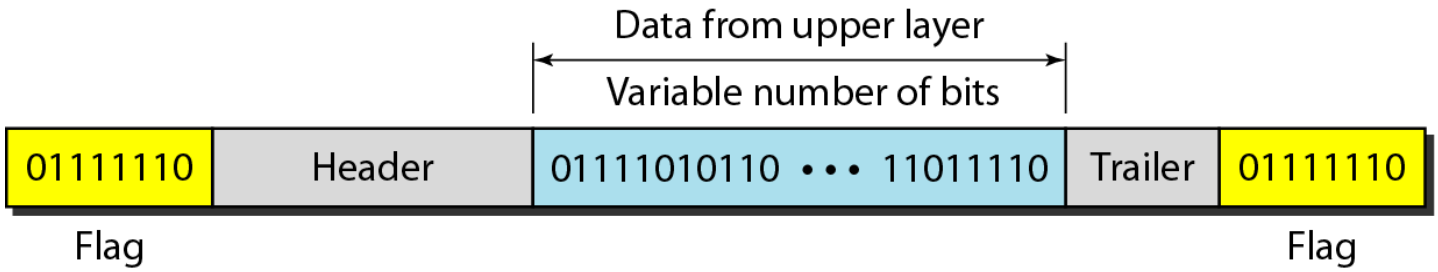


Bit-Oriented Protocols

Bit-Oriented Protocols

- Most protocols use a special 8-bit pattern flag 01111110 as the delimiter to define the beginning and the end of the frame.

Figure 11.3 *A frame in a bit-oriented protocol*

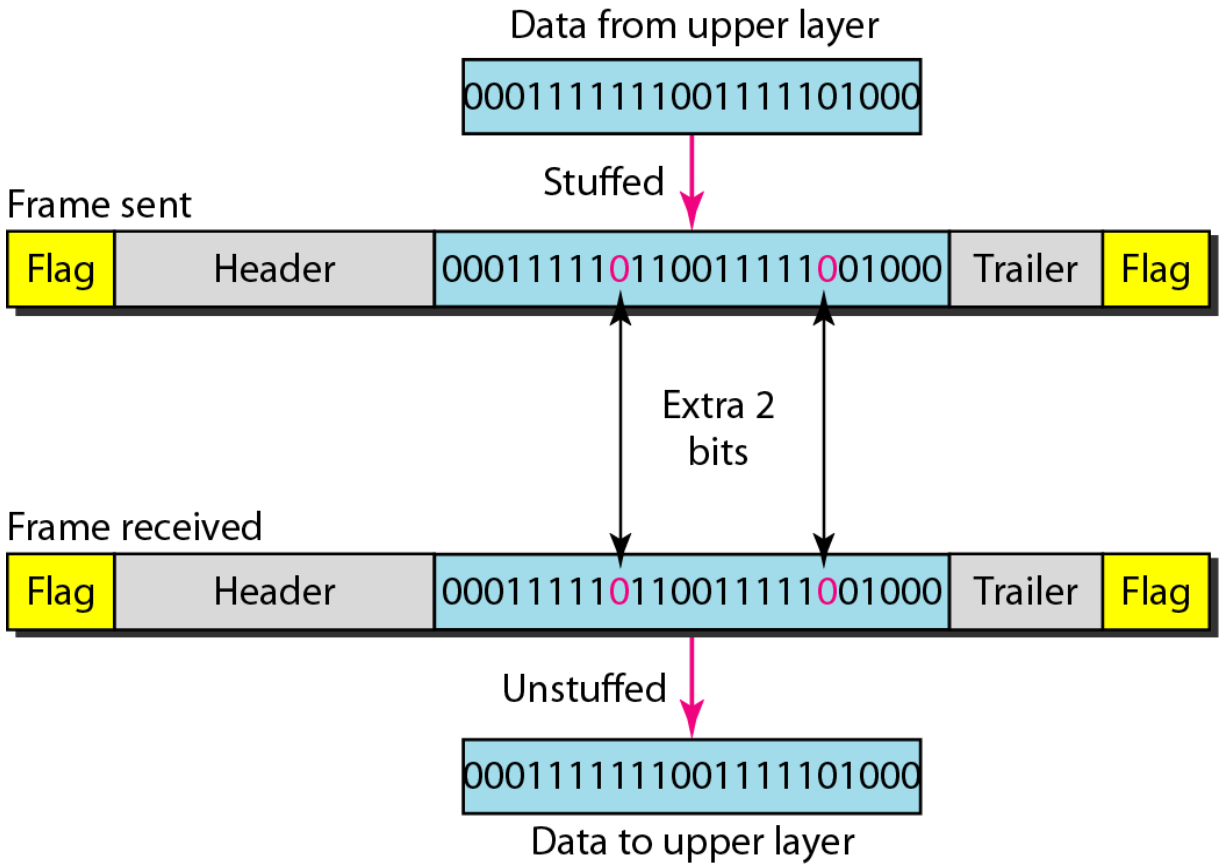


Bit-Oriented Protocols

- This flag can create the same problem in the byte-oriented protocol.
- We do this by stuffing 1 single bit (instead of 1 byte) to prevent the pattern from looking like a flag. The strategy is called Bit Stuffing.
- 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.
- This extra stuffed bit is removed from the data by the receiver

Bit-Oriented Protocols

Figure 11.4 *Bit stuffing and unstuffing*



11.2 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.

Topics discussed in this section:

Flow Control
Error Control

Flow 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.
- Flow control is the regulation of the sender's data rate so that the receiver buffer does not become overwhelmed.

Error Control

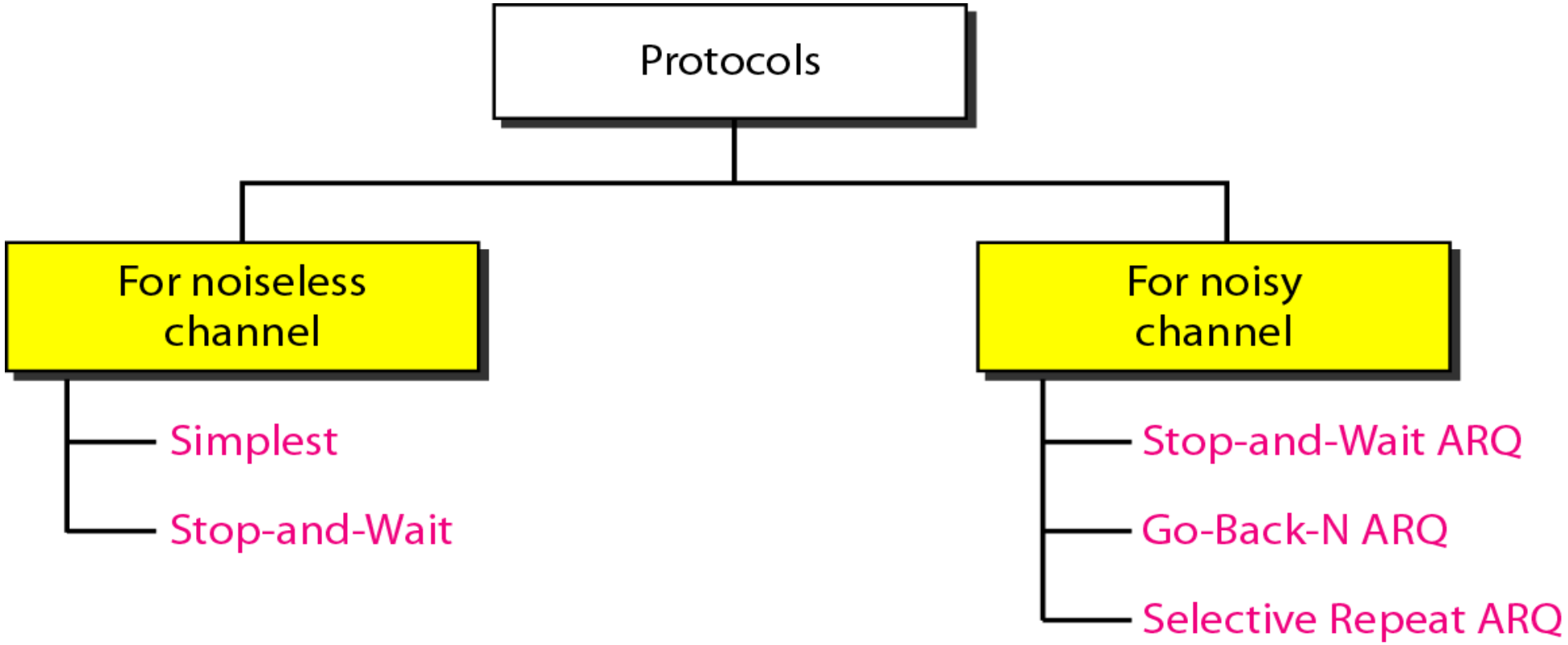
- Error control in the data link layer is based on automatic repeat request, which is the retransmission of data.
- Error control is both error detection and error correction.

11.3 PROTOCOLS

Now let us see how the data link layer can combine framing, flow control, and error control to achieve the delivery of data from one node to another. The protocols are normally implemented in software by using one of the common programming languages. To make our discussions language-free, we have written in pseudocode a version of each protocol that concentrates mostly on the procedure instead of delving into the details of language rules.

Protocols

Figure 11.5 *Taxonomy of protocols discussed in this chapter*



11.4 NOISELESS CHANNELS

Let us first assume we have an ideal channel in which no frames are lost, duplicated, or corrupted. We introduce two protocols for this type of channel.

Topics discussed in this section:

Simplest Protocol

Stop-and-Wait Protocol

Simplest Protocol

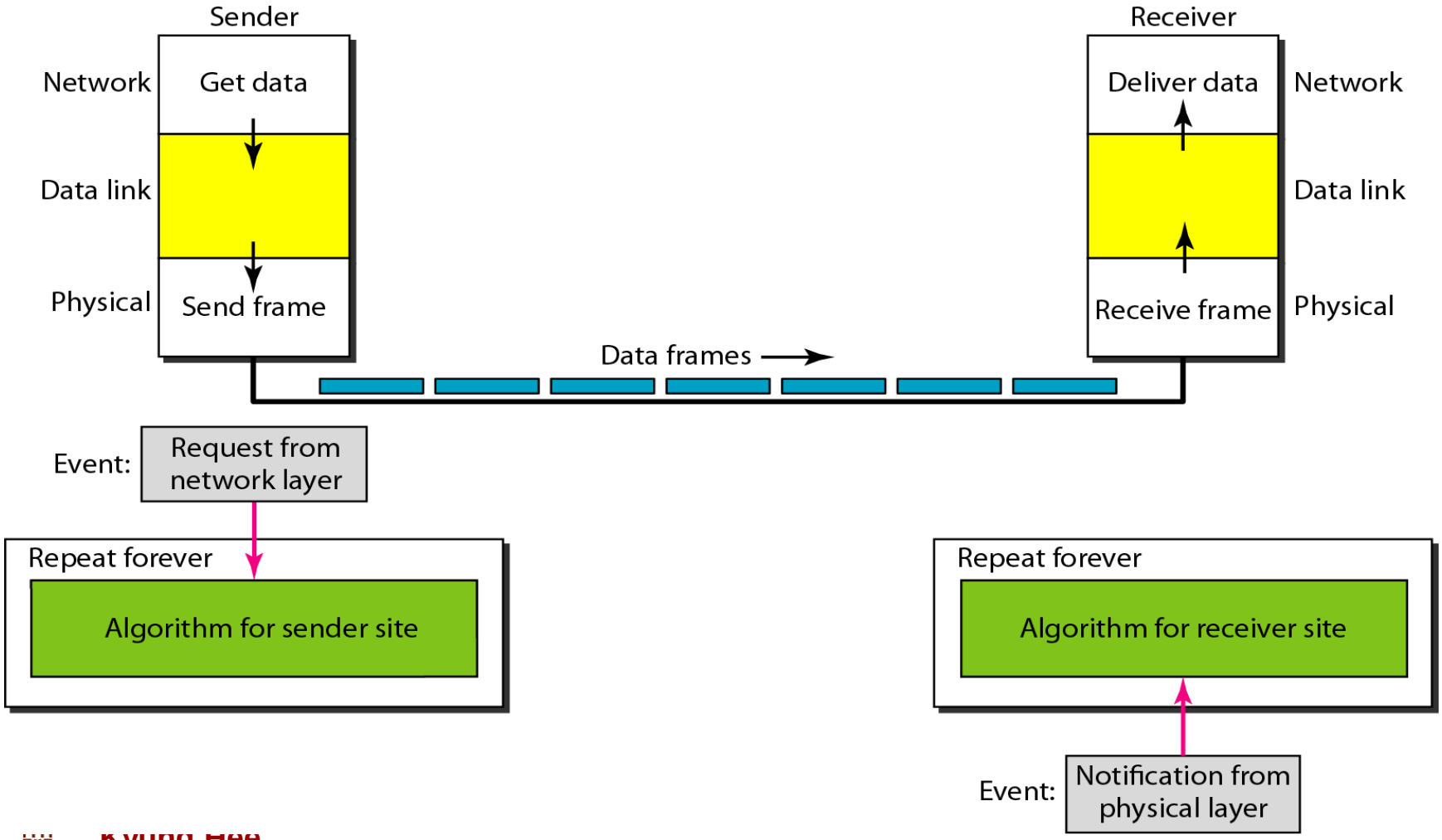
- Simplest Protocol has no flow and error control.
- This protocol is a unidirectional protocol in which data frames are traveling in only one direction from the sender to receiver.
- We assume that the receiver can immediately handle any frame it receives with a processing time that is small enough to be negligible.
  - In other words, the receiver can never be overwhelmed with incoming frames.

Simplest Protocol

- There is no need for flow control in this scheme.
- The data link layers of the sender and receiver provide transmission services for their network layers and use the services provided by their physical layers.
- The sender sites cannot send a frame until its network layer has data packet to send.
- The receiver sites cannot deliver a data packet to its network layer until a frame arrives.
- If the protocol is implemented as a procedure, we need to introduce the idea of events in the protocol.

Simplest Protocol

Figure 11.6 *The design of the simplest protocol with no flow or error control*



Simplest Protocol

Algorithm 11.1 *Sender-site algorithm for the simplest protocol*

```
 1  while(true)                          // Repeat forever
 2  {
 3    WaitForEvent();                    // Sleep until an event occurs
 4    if(Event(RequestToSend))           //There is a packet to send
 5    {
 6       GetData();
 7       MakeFrame();
 8       SendFrame();                    //Send the frame
 9    }
10  }
```

Simplest Protocol

Algorithm 11.2 *Receiver-site algorithm for the simplest protocol*

```
1  while(true)                          // Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(ArrivalNotification))     //Data frame arrived
5    {
6       ReceiveFrame();
7       ExtractData();
8       DeliverData();                  //Deliver data to network layer
9    }
10 }
```

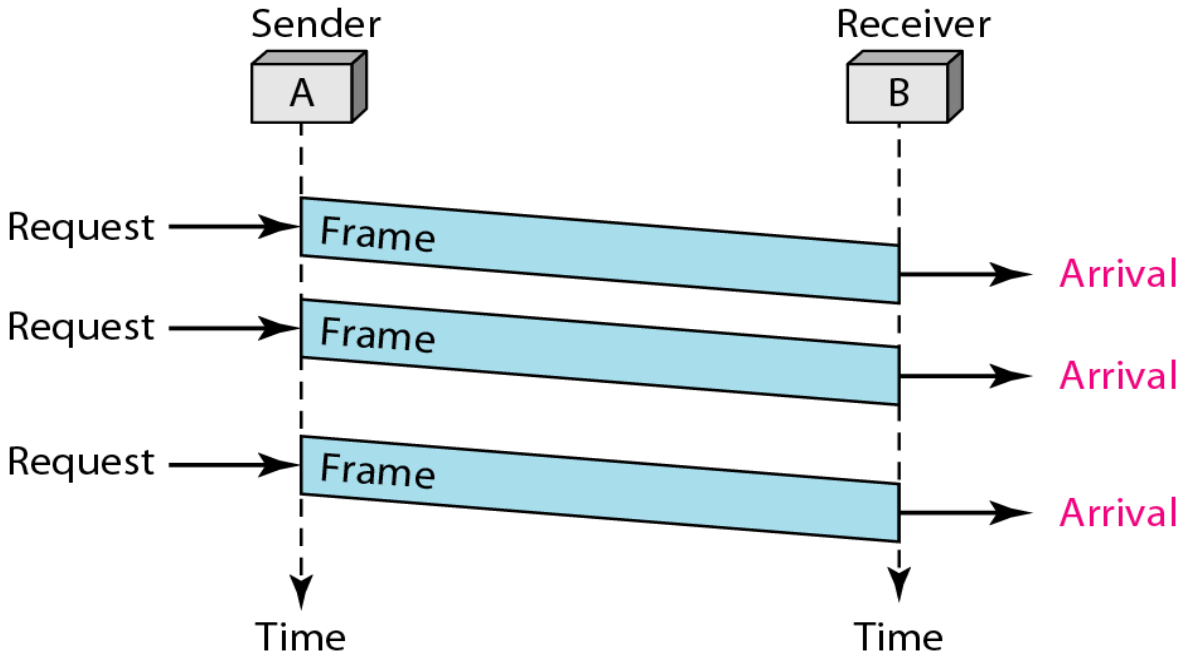
Simplest Protocol

Example 11.1

Figure 11.7 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.

Simplest Protocol

Figure 11.7 *Flow diagram for Example 11.1*



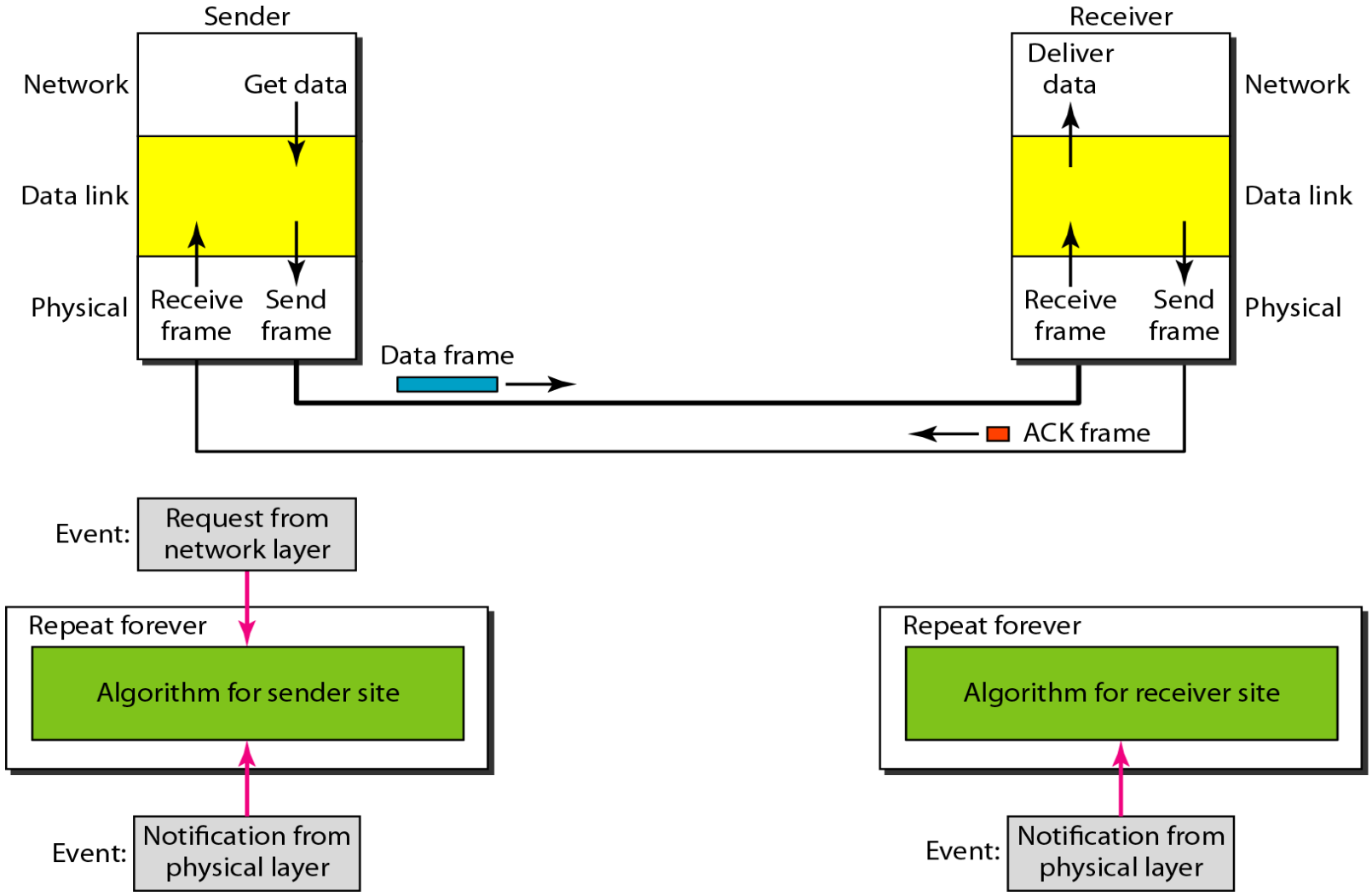
Stop and Wait Protocol

Stop and Wait Protocol

- To prevent the receiver from becoming overwhelmed with frames, we somehow need to tell the sender to slow down.
- We add flow control to the simplest protocol.
- The sender sends a frame and waits for acknowledgment from the receiver before sending the next frame.
- We still have unidirectional communication for data frames, but auxiliary ACK frames travel from the other direction.

Stop and Wait Protocol

Figure 11.8 *Design of Stop-and-Wait Protocol*



Stop and Wait Protocol

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

```
1  while(true)                          //Repeat forever
2  canSend = true                       //Allow the first frame to go
3  {
4    WaitForEvent();                    // Sleep until an event occurs
5    if(Event(RequestToSend) AND canSend)
6    {
7       GetData();
8       MakeFrame();
9       SendFrame();                    //Send the data frame
10      canSend = false;                //Cannot send until ACK arrives
11   }
12   WaitForEvent();                    // Sleep until an event occurs
13   if(Event(ArrivalNotification)      // An ACK has arrived
14    {
15      ReceiveFrame();                 //Receive the ACK frame
16      canSend = true;
17    }
18 }
```

Stop and Wait Protocol

Algorithm 11.4 *Receiver-site algorithm for Stop-and-Wait Protocol*

```
1  while(true)                          //Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(ArrivalNotification))     //Data frame arrives
5    {
6       ReceiveFrame();
7       ExtractData();
8       Deliver(data);                  //Deliver data to network layer
9       SendFrame();                    //Send an ACK frame
10   }
11 }
```

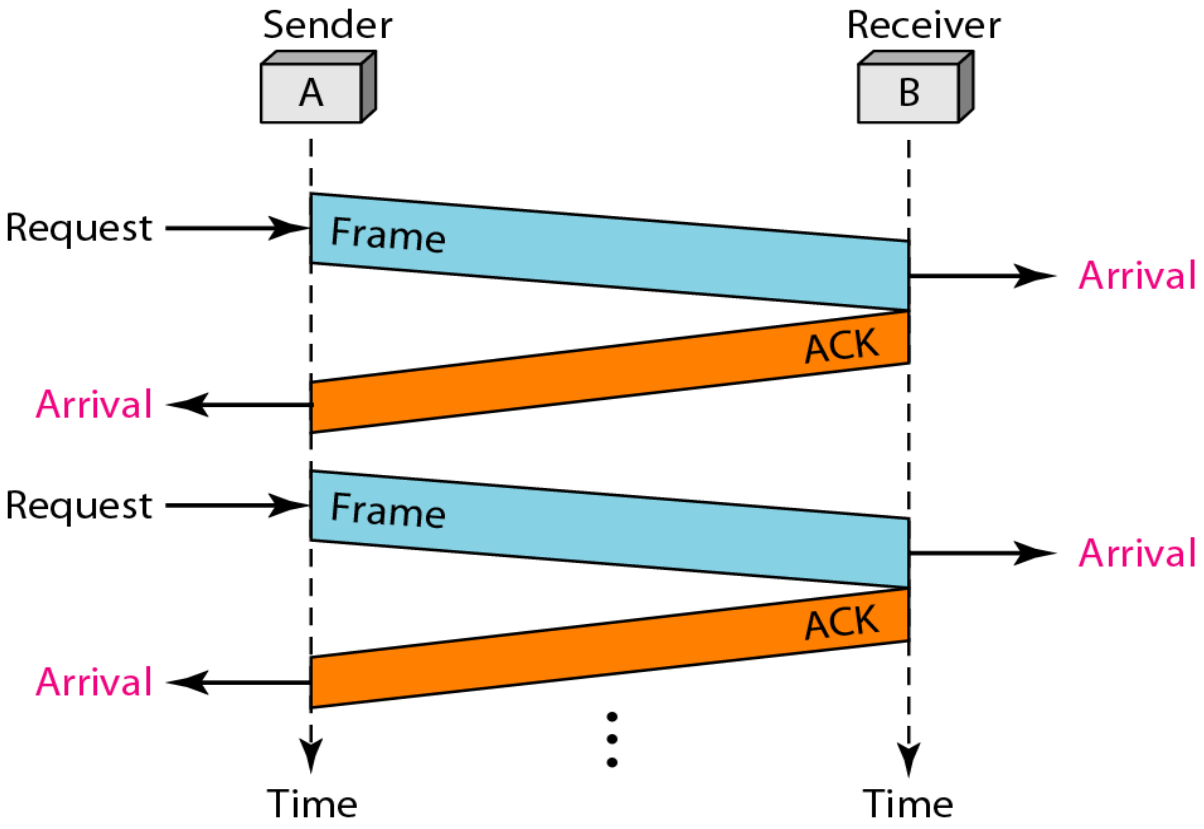
Stop and Wait Protocol

Example 11.2

Figure 11.9 shows an example of communication using this protocol. It is still very simple. The sender sends one frame and waits for feedback from the receiver. When the ACK arrives, the sender sends the next frame. Note that sending two frames in the protocol involves the sender in four events and the receiver in two events.

Stop and Wait Protocol

Figure 11.9 *Flow diagram for Example 11.2*



11.5 NOISY CHANNELS

Although the Stop-and-Wait Protocol gives us an idea of how to add flow control to its predecessor, noiseless channels are nonexistent. We discuss three protocols in this section that use error control.

Topics discussed in this section:

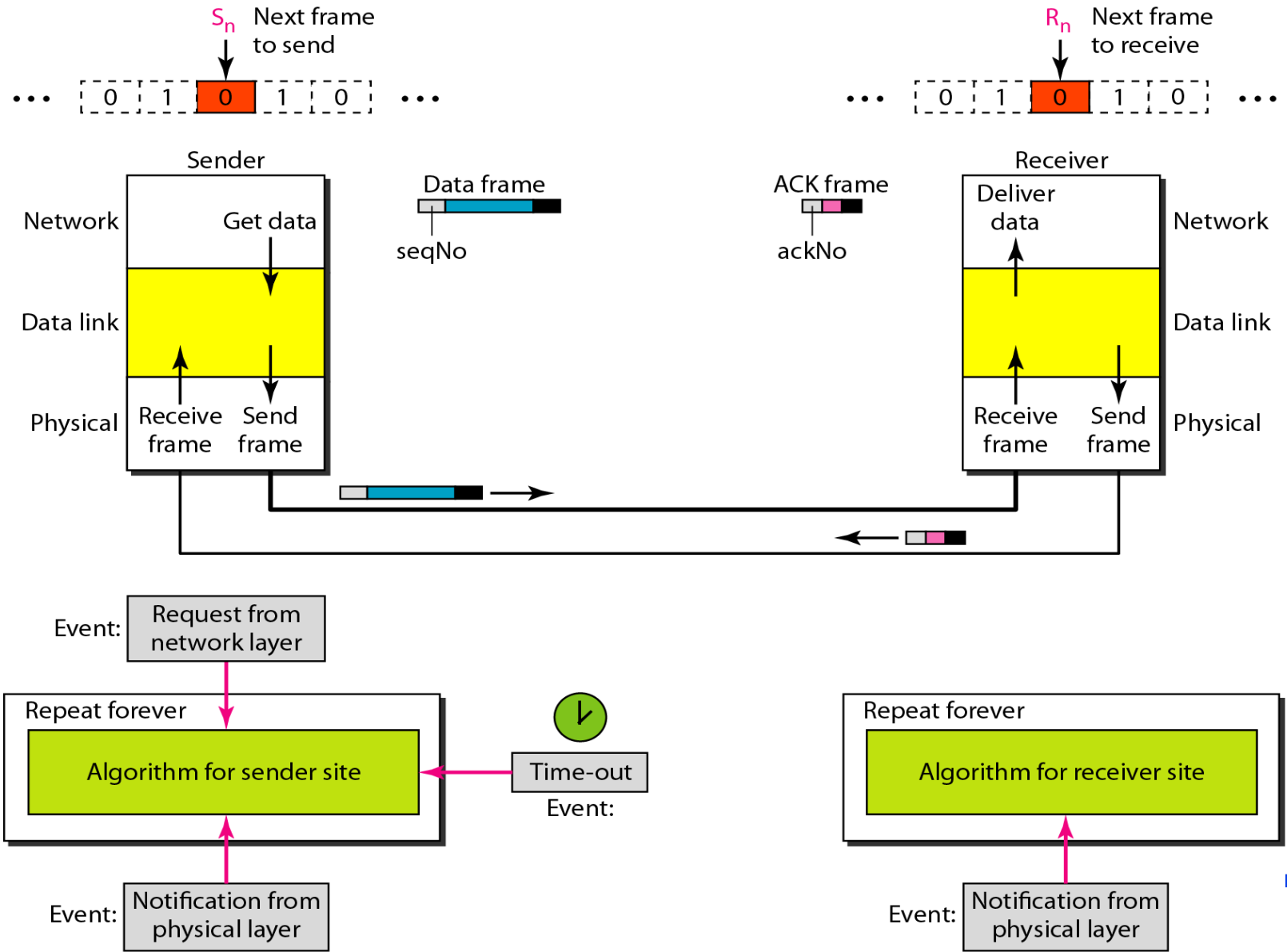
Stop-and-Wait Automatic Repeat Request
Go-Back-N Automatic Repeat Request
Selective Repeat Automatic Repeat Request

Stop and Wait ARQ

- Stop and wait ARQ adds a simple error control mechanism to the Stop and Wait Protocol.
- 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.
  - We use sequence numbers to number the frames. The sequence numbers are based on modulo-2 arithmetic.
    - Sequence Number : 0 ~ 2^m -1, and then repeated
- The frame arrives safe and sound at the receiver; need to be sent an acknowledgment, and the corrupted and lost frames need to be resent.
  - The acknowledgment number always announces in modulo-2 arithmetic the sequence number of the next frame expected. (x + 1)

Stop and Wait ARQ

Figure 11.10 *Design of the Stop-and-Wait ARQ Protocol*



Stop and Wait ARQ

Algorithm 11.5 *Sender-site algorithm for Stop-and-Wait ARQ*

```
1  Sn = 0;                                  // Frame 0 should be sent first
2  canSend = true;                          // Allow the first request to go
3  while(true)                              // Repeat forever
4  {
5    WaitForEvent();                        // Sleep until an event occurs
6    if(Event(RequestToSend) AND canSend)
7    {
8       GetData();
9       MakeFrame(Sn);                      //The seqNo is Sn
10      StoreFrame(Sn);                     //Keep copy
11      SendFrame(Sn);
12      StartTimer();
13      Sn = Sn + 1;
14      canSend = false;
15   }
16   WaitForEvent();                        // Sleep
```

(continued)

Stop and Wait ARQ

Algorithm 11.5 *Sender-site algorithm for Stop-and-Wait ARQ* *(continued)*

```
17      if(Event(ArrivalNotification)          // An ACK has arrived
18      {
19        ReceiveFrame(ackNo);                 //Receive the ACK frame
20        if(not corrupted AND ackNo == Sn)    //Valid ACK
21          {
22            Stoptimer();
23            PurgeFrame(Sn-1);                //Copy is not needed
24            canSend = true;
25          }
26       }
27
28      if(Event(TimeOut)                      // The timer expired
29      {
30       StartTimer();
31       ResendFrame(Sn-1);                    //Resend a copy check
32      }
33  }
```

Stop and Wait ARQ

Algorithm 11.6 *Receiver-site algorithm for Stop-and-Wait ARQ Protocol*

```
1  Rn = 0;                          // Frame 0 expected to arrive first
2  while(true)
3  {
4    WaitForEvent();                // Sleep until an event occurs
5    if(Event(ArrivalNotification))  //Data frame arrives
6    {
7       ReceiveFrame();
8       if(corrupted(frame));
9          sleep();
10      if(seqNo == Rn)              //Valid data frame
11      {
12       ExtractData();
13        DeliverData();             //Deliver data
14        Rn = Rn + 1;
15      }
16       SendFrame(Rn);             //Send an ACK
17   }
18 }
```

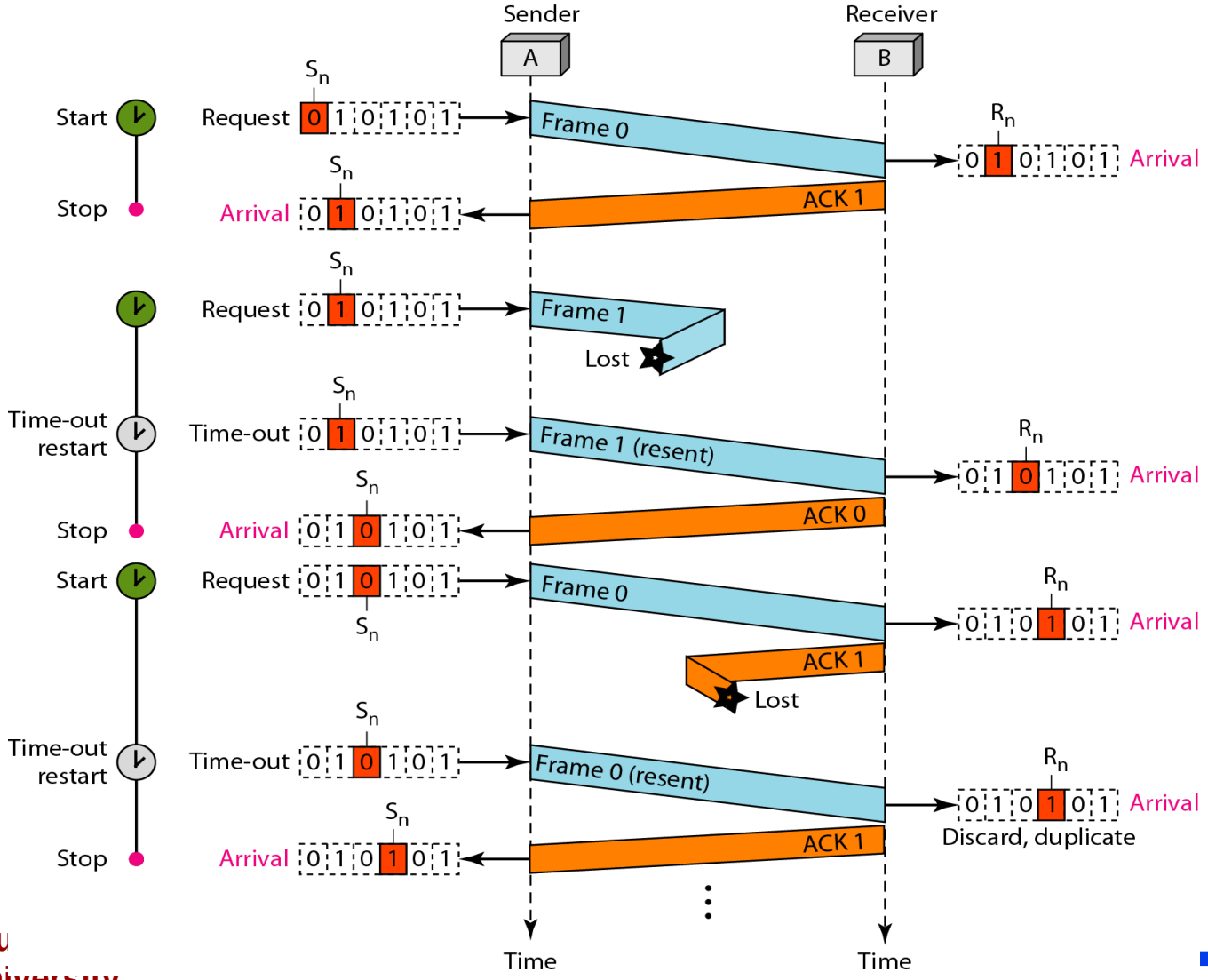
Stop and Wait ARQ

Example 11.3

Figure 11.11 shows an example of Stop-and-Wait ARQ. 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.

Stop and Wait ARQ

Figure 11.11 *Flow diagram for Example 11.3*



Bandwidth-delay product

- **Bandwidth-delay product is a measure of the number of bits we can send out of our system while waiting for news from the receiver.**
  - **Bandwidth-delay product = Bandwidth x delay time**
  - **Utilization % of the link = (frame length / Bandwidth-delay product) x 100**

Stop and Wait ARQ

Example 11.4

Assume that, in a Stop-and-Wait ARQ system, the bandwidth of the line is 1 Mbps, and 1 bit takes 20 ms to make a round trip. What is the bandwidth-delay product? If the system data frames are 1000 bits in length, what is the utilization percentage of the link?

Solution

The bandwidth-delay product is

$$(1 \times 10^6) \times (20 \times 10^{-3}) = 20{,}000 \text{ bits}$$

Stop and Wait ARQ

Example 11.4 (continued)

The system can send 20,000 bits during the time it takes for the data to go from the sender to the receiver and then back again. However, the system sends only 1000 bits. We can say that the link utilization is only 1000/20,000, or 5 percent. For this reason, for a link with a high bandwidth or long delay, the use of Stop-and-Wait ARQ wastes the capacity of the link.

Stop and Wait ARQ

Example 11.5

What is the utilization percentage of the link in Example 11.4 if we have a protocol that can send up to 15 frames before stopping and worrying about the acknowledgments?

Solution

The bandwidth-delay product is still 20,000 bits. The system can send up to 15 frames or 15,000 bits during a round trip. This means the utilization is 15,000/20,000, or 75 percent. Of course, if there are damaged frames, the utilization percentage is much less because frames have to be resent.

pipelining

- In networking and in other areas, a task is often begun before the previous task has ended. - This is known as Pipelining.
  - There is no pipelining in stop-and-wait ARQ
  - Pipelining does apply to Go-Back-N ARQ and Selective Repeat ARQ
- Pipelining improves the efficiency of the transmission if the number of bits in transition is large with respect to the bandwidth delay product.

Go-Back-N ARQ

- **In Go-Back-N ARQ, we can send several frames before receiving acknowledgement; we keep a copy of these frames until the acknowledgments arrive.**

- **Sequence Numbers**
  - **Frames from a sending station are numbered sequentially.**
  - **m-bits for the sequence number : range repeat from 0 to 2^m-1.**
    - **0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15, 0,1,2,3,4,5,6,7,8,9,10,11,…..**

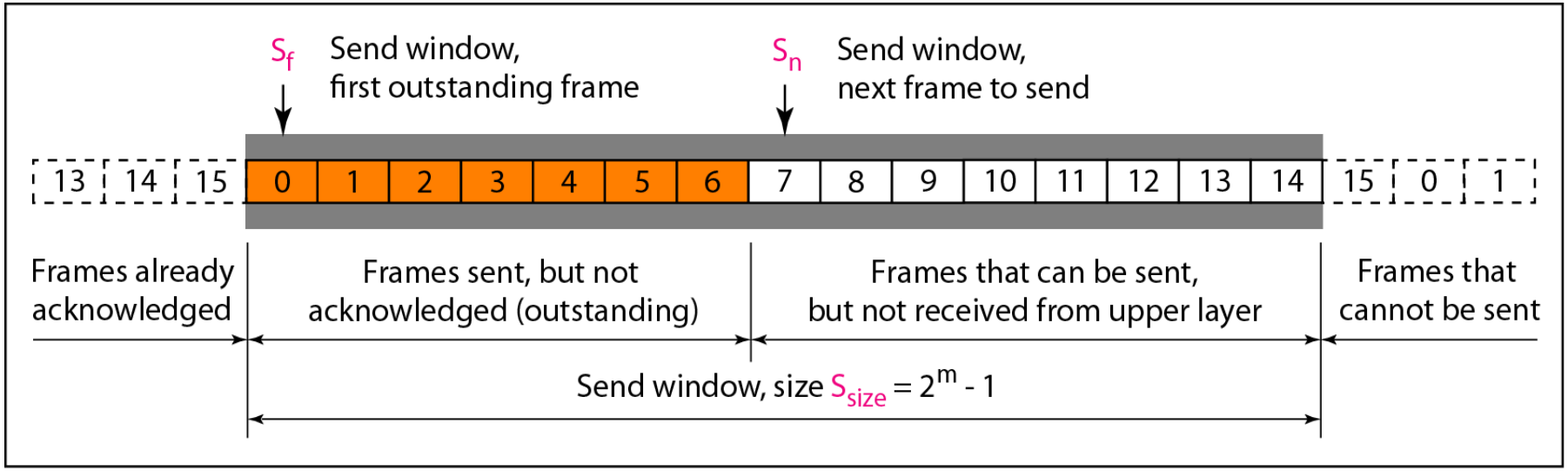
modulo-2^m

Go-Back-N ARQ

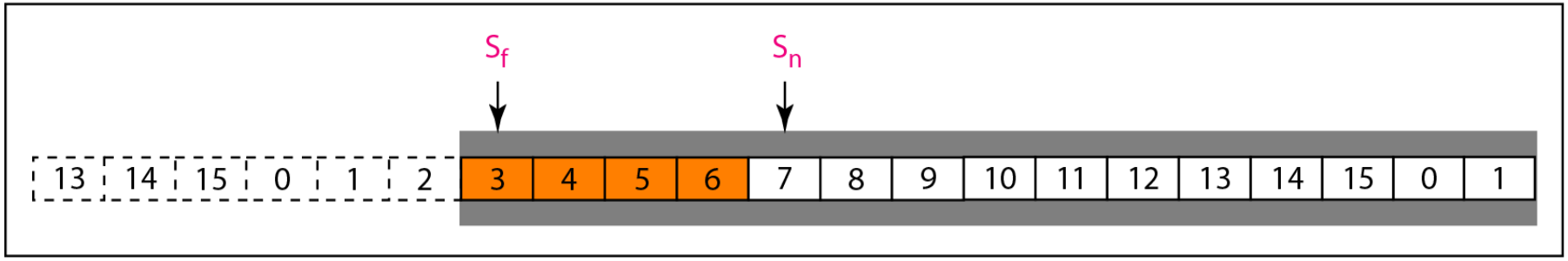
- **Sliding window**
  - is an abstract concept that defines the range of sequence numbers that is the concern of the sender and receiver.
- **Send Sliding Window**
  - The send window is an abstract concept defining an imaginary box of size $2^m - 1$ with three variables: S_f, S_n, and S_{size}.
  - The send window can slide one or more slots when a valid acknowledgment arrives.

Go-Back-N ARQ

Figure 11.12 *Send window for Go-Back-N ARQ*



a. Send window before sliding



b. Send window after sliding

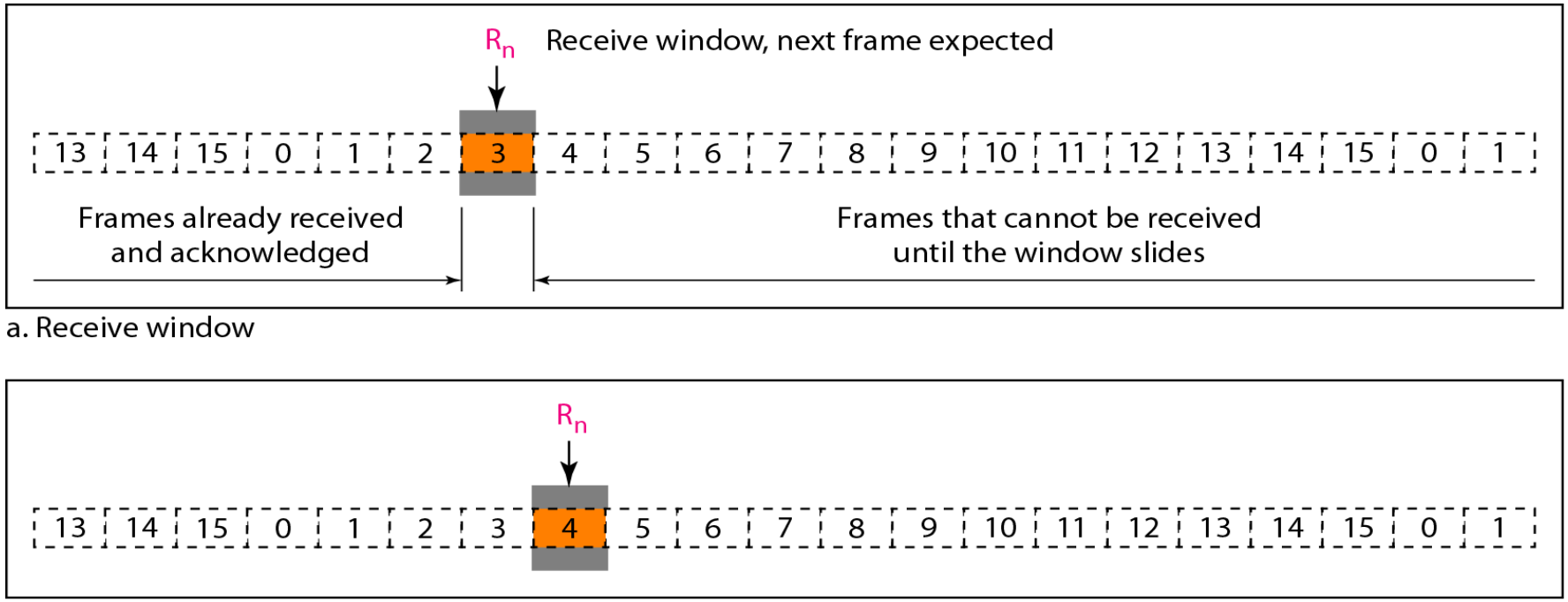
Go-Back-N ARQ

Receive Sliding Window

- 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.
- Any frame arriving out of order is discarded and needs to be resent.

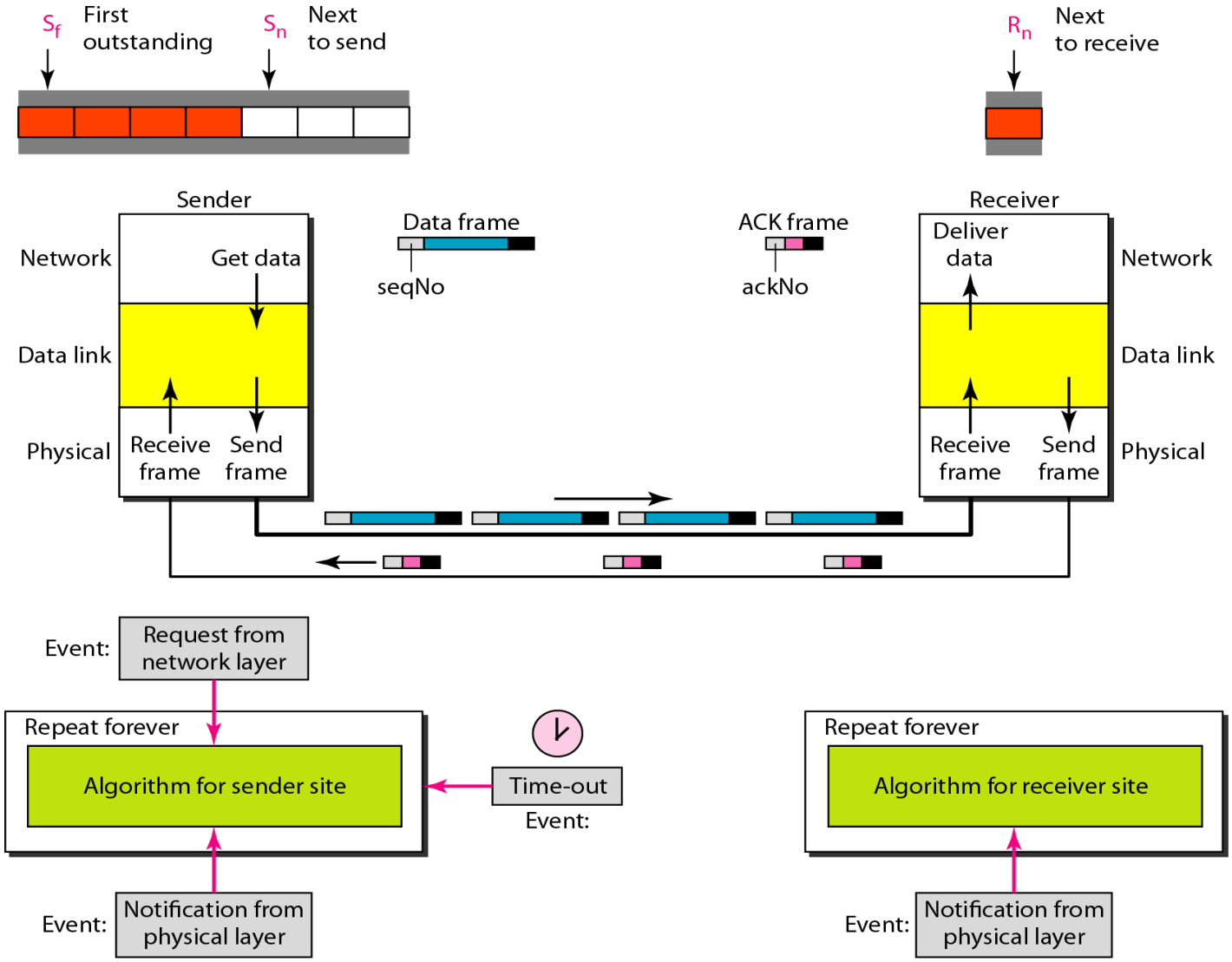
Go-Back-N ARQ

Figure 11.13 *Receive window for Go-Back-N ARQ*



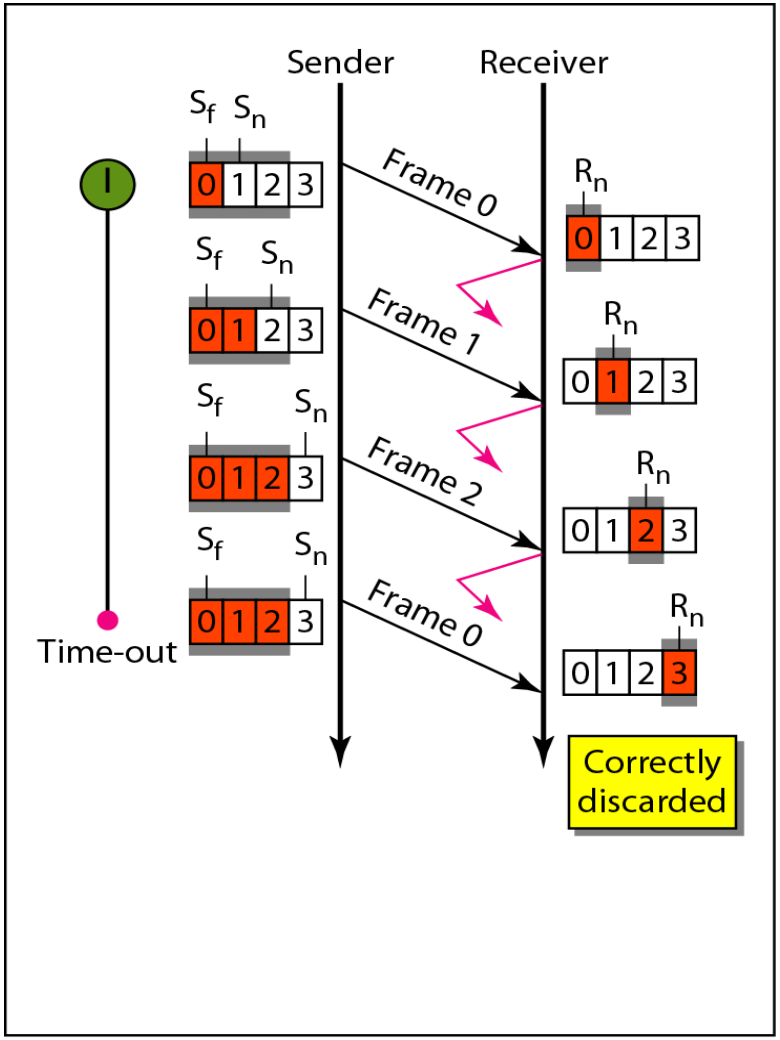
Go-Back-N ARQ

Figure 11.14 *Design of Go-Back-N ARQ*

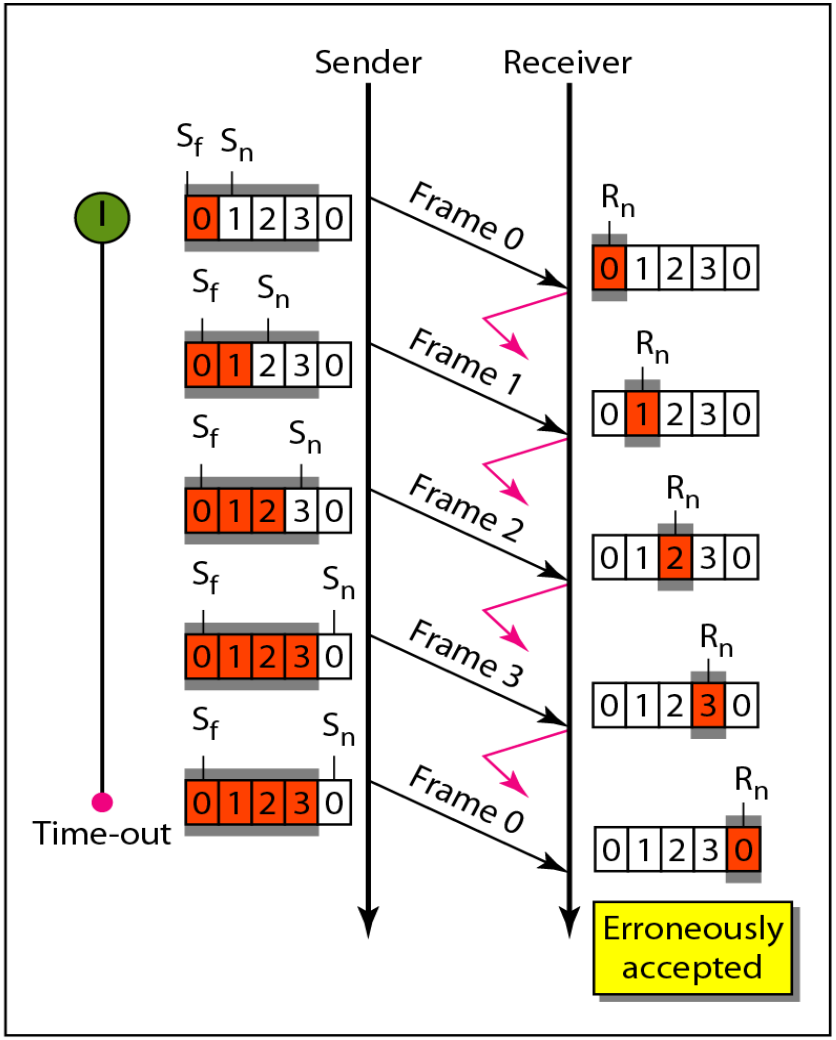


Go-Back-N ARQ

Figure 11.15 *Window size for Go-Back-N ARQ*



a. Window size $< 2^m$



b. Window size $= 2^m$

Go-Back-N ARQ

Note

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.

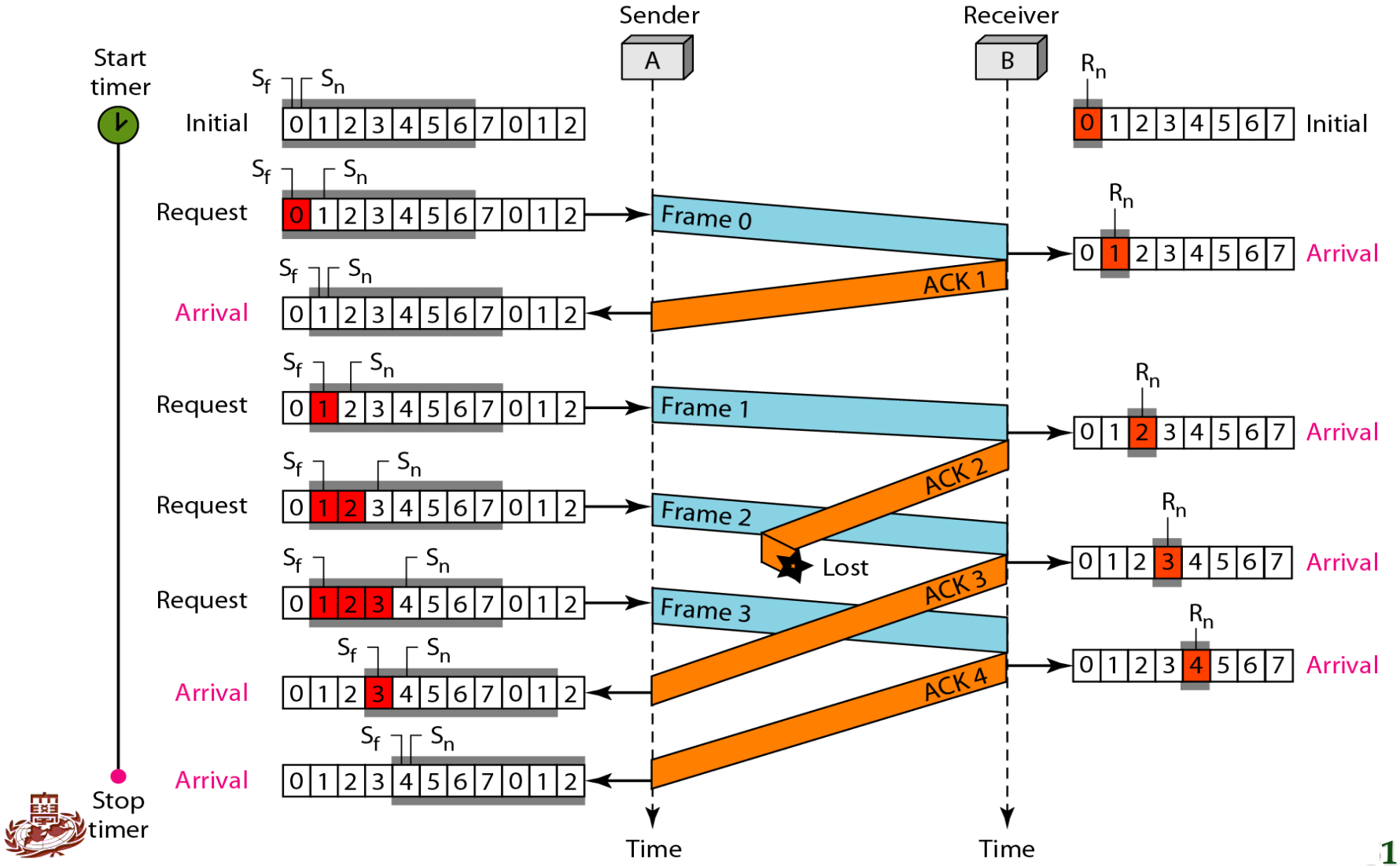
Go-Back-N ARQ

Example 11.6

Figure 11.16 shows an example of Go-Back-N. 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.

Go-Back-N ARQ

Figure 11.16 *Flow diagram for Example 11.6*



Go-Back-N ARQ

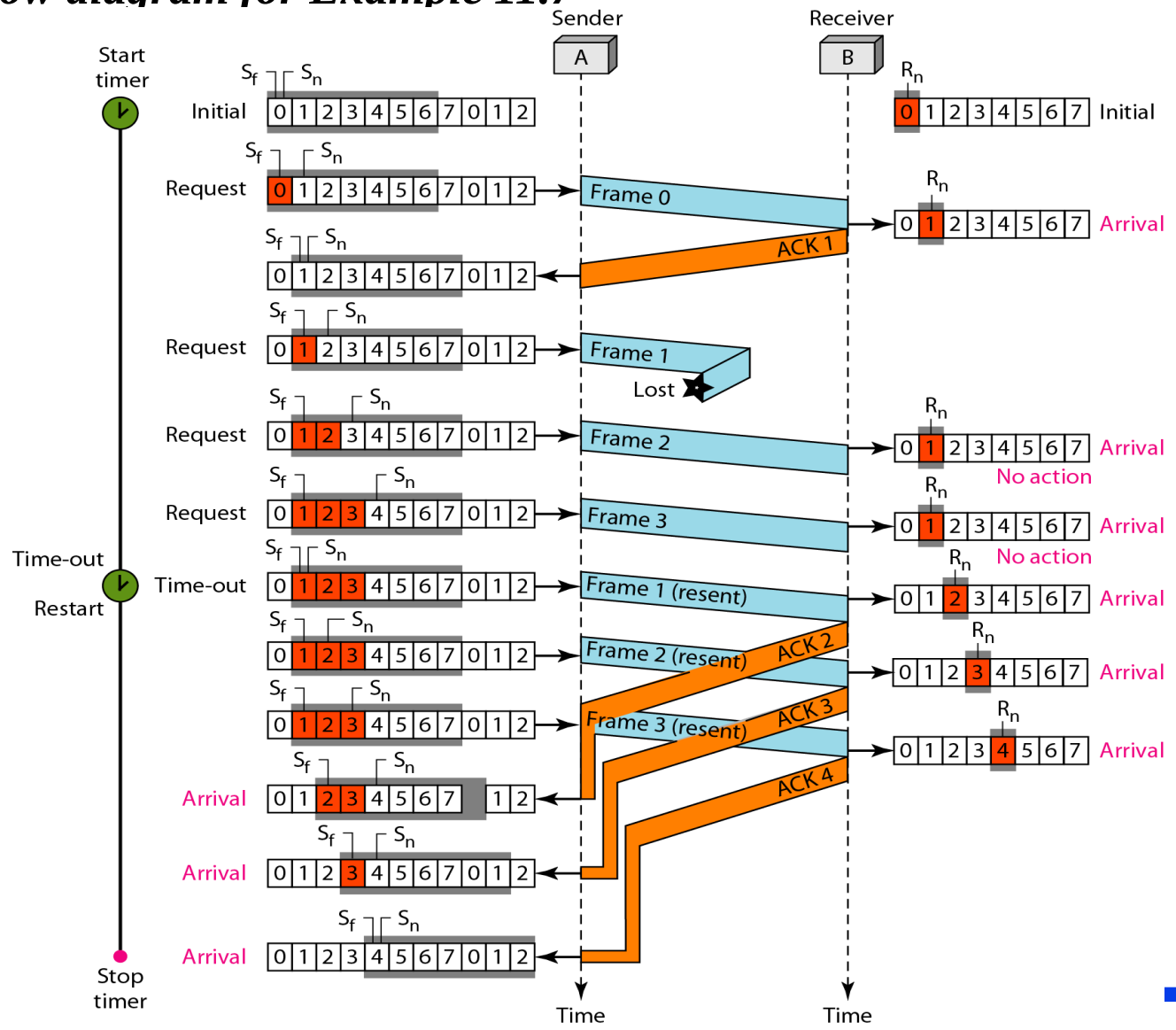
Example 11.7

Figure 11.17 shows 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.

The physical layer must wait until this event is completed and the data link layer goes back to its sleeping state. We have shown a vertical line to indicate the delay. It is the same story with ACK 3; but when ACK 3 arrives, the sender is busy responding to ACK 2. It happens again when ACK 4 arrives. Note that before the second timer expires, all outstanding frames have been sent and the timer is stopped.

Go-Back-N ARQ

Figure 11.17 *Flow diagram for Example 11.7*



Go-Back-N ARQ

Note

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

Selective-Repeat ARQ

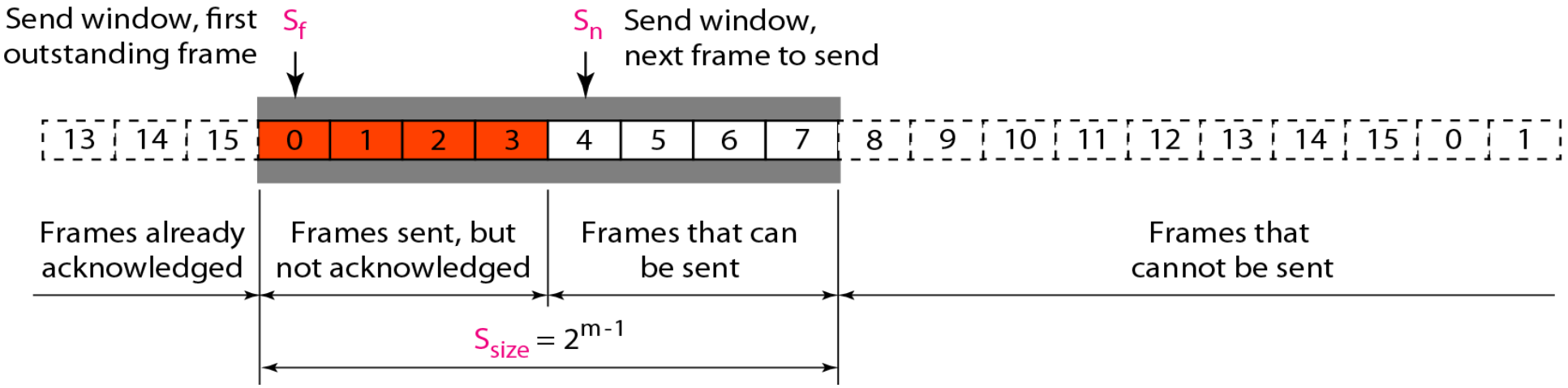
- Go-Back-N ARQ is very inefficient for a noisy link
- For noisy links, Selective Repeat ARQ does not resend N frames when just one frame is damaged; only the damaged frame is resent.
- It is more efficient for noisy links, but the processing at the receiver is more complex.

Selective-Repeat ARQ

- The selective Repeat Protocol also uses two windows;
  - a send window and receive window.
  - The size of send and receive window are the same and much smaller; it is 2^{m-1}
  - The Selective Repeat Protocol allows as many frames as the size of the receive window to arrive out of order and be kept until there is a set of in-order frames to be delivered to the network layer.
  - NAK; negative acknowledgement
    - If a valid NAK frame arrives, a sender resends the corrresponding frame

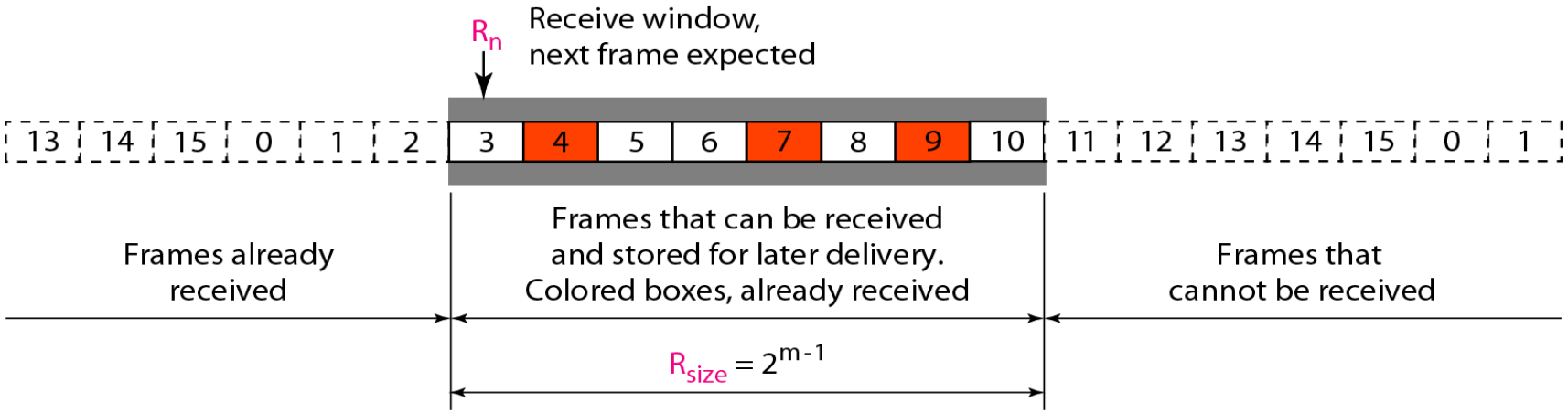
Selective-Repeat ARQ

Figure 11.18 *Send window for Selective Repeat ARQ*



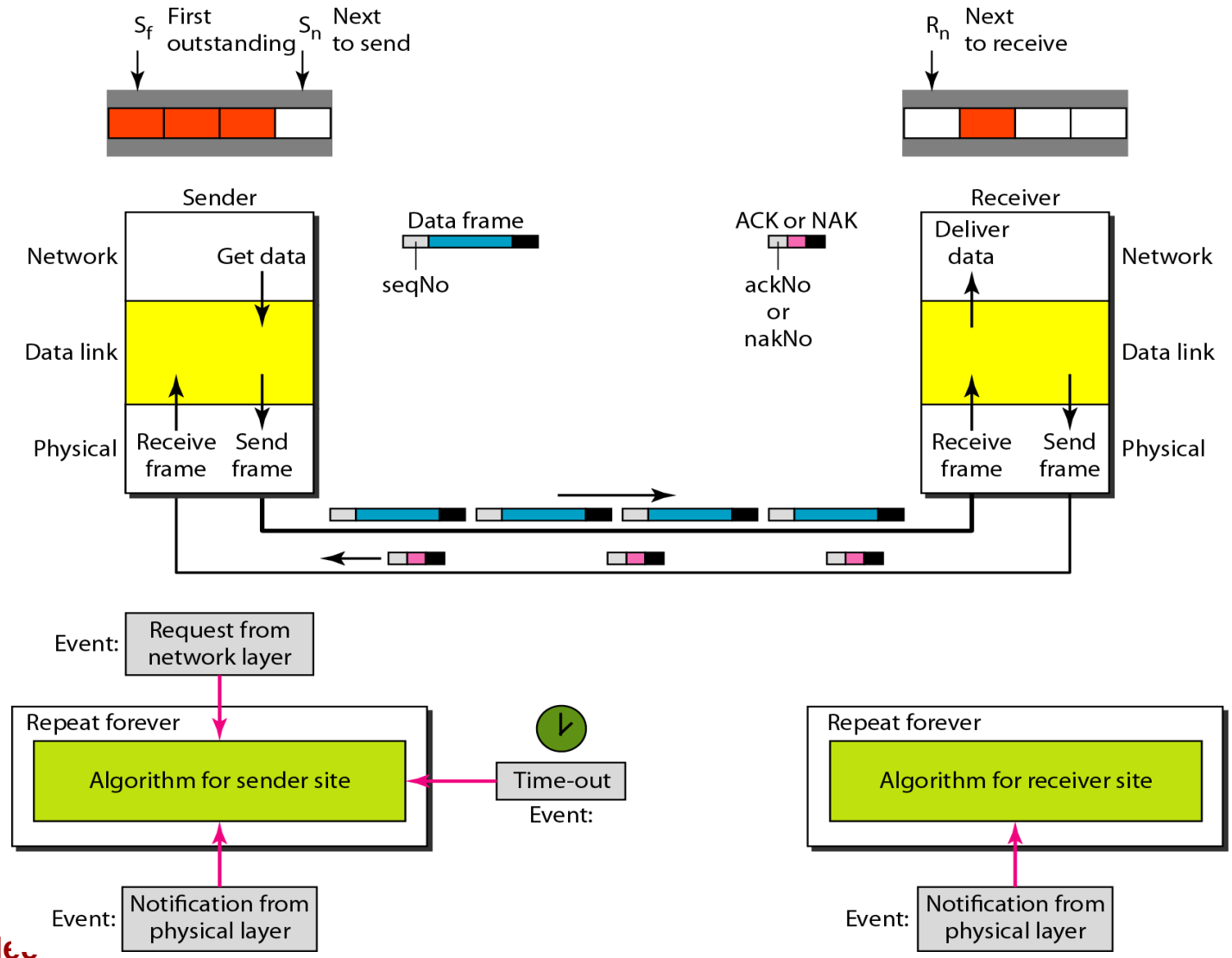
Selective-Repeat ARQ

Figure 11.19 *Receive window for Selective Repeat ARQ*



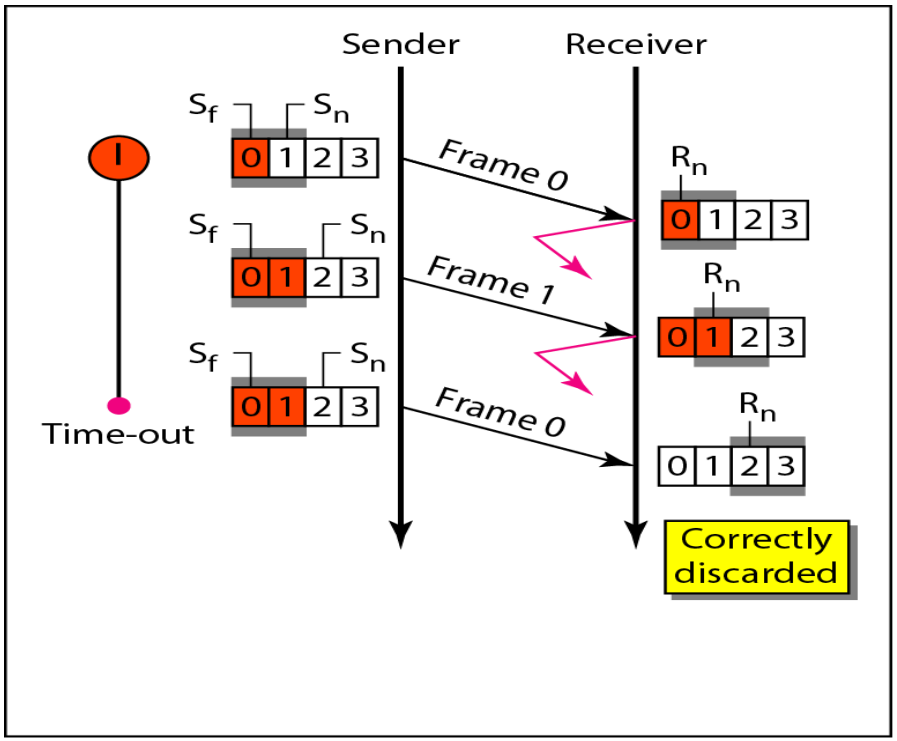
Selective-Repeat ARQ

Figure 11.20 *Design of Selective Repeat ARQ*

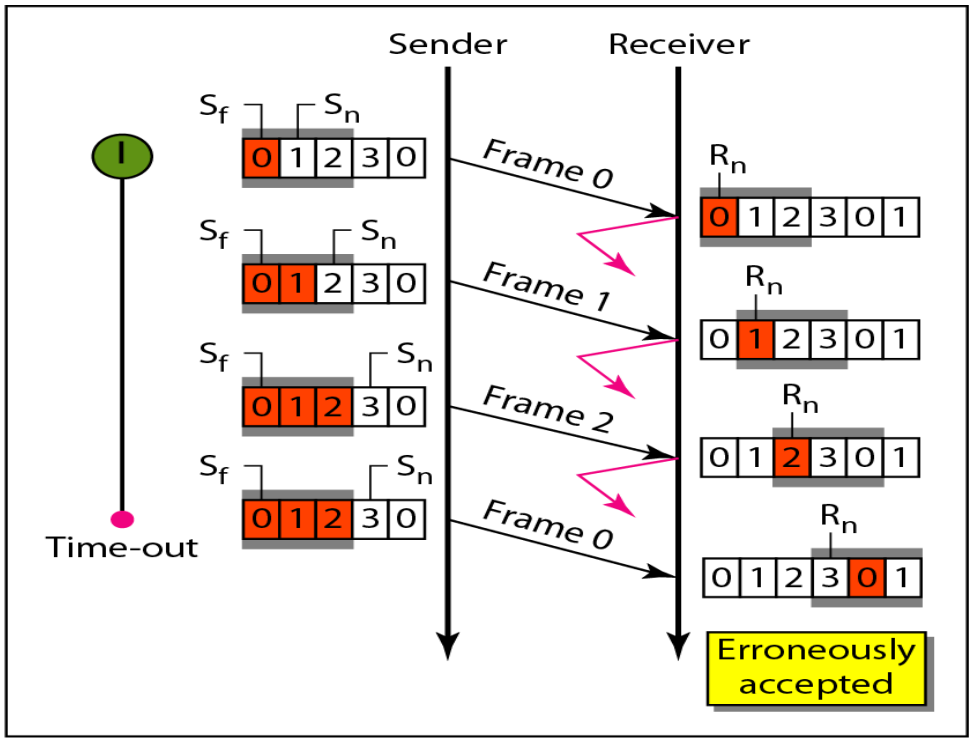


Selective-Repeat ARQ

Figure 11.21 *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.

Selective-Repeat ARQ

Algorithm 11.9 *Sender-site Selective Repeat algorithm*

```
1  Sw = 2^(m-1) ;
2  Sf = 0;
3  Sn = 0;
4
5  while (true)                          //Repeat forever
6  {
7    WaitForEvent();
8    if(Event(RequestToSend))            //There is a packet to send
9    {
10     if(Sn-Sf >= Sw)                   //If window is full
11           Sleep();
12     GetData();
13     MakeFrame(Sn);
14     StoreFrame(Sn);
15     SendFrame(Sn);
16     Sn = Sn + 1;
17     StartTimer(Sn);
18   }
19
```

(continued)

Selective-Repeat ARQ

Algorithm 11.9 *Sender-site Selective Repeat algorithm* (continued)

```
20  if(Event(ArrivalNotification)) //ACK arrives
21  {
22      Receive(frame);                //Receive ACK or NAK
23      if(corrupted(frame))
24          Sleep();
25      if (FrameType == NAK)
26          if (nakNo between Sf and Sn)
27          {
28           resend(nakNo);
29           StartTimer(nakNo);
30          }
31      if (FrameType == ACK)
32          if (ackNo between Sf and Sn)
33          {
34            while(sf < ackNo)
35            {
36             Purge(sf);
37             StopTimer(sf);
38             Sf = Sf + 1;
39            }
40          }
41  }
```

(continued)

Selective-Repeat ARQ

Algorithm 11.9 *Sender-site Selective Repeat algorithm* *(continued)*

```
42
43    if(Event(TimeOut(t)))              //The timer expires
44    {
45     StartTimer(t);
46     SendFrame(t);
47    }
48  }
```

Selective-Repeat ARQ

Algorithm 11.10 *Receiver-site Selective Repeat algorithm*

```
 1  Rn = 0;
 2  NakSent = false;
 3  AckNeeded = false;
 4  Repeat(for all slots)
 5      Marked(slot) = false;
 6
 7  while (true)                                  //Repeat forever
 8  {
 9    WaitForEvent();
10
11    if(Event(ArrivalNotification))              /Data frame arrives
12    {
13       Receive(Frame);
14       if(corrupted(Frame))&& (NOT NakSent)
15       {
16        SendNAK(Rn);
17        NakSent = true;
18        Sleep();
19       }
20       if(seqNo <> Rn)&& (NOT NakSent)
21       {
22        SendNAK(Rn);
```

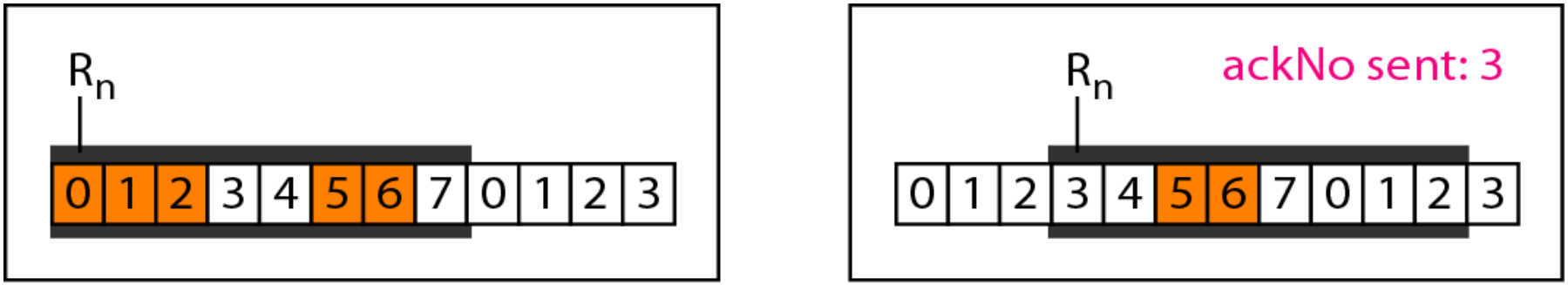
Selective-Repeat ARQ

Algorithm 11.10 *Receiver-site Selective Repeat algorithm*

```
23          NakSent = true;
24          if ((seqNo in window)&&(!Marked(seqNo))
25          {
26           StoreFrame(seqNo)
27           Marked(seqNo)= true;
28           while(Marked(Rn))
29           {
30            DeliverData(Rn);
31            Purge(Rn);
32            Rn = Rn + 1;
33            AckNeeded = true;
34           }
35            if(AckNeeded);
36            {
37            SendAck(Rn);
38            AckNeeded = false;
39            NakSent = false;
40            }
41          }
42        }
43    }
44  }
```

Selective-Repeat ARQ

Figure 11.22 *Delivery of data in Selective Repeat ARQ*



Selective-Repeat ARQ

Example 11.8

This example is similar to Example 11.3 in which frame 1 is lost. We show how Selective Repeat behaves in this case. Figure 11.23 shows the situation. 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.

Selective-Repeat ARQ

Example 11.8 (continued)

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.

Selective-Repeat ARQ

Example 11.8 (continued)

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.

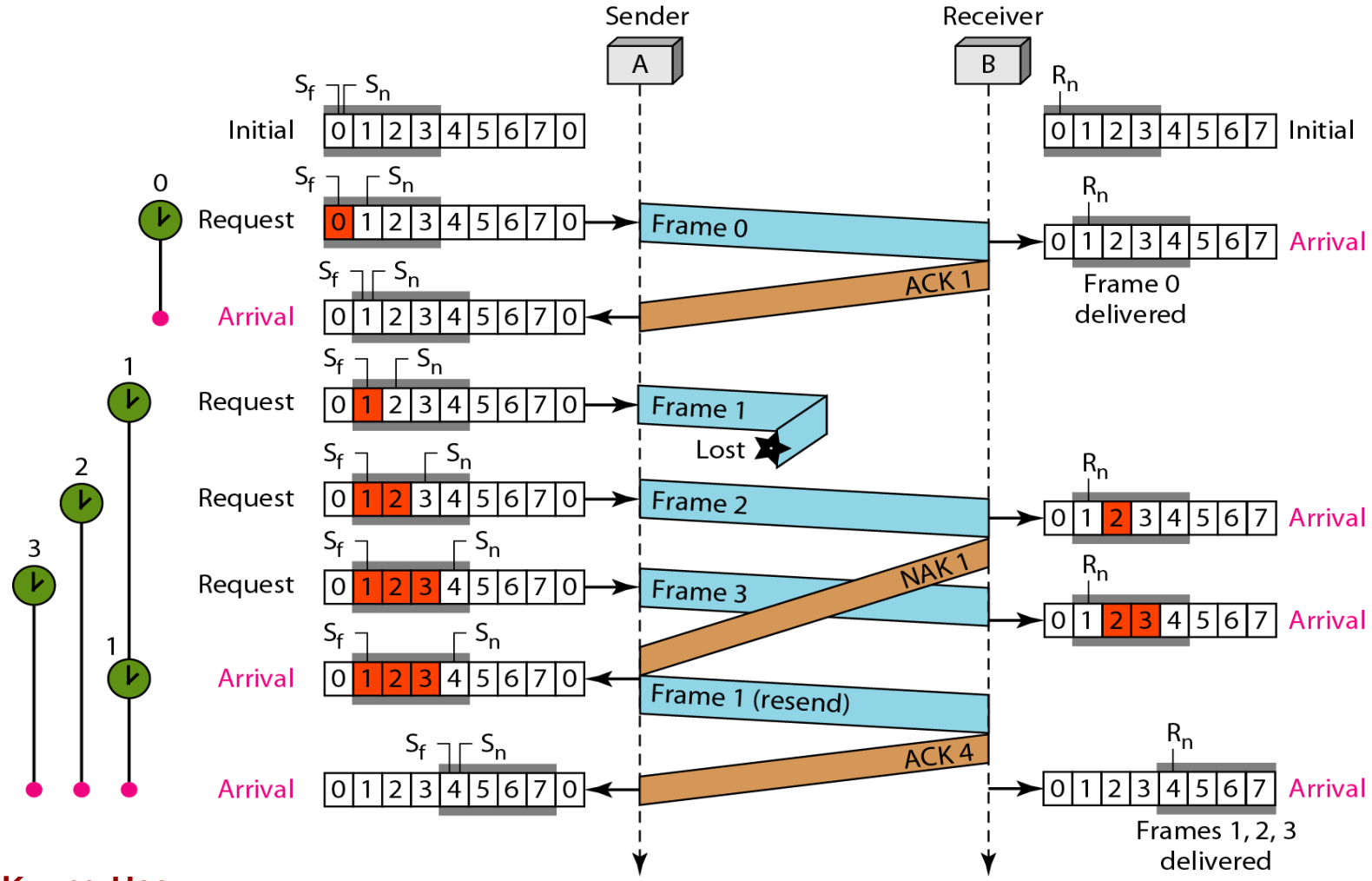
Selective-Repeat ARQ

Example 11.8 (continued)

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.

Selective-Repeat ARQ

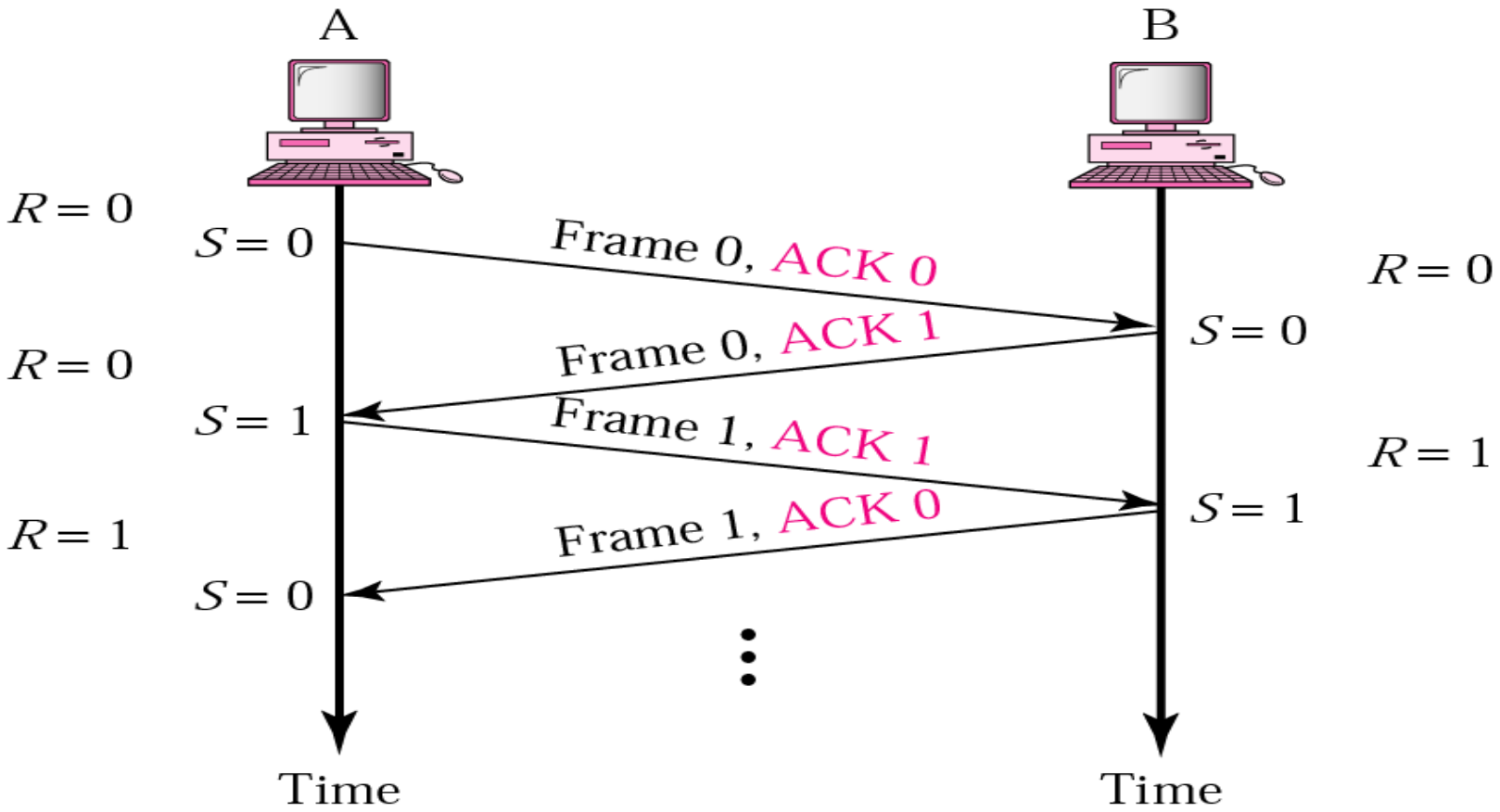
Figure 11.23 *Flow diagram for Example 11.8*



Piggybacking

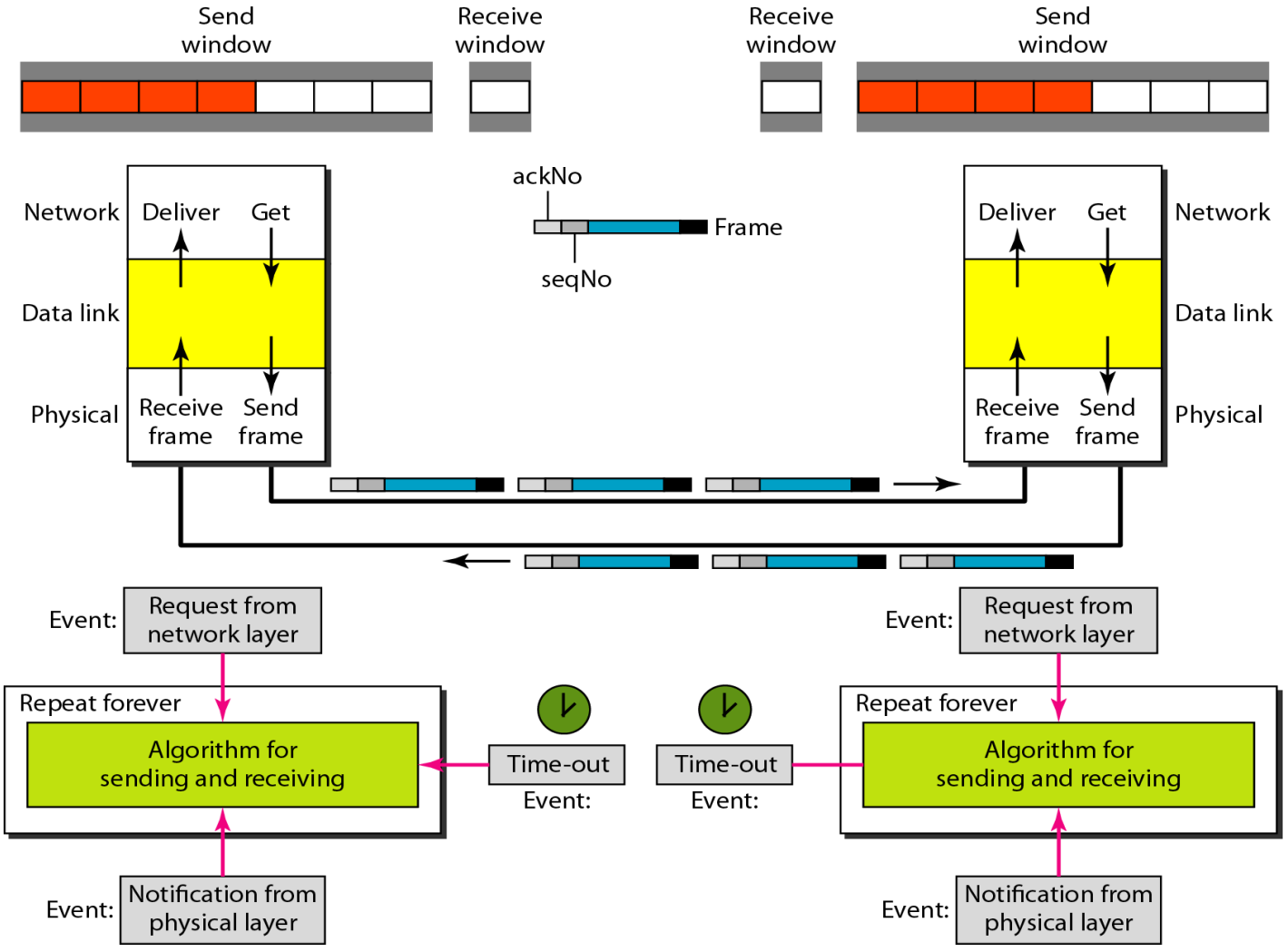
- **Piggybacking**

  meaning combining data to be sent and acknowledgment of the frame received in one single frame



Piggybacking

Figure 11.24 *Design of piggybacking in Go-Back-N ARQ*

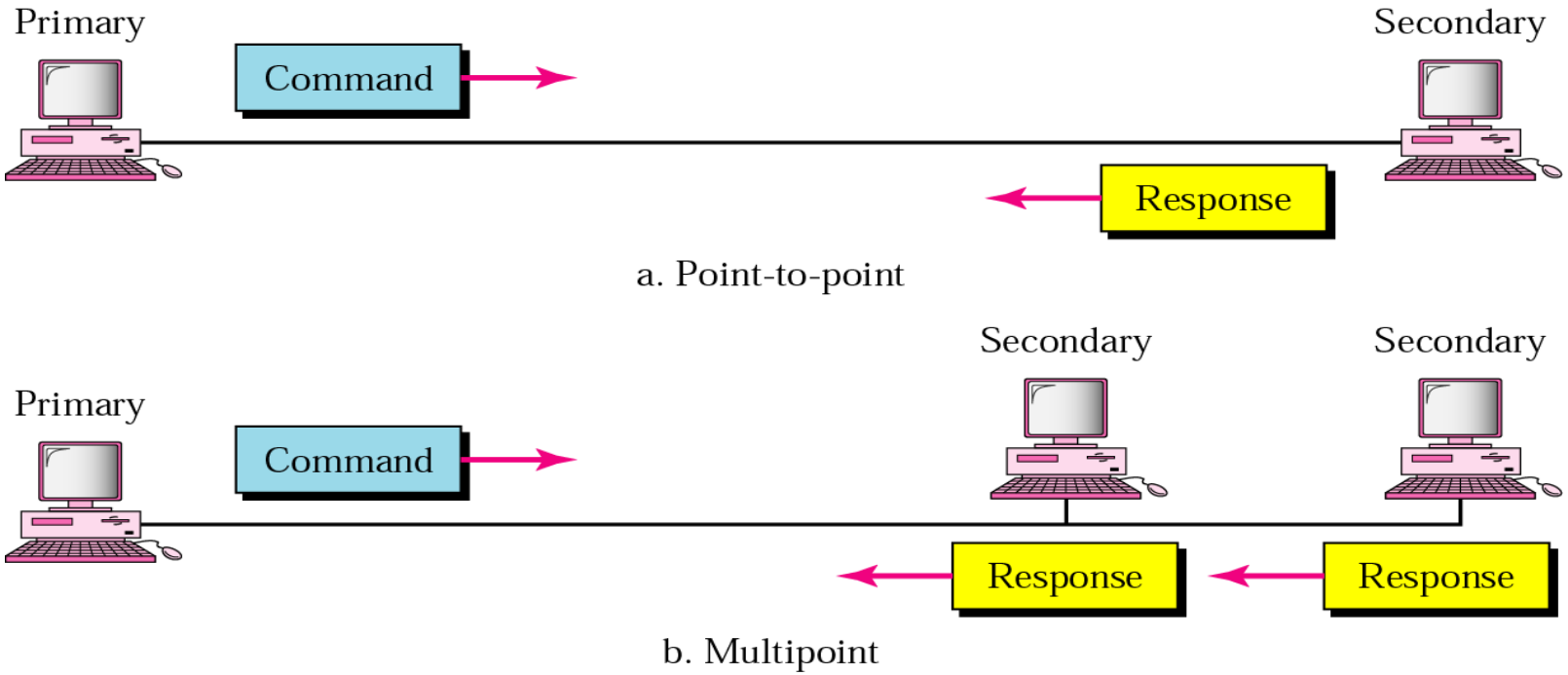


HDLC (High-level Data Link Control)

- A mode in HDLC is the relationship between two devices involved in an exchange; The mode of communication describes who controls the link
- HDLC supports two modes of communication between stations
  - NRM (Normal Response Mode)
  - ABM (Asynchronous Balanced Mode)

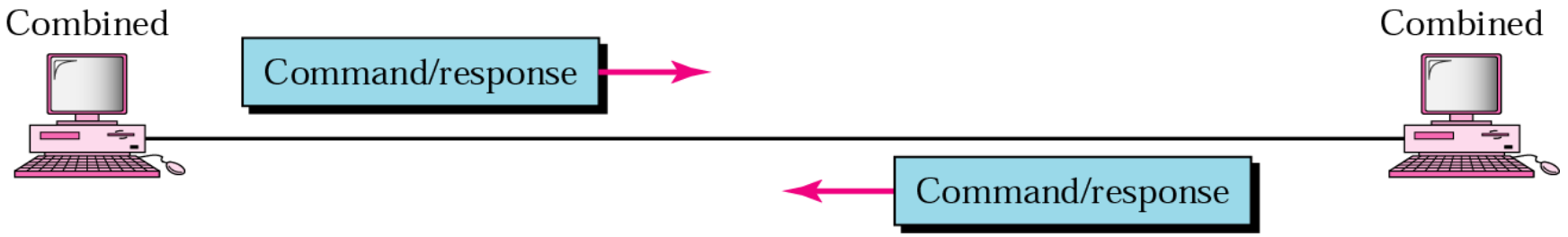
HDLC (cont'd)

- NRM (Normal Response Mode)
  - refers to the standard primary-secondary relationship
  - secondary device must have permission from the primary device before transmitting



HDLC (cont'd)

- ABM (Asynchronous Balanced Mode)
  - all stations are equal and therefore only combined stations connected in point-to-point are used
  - Either combined station may initiate transmission with the other combined station without permission

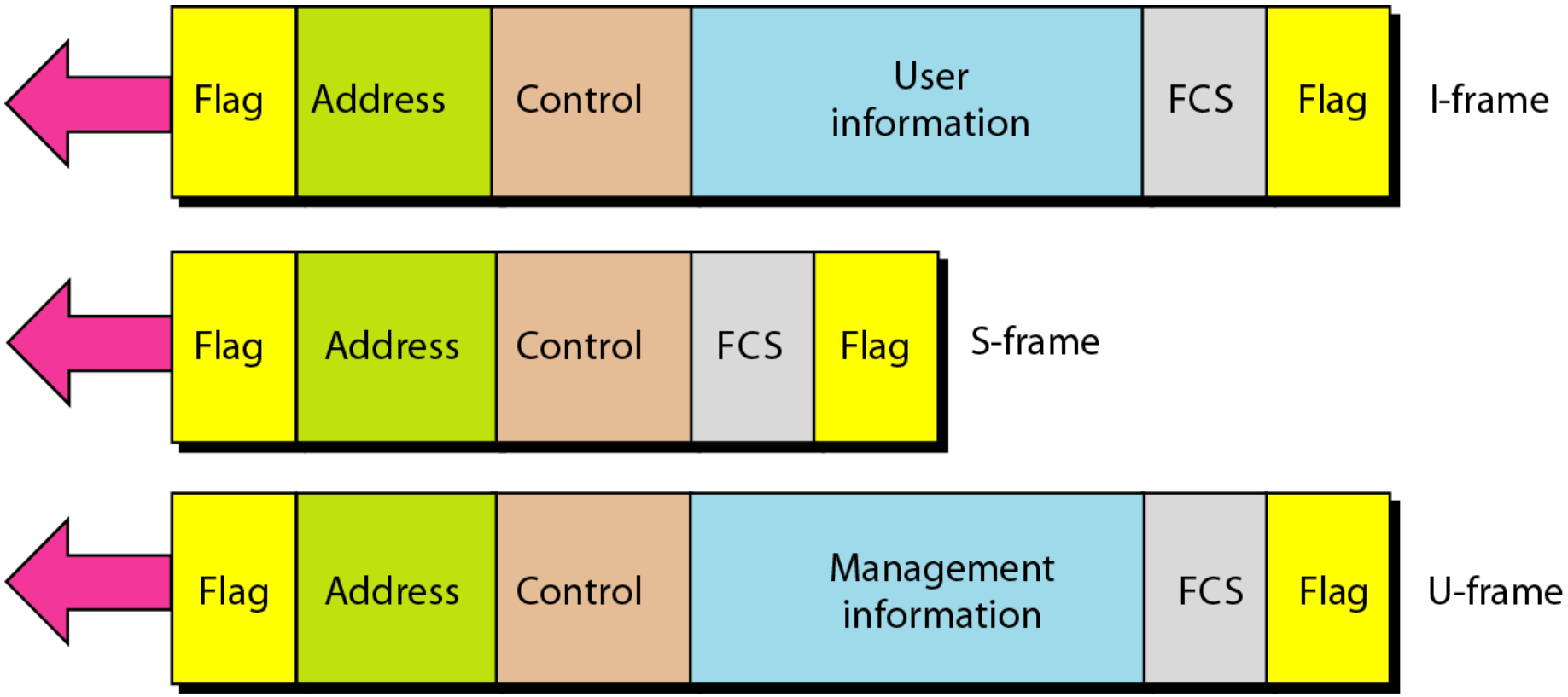


HDLC (cont'd)

- **HDLC Frame Types**
  - **I (Information) Frame**
    - **used to transport user data and control information relating to user data**
  - **S (Supervisory) Frame**
    - **used to only to transport control information, primarily data link layer flow and error controls**
  - **U (Unnumbered) Frame**
    - **is reserved for system management**
    - **Information carried by U-frame is intended for managing the link itself**

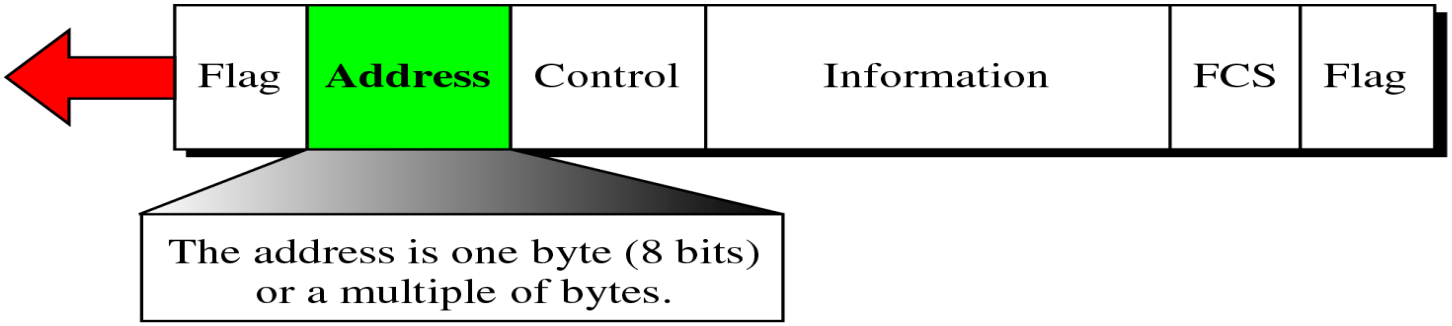
HDLC (cont'd)

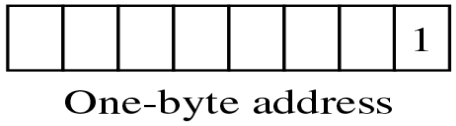
- HDLC frame



HDLC (cont'd)

- **Frame may contain up to six fields**
  - **Flag field**
    - **The flag field is an 8-bit sequence with the bit pattern 01111110 that identifies both the beginning and the end of a frame and serves as a synchronization pattern for the receiver.**
  - **Address Field**
    - **Address field contains the address of the secondary station that is either the originator or destination of the frame.**





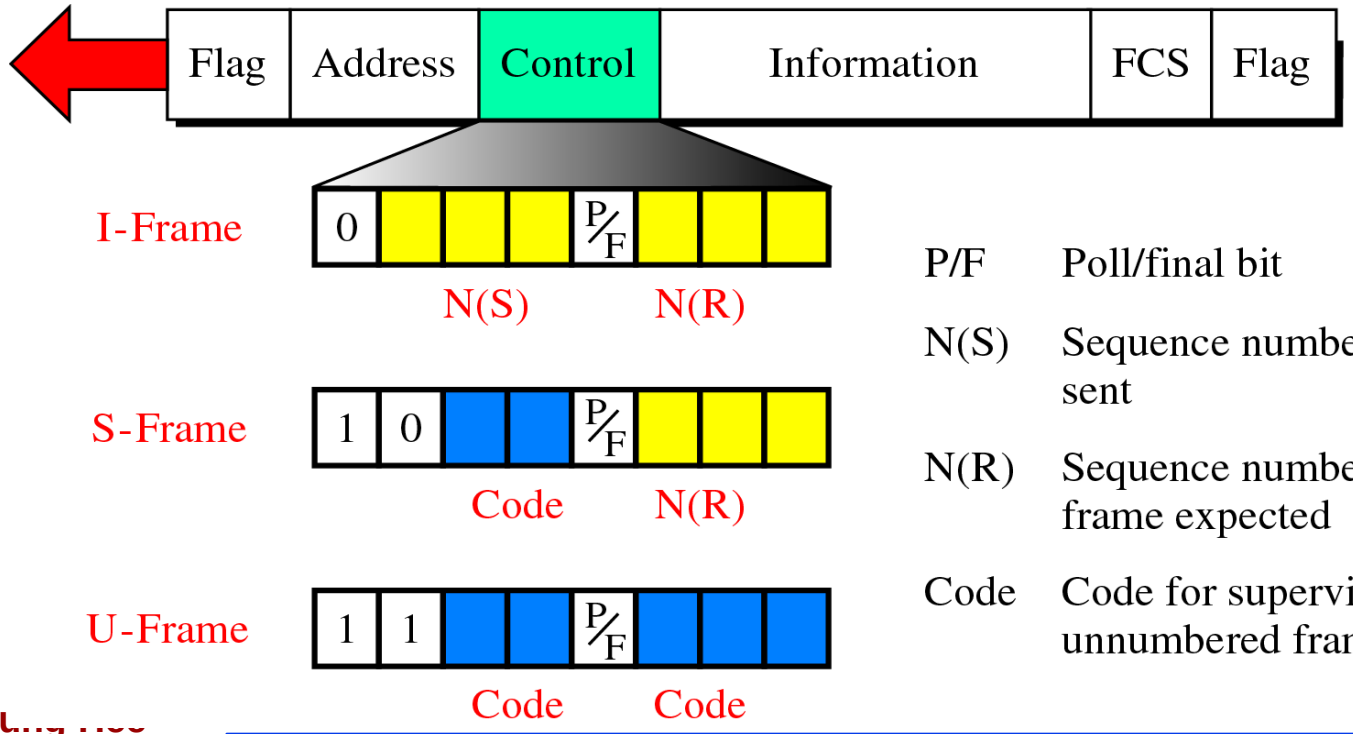
One-byte address

Multibyte address

HDLC (cont'd)

Control Field

- ~ is a 1~2byte segment of the frame used for flow and error control.
- The interpretation of bits in this field depends on the frame type.

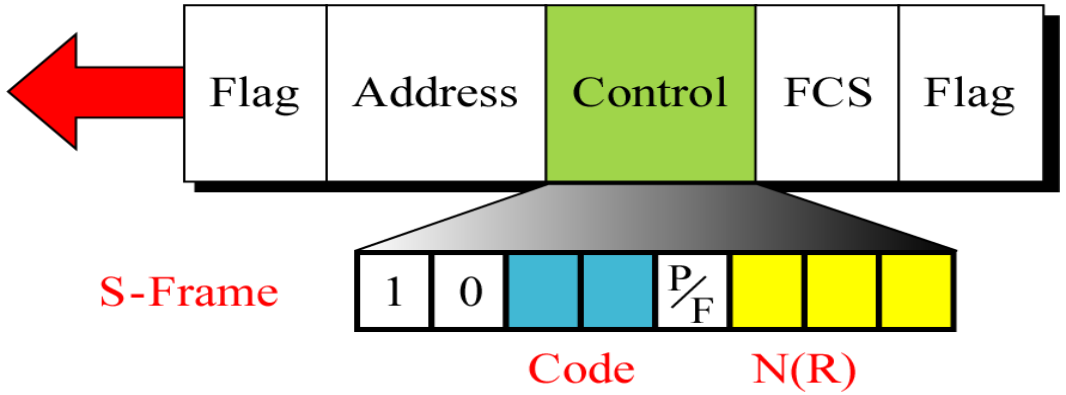


P/F	Poll/final bit
N(S)	Sequence number of frame sent
N(R)	Sequence number of next frame expected
Code	Code for supervisory or unnumbered frame

HDLC(cont'd)

❖ Control Field (S-frame)

- is used for acknowledgment, flow control, and error control

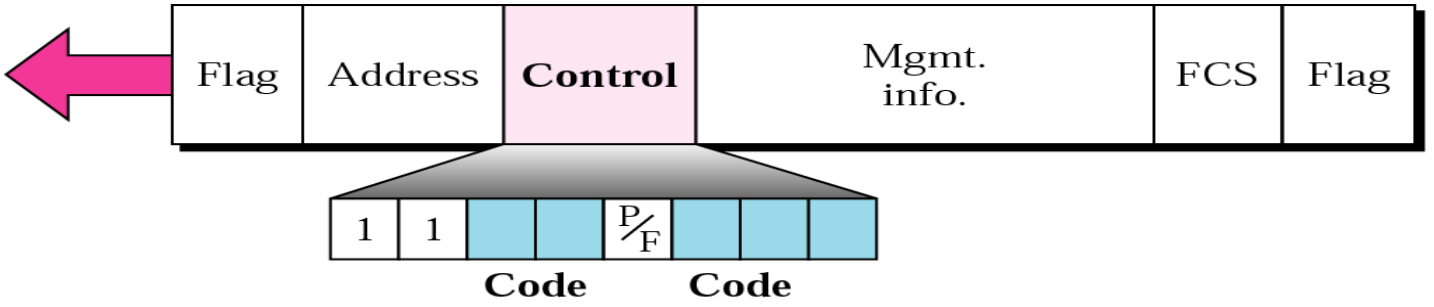


Code	Command	
00	RR	Receive ready
01	REJ	Reject
10	RNR	Receive not ready
11	SREJ	Selective-reject

HDLC (cont'd)

Control Field (U-Frame)

- is used to exchange session management and control information between connected devices

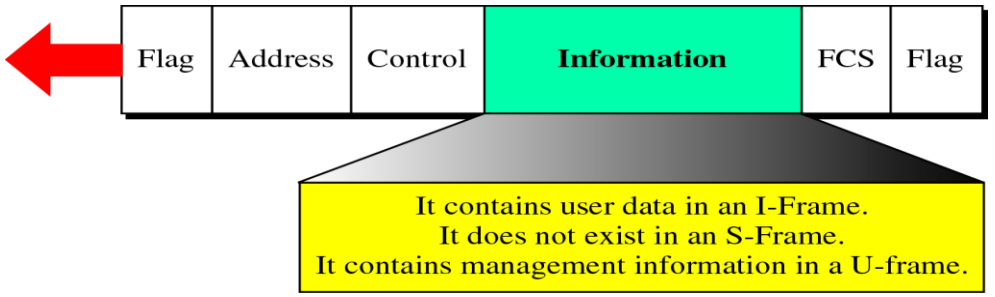


Code	Command	Response
00 001	SNRM	
11 011	SNRME	
11 100	SABM	DM
11 110	SABME	
00 000	UI	UI
00 110		UA
00 010	DISC	RD
10 000	SIM	RIM
00 100	UP	
11 001	RSET	
11 101	XID	XID
10 001		FRMR

HDLC (cont'd)

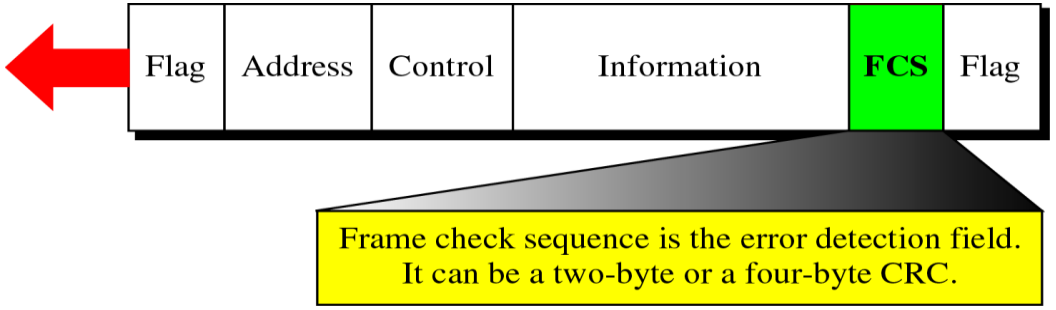
Information Field

- ~ contains the user's data from the network layer or management information.



FCS (Frame Check Sequence) Field

- **The frame check sequence is the HDLC error detection field.**



HDLC (cont'd)

Table 11.1 *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

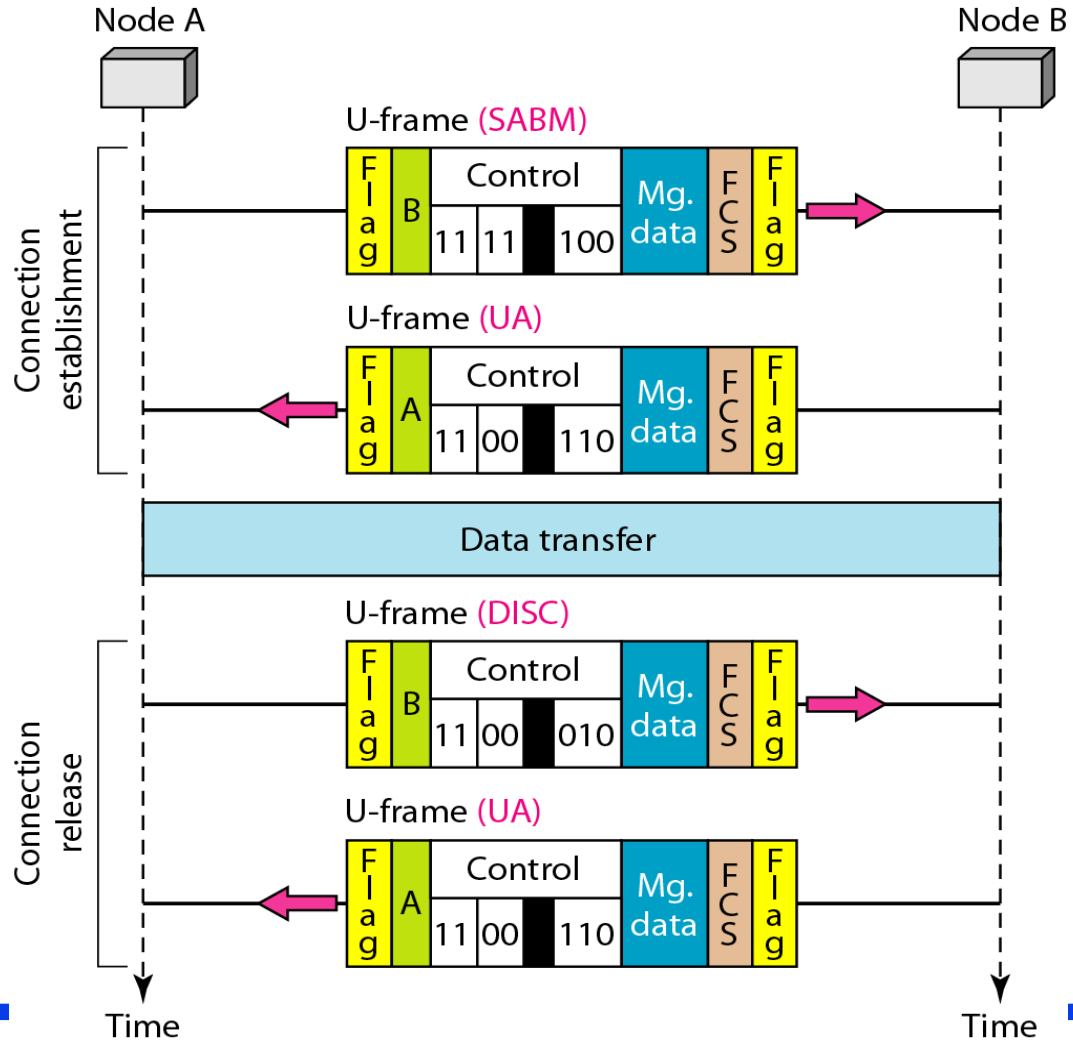
HDLC (cont'd)

Example 11.9

Figure 11.29 shows how U-frames can be used for connection establishment and connection release. 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).

HDLC (cont's)

Figure 11.29 *Example of connection and disconnection*



HDLC (cont'd)

Example 11.10

Figure 11.30 shows an exchange using piggybacking. Node A begins the exchange of information with an I-frame numbered 0 followed by another I-frame numbered 1. Node B piggybacks its acknowledgment of both frames onto an I-frame of its own. Node B's first I-frame is also numbered 0 [N(S) field] and contains a 2 in its N(R) field, acknowledging the receipt of A's frames 1 and 0 and indicating that it expects frame 2 to arrive next. Node B transmits its second and third I-frames (numbered 1 and 2) before accepting further frames from node A.

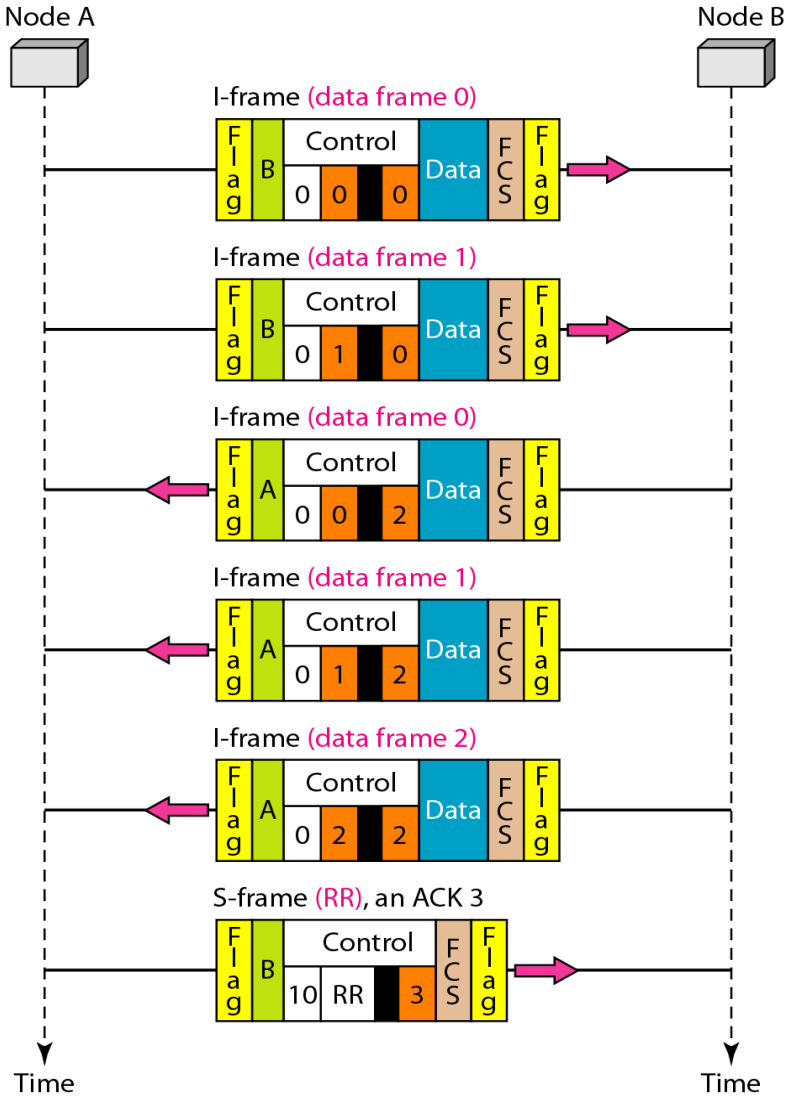
HDLC (cont'd)

Example 11.10 (continued)

Its N(R) information, therefore, has not changed: B frames 1 and 2 indicate that node B is still expecting A's frame 2 to arrive next. Node A has sent all its data. Therefore, it cannot piggyback an acknowledgment onto an I-frame and sends an S-frame instead. The RR code indicates that A is still ready to receive. The number 3 in the N(R) field tells B that frames 0, 1, and 2 have all been accepted and that A is now expecting frame number 3.

HDLC (cont'd)

Figure 11.30 *Example of piggybacking without error*



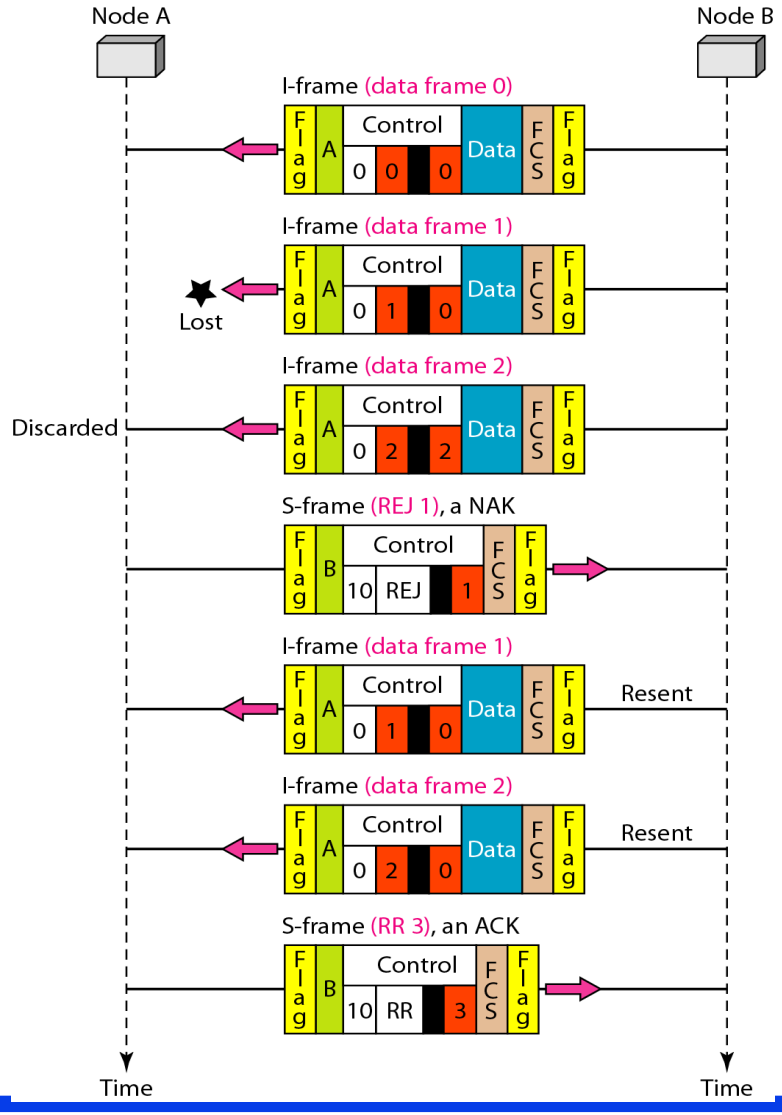
HDLC (cont'd)

Example 11.11

Figure 11.31 shows an exchange in which a frame is lost. Node B sends three data frames (0, 1, and 2), but frame 1 is lost. When node A receives frame 2, it discards it and sends a REJ frame for frame 1. Note that the protocol being used is Go-Back-N with the special use of an REJ frame as a NAK frame. The NAK frame does two things here: It confirms the receipt of frame 0 and declares that frame 1 and any following frames must be resent. Node B, after receiving the REJ frame, resends frames 1 and 2. Node A acknowledges the receipt by sending an RR frame (ACK) with acknowledgment number 3.

HDLC (cont'd)

Figure 11.31 *Example of piggybacking with error*



Summary (1)

- Data link control deals with the design and procedures for communication between two adjacent nodes: node-to-node communication.
- Frames can be of fixed or variable size. In fixed-size framing, there is no need for defining the boundaries of frames; in variable-size framing, we need a delimiter (flag) to define the boundary of two frames.
- Variable-size framing uses two categories of protocols: byte-oriented (or character-oriented) and bit-oriented. In a byte-oriented protocol, the data section of a frame is a sequence of bytes; in a bit-oriented protocol, the data section of a frame is a sequence of bits.
- In byte-oriented (or character-oriented) protocols, we use byte stuffing; a special byte added to the data section of the frame when there is a character with the same pattern as the flag.
- In bit-oriented protocols, we use bit stuffing; an extra 0 is added to the data section of the frame when there is a sequence of bits with the same pattern as the flag.
- 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 refers to methods of error detection and correction.

Summary (2)

- For the noiseless channel, we discussed two protocols: the Simplest Protocol and the Stop-and-Wait Protocol.
- For the noisy channel, we discussed three protocols: Stop-and-Wait ARQ, Go-Back-*N*, and Selective Repeat ARQ.
- Both Go-Back-*N*, and Selective-Repeat Protocols use a sliding window.
- A technique called piggybacking is used to improve the efficiency of the bidirectional protocols.
- High-level Data Link Control (HDLC) is a bit-oriented protocol for communication over point-to-point and multipoint links. However, the most common protocols for point-to-point access is the Point-to-Point Protocol (PPP), which is a byte-oriented protocol.

Q and A