

# *Data Link Control*

The two main functions of the data link layer are data link control and media access control. The first, data link control, deals with the design and procedures for communication between two adjacent nodes: node-to-node communication. We discuss this functionality in this chapter. The second function of the data link layer is media access control, or how to share the link. We discuss this functionality in Chapter 12.

**Data link control** functions include framing, flow and error control, and software-implemented protocols that provide smooth and reliable transmission of frames between nodes. In this chapter, we first discuss framing, or how to organize the bits that are carried by the physical layer. We then discuss flow and error control. A subset of this topic, techniques for error detection and correction, was discussed in Chapter 10.

To implement data link control, we need protocols. Each protocol is a set of rules that need to be implemented in software and run by the two nodes involved in data exchange at the data link layer. We discuss five protocols: two for noiseless (ideal) channels and three for noisy (real) channels. Those in the first category are not actually implemented, but provide a foundation for understanding the protocols in the second category.

After discussing the five protocol designs, we show how a bit-oriented protocol is actually implemented by using the High-level Data Link Control (HDLC) Protocol as an example. We also discuss a popular byte-oriented protocol, Point-to-Point Protocol (PPP).

---

### 11.1 FRAMING

Data transmission in the physical layer means moving bits in the form of a signal from the source to the destination. The physical layer provides bit synchronization to ensure that the sender and receiver use the same bit durations and timing.

The data link layer, on the other hand, 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. In addition, each envelope defines the sender and receiver addresses since the postal system is a many-to-many carrier facility.

Framing in the data link layer separates a message from one source to a destination, or from other messages to other destinations, by adding a sender address and a destination address. The destination address defines where the packet is to go; the sender address helps the recipient acknowledge the receipt.

Although the whole message could be packed in one frame, that is not normally done. One reason is that a frame can be very large, making flow and error control very inefficient. When a message is carried in one very large frame, even a single-bit error would require the retransmission of the whole message. When a message is divided into smaller frames, a single-bit error affects only that small frame.

### Fixed-Size Framing

Frames can be of fixed or variable size. In fixed-size framing, there is no need for defining the boundaries of the frames; the size itself can be used as a delimiter. An example of this type of framing is the ATM wide-area network, which uses frames of fixed size called cells. We discuss ATM in Chapter 18.

### Variable-Size Framing

Our main discussion in this chapter concerns variable-size framing, prevalent in local-area networks. In variable-size framing, we need a way to define the end of the frame and the beginning of the next. Historically, two approaches were used for this purpose: a character-oriented approach and a bit-oriented approach.

#### *Character-Oriented Protocols*

In a character-oriented protocol, data to be carried are 8-bit characters from a coding system such as ASCII (see Appendix A). The header, which normally carries the source and destination addresses and other control information, and the trailer, which carries error detection or error correction redundant bits, are also multiples of 8 bits. To separate one frame from the next, an 8-bit (1-byte) flag is added at the beginning and the end of a frame. The flag, composed of protocol-dependent special characters, signals the start or end of a frame. Figure 11.1 shows the format of a frame in a character-oriented protocol.

---

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

---

Data from upper layer

---



---

Character-oriented framing was popular when only text was exchanged by the data link layers. The flag could be selected to be any character not used for text communication. Now, however, we send other types of information such as graphs, audio, and video. Any pattern used for the flag could also be part of the information. If this happens, the receiver, when it encounters this pattern in the middle of the data, thinks it has reached the end of the frame. To fix this problem, a byte-stuffing strategy was added to

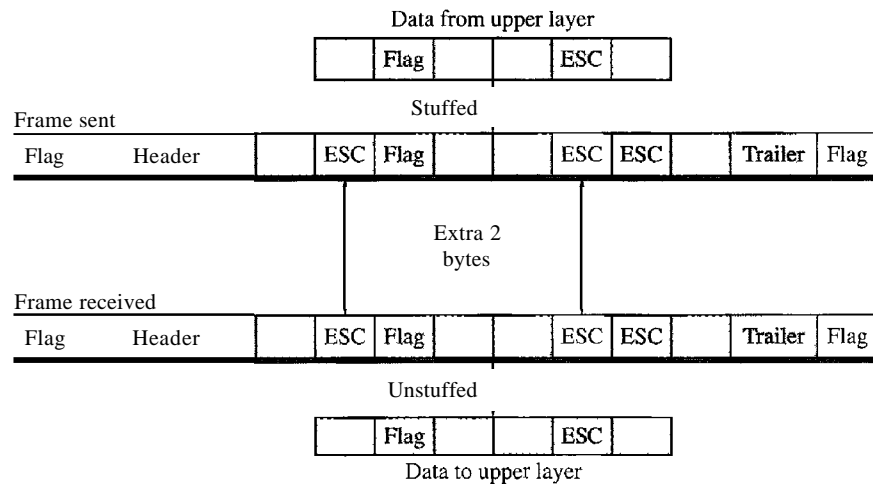
character-oriented framing. In byte stuffing (or character stuffing), a special byte is added to the data section of the frame when there is a character with the same pattern as the flag. The data section is stuffed with an extra byte. This byte is usually called the escape character (ESC), which has a predefined bit pattern. Whenever the receiver encounters the ESC character, it removes it from the data section and treats the next character as data, not a delimiting flag.

Byte stuffing by the escape character allows the presence of the flag in the data section of the frame, but it creates another problem. What happens if the text contains one or more escape characters followed by a flag? The receiver removes the escape character, but keeps the flag, which is incorrectly interpreted as the end of the frame. To solve this problem, the escape characters that are part of the text must also be marked by another escape character. In other words, if the escape character is part of the text, an extra one is added to show that the second one is part of the text. Figure 11.2 shows the situation.

---

Figure 11.2 *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.

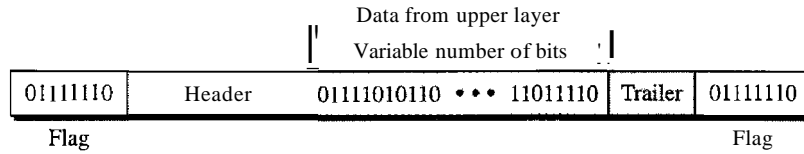
---

Character-oriented protocols present another problem in data communications. The universal coding systems in use today, such as Unicode, have 16-bit and 32-bit characters that conflict with 8-bit characters. We can say that in general, the tendency is moving toward the bit-oriented protocols that we discuss next.

### *Bit-Oriented Protocols*

In a bit-oriented protocol, 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. However, in addition to headers (and possible trailers), we still need a delimiter to separate one frame from the other. Most protocols use a special 8-bit pattern flag 01111110 as the delimiter to define the beginning and the end of the frame, as shown in Figure 11.3.

Figure 11.3 A frame in a bit-oriented protocol

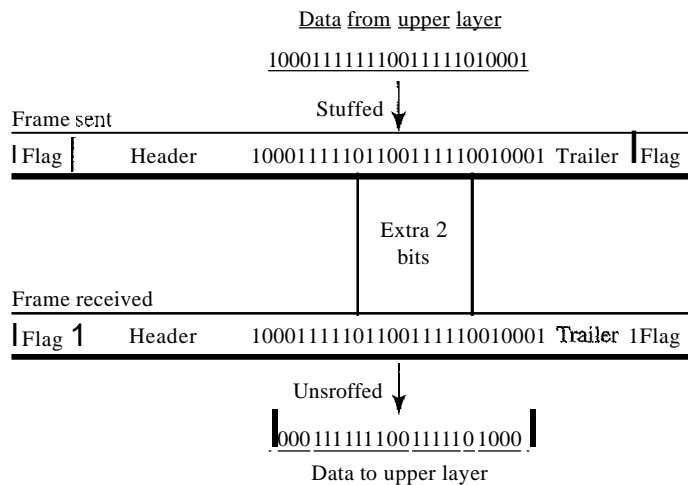


This flag can create the same type of problem we saw in the byte-oriented protocols. That is, if the flag pattern appears in the data, we need to somehow inform the receiver that this is not the end of the frame. We do this by stuffing 1 single bit (instead of 1 byte) to prevent the pattern from looking like a flag. The strategy is called bit stuffing. In bit stuffing, if a 0 and five consecutive 1 bits are encountered, an extra 0 is added. This extra stuffed bit is eventually removed from the data by the receiver. Note that the extra bit is added after one 0 followed by five 1s regardless of the value of the next bit. This guarantees that the flag field sequence does not inadvertently appear in the frame.

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.

Figure 11.4 shows bit stuffing at the sender and bit removal at the receiver. Note that even if we have a 0 after five 1s, we still stuff a 0. The 0 will be removed by the receiver.

Figure 11.4 Bit stuffing and unstuffing



This means that if the flaglike pattern 01111110 appears in the data, it will change to 011111010 (stuffed) and is not mistaken as a flag by the receiver. The real flag 01111110 is not stuffed by the sender and is recognized by the receiver.

---

## 11.2 FLOW AND ERROR CONTROL

Data communication requires at least two devices working together, one to send and the other to receive. Even such a basic arrangement requires a great deal of coordination for an intelligible exchange to occur. 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

Flow control coordinates the amount of data that can be sent before receiving an acknowledgment and is one of the most important duties of the data link layer. In most protocols, flow control is a set of procedures that tells the sender how much data it can transmit before it must wait for an acknowledgment from the receiver. The flow of data must not be allowed to overwhelm the receiver. Any receiving device has a limited speed at which it can process incoming data and a limited amount of memory in which to store incoming data. The receiving device must be able to inform the sending device before those limits are reached and to request that the transmitting device send fewer frames or stop temporarily. Incoming data must be checked and processed before they can be used. The rate of such processing is often slower than the rate of transmission. For this reason, each receiving device has a block of memory, called a *buffer*, reserved for storing incoming data until they are processed. If the buffer begins to fill up, the receiver must be able to tell the sender to halt transmission until it is once again able to receive.

---

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

Error control is both error detection and error correction. It allows the receiver to inform the sender of any frames lost or damaged in transmission and coordinates the retransmission of those frames by the sender. In the data link layer, the term *error control* refers primarily to methods of error detection and retransmission. Error control in the data link layer is often implemented simply: Any time an error is detected in an exchange, specified frames are retransmitted. This process is called automatic repeat request (ARQ).

---

Error control in the data link layer is based on automatic repeat request, which is the retransmission of data.

---

---

## 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

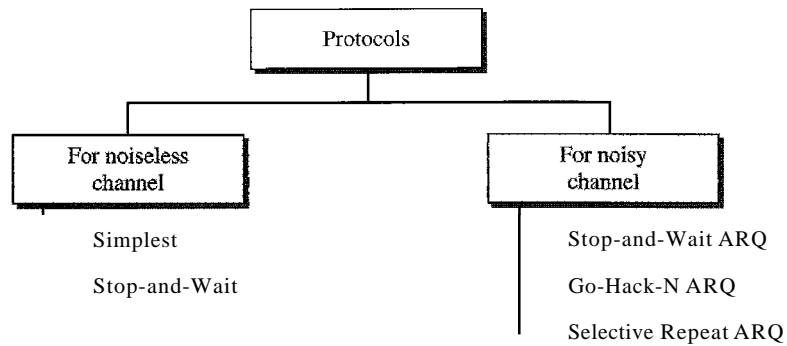
discussions language-free, we have written in pseudocode a version of each protocol that concentrates mostly on the procedure instead of delving into the details of language rules.

We divide the discussion of protocols into those that can be used for noiseless (error-free) channels and those that can be used for noisy (error-creating) channels. The protocols in the first category cannot be used in real life, but they serve as a basis for understanding the protocols of noisy channels. Figure 11.5 shows the classifications.

---

Figure 11.5 *Taxonomy of protocols discussed in this chapter*

---




---

There is a difference between the protocols we discuss here and those used in real networks. All the protocols we discuss are unidirectional in the sense that the data frames travel from one node, called the sender, to another node, called the receiver. Although special frames, called acknowledgment (ACK) and negative acknowledgment (NAK) can flow in the opposite direction for flow and error control purposes, data flow in only one direction.

In a real-life network, the data link protocols are implemented as bidirectional; data flow in both directions. In these protocols the flow and error control information such as ACKs and NAKs is included in the data frames in a technique called piggybacking. Because bidirectional protocols are more complex than unidirectional ones, we chose the latter for our discussion. If they are understood, they can be extended to bidirectional protocols. We leave this extension as an exercise.

---

## 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. The first is a protocol that does not use flow control; the second is the one that does. Of course, neither has error control because we have assumed that the channel is a perfect noiseless channel.

### Simplest Protocol

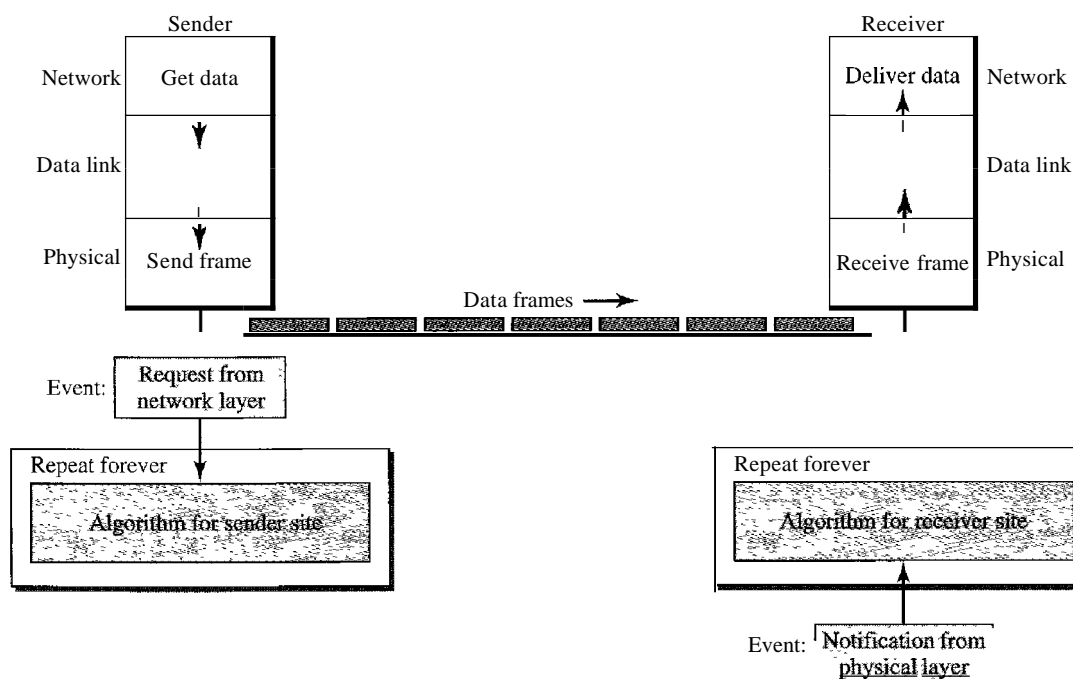
Our first protocol, which we call the Simplest Protocol for lack of any other name, is one that has no flow or error control. Like other protocols we will discuss in this chapter, it is a unidirectional protocol in which data frames are traveling in only one direction—from the

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. The data link layer of the receiver immediately removes the header from the frame and hands the data packet to its network layer, which can also accept the packet immediately. In other words, the receiver can never be overwhelmed with incoming frames.

### Design

There is no need for flow control in this scheme. The data link layer at the sender site gets data from its network layer, makes a frame out of the data, and sends it. The data link layer at the receiver site receives a frame from its physical layer, extracts data from the frame, and delivers the data to its network layer. The data link layers of the sender and receiver provide transmission services for their network layers. The data link layers use the services provided by their physical layers (such as signaling, multiplexing, and so on) for the physical transmission of bits. Figure 11.6 shows a design.

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



We need to elaborate on the procedure used by both data link layers. The sender site cannot send a frame until its network layer has a data packet to send. The receiver site 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. The procedure at the sender site is constantly running; there is no action until there is a request from the network layer. The procedure at the receiver site is also constantly running, but there is no action until notification from the physical layer arrives. Both procedures are constantly running because they do not know when the corresponding events will occur.

*Algorithms*

Algorithm 11.1 shows the procedure at the sender site.

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

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

**Analysis** The algorithm has an infinite loop, which means lines 3 to 9 are repeated forever once the program starts. The algorithm is an event-driven one, which means that it *sleeps* (line 3) until an event *wakes it up* (line 4). This means that there may be an undefined span of time between the execution of line 3 and line 4; there is a gap between these actions. When the event, a request from the network layer, occurs, lines 6 through 8 are executed. The program then repeats the loop and again sleeps at line 3 until the next occurrence of the event. We have written pseudocode for the main process. We do not show any details for the modules GetData, MakeFrame, and SendFrame. GetDataO takes a data packet from the network layer, MakeFrameO adds a header and delimiter flags to the data packet to make a frame, and SendFrameO delivers the frame to the physical layer for transmission.

Algorithm 11.2 shows the procedure at the receiver site.

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

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

**Analysis** This algorithm has the same format as Algorithm 11.1, except that the direction of the frames and data is upward. The event here is the arrival of a data frame. After the event occurs, the data link layer receives the frame from the physical layer using the ReceiveFrameO process, extracts the data from the frame using the ExtractDataO process, and delivers the data to the network layer using the DeliverDataO process. Here, we also have an event-driven algorithm because the algorithm never knows when the data frame will arrive.



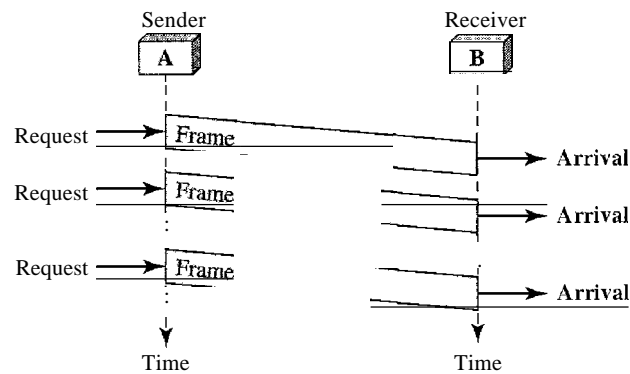
*Example 11.1*

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.

---

Figure 11.7 *Flow diagram for Example 11.1*

---




---

## Stop-and-Wait Protocol

If data frames arrive at the receiver site faster than they can be processed, the frames must be stored until their use. Normally, the receiver does not have enough storage space, especially if it is receiving data from many sources. This may result in either the discarding of frames or denial of service. To prevent the receiver from becoming overwhelmed with frames, we somehow need to tell the sender to slow down. There must be feedback from the receiver to the sender.

The protocol we discuss now is called the Stop-and-Wait Protocol because the sender sends one frame, stops until it receives confirmation from the receiver (okay to go ahead), and then sends the next frame. We still have unidirectional communication for data frames, but auxiliary ACK frames (simple tokens of acknowledgment) travel from the other direction. We add flow control to our previous protocol.

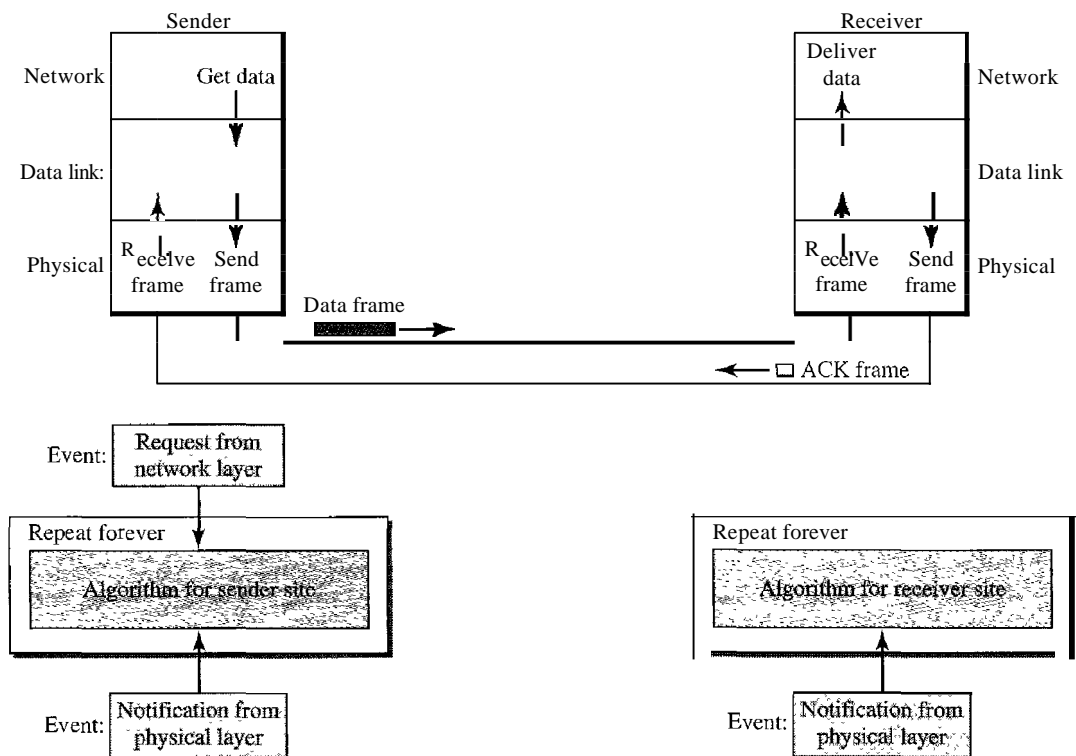
### *Design*

Figure 11.8 illustrates the mechanism. Comparing this figure with Figure 11.6, we can see the traffic on the forward channel (from sender to receiver) and the reverse channel. At any time, there is either one data frame on the forward channel or one ACK frame on the reverse channel. We therefore need a half-duplex link.

### *Algorithms*

Algorithm 11.3 is for the sender site.

Figure 11.8 Design of 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() i
8      MakeFrame();
9      SendFrame() i                             // Send the data frame
10     canSend = false;                          // cannot send until ACK arrives
11   }
12   WaitForEvent()                             // Sleep until an event occurs
13   if (Event (ArrivalNotification) /I An ACK has arrived
14   {
15     ReceiveFrame();                           // Receive the ACK frame
16     canSend = true;
17   }
18 }

```

**Analysis** Here two events can occur: a request from the network layer or an arrival notification from the physical layer. The responses to these events must alternate. In other words, after a frame is sent, the algorithm must ignore another network layer request until that frame is

acknowledged. We know that two arrival events cannot happen one after another because the channel is error-free and does not duplicate the frames. The requests from the network layer, however, may happen one after another without an arrival event in between. We need somehow to prevent the immediate sending of the data frame. Although there are several methods, we have used a simple *canSend* variable that can either be true or false. When a frame is sent, the variable is set to false to indicate that a new network request cannot be sent until *canSend* is true. When an ACK is received, *canSend* is set to true to allow the sending of the next frame.

Algorithm 11.4 shows the procedure at the receiver site.

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

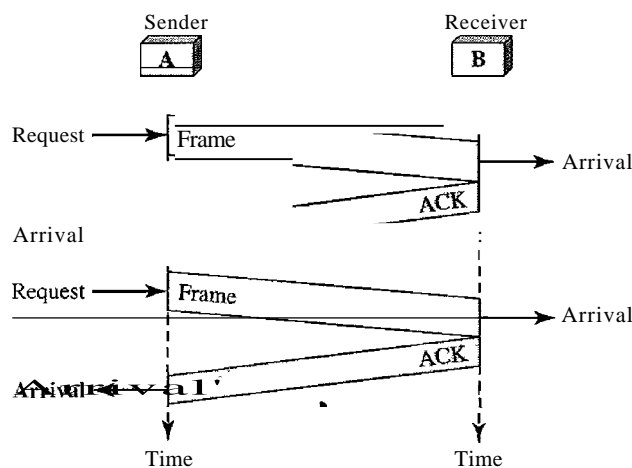
1	while (true)	II Repeat forever
2	{	
3	WaitForEvent();	II Sleep until an event occurs
4	if(Event(ArrivalNotification))	II Data frame arrives
5	{	
6	ReceiveFrame();	
7	ExtractData();	
8	Deliver(data);	II Deliver data to network layer
9	SendFrame();	II Send an ACK frame
10	}	
11	}	

**Analysis** This is very similar to Algorithm 11.2 with one exception. After the data frame arrives, the receiver sends an ACK frame (line 9) to acknowledge the receipt and allow the sender to send the next frame.

### 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.

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 can ignore the error (as we sometimes do), or we need to add error control to our protocols. We discuss three protocols in this section that use error control.

### Stop-and-Wait Automatic Repeat Request

Our first protocol, called the Stop-and-Wait Automatic Repeat Request (Stop-and-Wait ARQ), adds a simple error control mechanism to the Stop-and-Wait Protocol. Let us see how this protocol detects and corrects errors.

To detect and correct corrupted frames, we need to add redundancy bits to our data frame (see Chapter 10). When the frame arrives at the receiver site, it is checked and if it is corrupted, it is silently discarded. The detection of errors in this protocol is manifested by the silence of the receiver.

Lost frames are more difficult to handle than corrupted ones. In our previous protocols, there was no way to identify a frame. The received frame could be the correct one, or a duplicate, or a frame out of order. The solution is to number the frames. When the receiver receives a data frame that is out of order, this means that frames were either lost or duplicated.

The corrupted and lost frames need to be resent in this protocol. If the receiver does not respond when there is an error, how can the sender know which frame to resend? To remedy this problem, the sender keeps a copy of the sent frame. At the same time, it starts a timer. If the timer expires and there is no ACK for the sent frame, the frame is resent, the copy is held, and the timer is restarted. Since the protocol uses the stop-and-wait mechanism, there is only one specific frame that needs an ACK even though several copies of the same frame can be in the network.

---

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.

---

Since an ACK frame can also be corrupted and lost, it too needs redundancy bits and a sequence number. The ACK frame for this protocol has a sequence number field. In this protocol, the sender simply discards a corrupted ACK frame or ignores an out-of-order one.

#### *Sequence Numbers*

As we discussed, the protocol specifies that frames need to be numbered. This is done by using sequence numbers. A field is added to the data frame to hold the sequence number of that frame.

One important consideration is the range of the sequence numbers. Since we want to minimize the frame size, we look for the smallest range that provides unambiguous

communication. The sequence numbers of course can wrap around. For example, if we decide that the field is  $m$  bits long, the sequence numbers start from 0, go to  $2^m - 1$ , and then are repeated.

Let us reason out the range of sequence numbers we need. Assume we have used  $x$  as a sequence number; we only need to use  $x + 1$  after that. There is no need for  $x + 2$ . To show this, assume that the sender has sent the frame numbered  $x$ . Three things can happen.

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  $x + 1$  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.

We can see that there is a need for sequence numbers  $x$  and  $x + 1$  because the receiver needs to distinguish between case 1 and case 2. But there is no need for a frame to be numbered  $x + 2$ . In case 1, the frame can be numbered  $x$  again because frames  $x$  and  $x + 1$  are acknowledged and there is no ambiguity at either site. In cases 2 and 3, the new frame is  $x + 1$ , not  $x + 2$ . If only  $x$  and  $x + 1$  are needed, we can let  $x = 0$  and  $x + 1 = 1$ . This means that the sequence is 0, 1, 0, 1, 0, and so on. Is this pattern familiar? This is modulo-2 arithmetic as we saw in Chapter 10.

---

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

---

### *Acknowledgment Numbers*

Since the sequence numbers must be suitable for both data frames and ACK frames, we use this convention: The acknowledgment numbers always announce the sequence number of the next frame expected by the receiver. For example, if frame 0 has arrived safe and sound, the receiver sends an ACK frame with acknowledgment 1 (meaning frame 1 is expected next). If frame 1 has arrived safe and sound, the receiver sends an ACK frame with acknowledgment 0 (meaning frame 0 is expected).

---

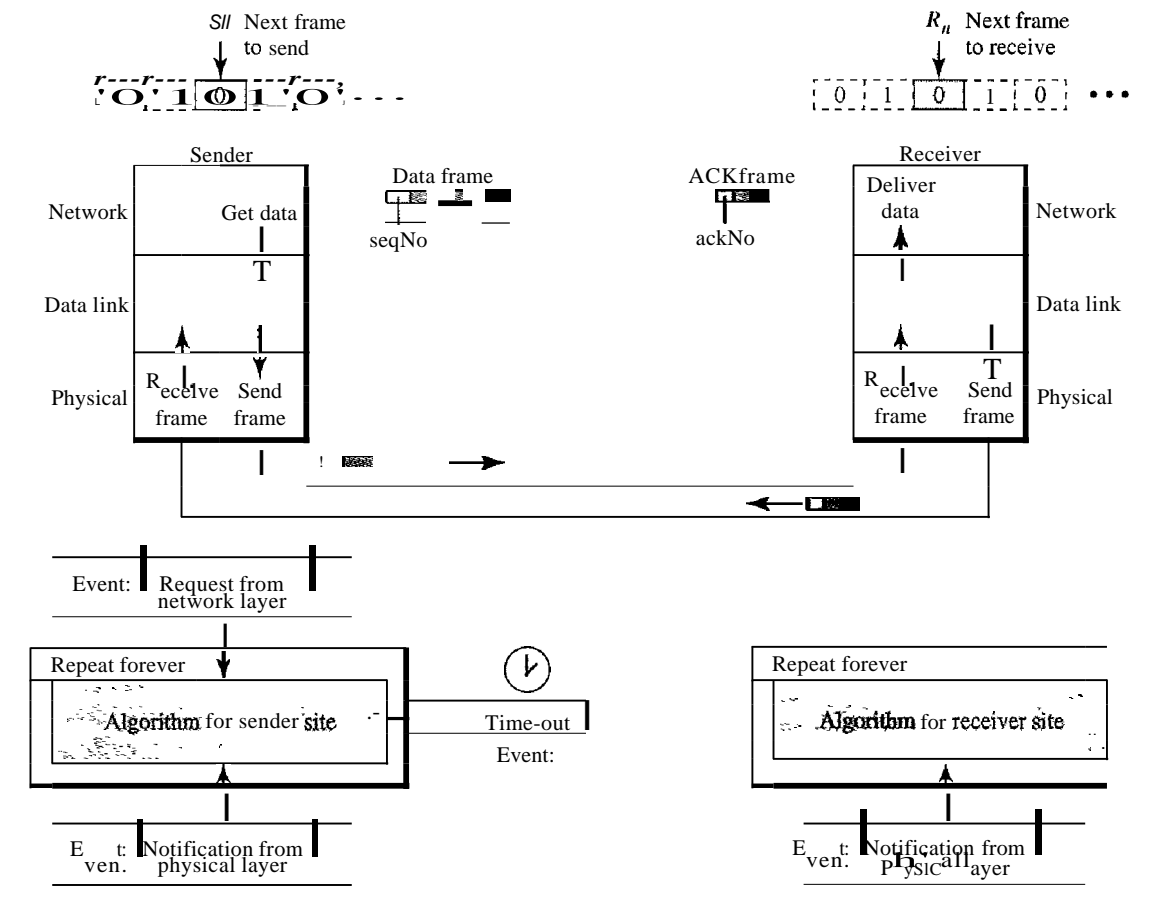
In Stop-and-Wait **ARQ**, the acknowledgment number always announces in modulo-2 arithmetic the sequence number of the next frame expected.

---

### *Design*

Figure 11.10 shows the design of the Stop-and-Wait ARQ Protocol. The sending device keeps a copy of the last frame transmitted until it receives an acknowledgment for that frame. A data frames uses a seqNo (sequence number); an ACK frame uses an ackNo (acknowledgment number). The sender has a control variable, which we call  $S_n$  (sender, next frame to send), that holds the sequence number for the next frame to be sent (0 or 1).

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



The receiver has a control variable, which we call  $R_n$  (receiver, next frame expected), that holds the number of the next frame expected. When a frame is sent, the value of  $S_n$  is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa. When a frame is received, the value of  $R_n$  is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa. Three events can happen at the sender site; one event can happen at the receiver site. Variable  $S_n$  points to the slot that matches the sequence number of the frame that has been sent, but not acknowledged;  $R_n$  points to the slot that matches the sequence number of the expected frame.

*Algorithms*

Algorithm 11.5 is for the sender site.

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

```

1  n = 0; // Frame 0 should be sent first
2  anSend = true; // Allow the first request to go
3  hile (true) // Repeat forever
4  {
5  WaitForEvent(); // Sleep until an event occurs

```

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

```

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      StartTimerO;
13      Sn = Sn + 1;
14      canSend = false;
15  }
16  WaitForEvent();               // Sleep
17  if (Event (ArrivalNotification) // An ACK has arrived
18  {
19      ReceiveFrame(ackNo);       //Receive the ACE fram
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 }

```

**Analysis** We first notice the presence of  $Sn'$  the sequence number of the next frame to be sent. This variable is initialized once (line 1), but it is incremented every time a frame is sent (line 13) in preparation for the next frame. However, since this is modulo-2 arithmetic, the sequence numbers are 0, 1, 0, 1, and so on. Note that the processes in the first event (SendFrame, StoreFrame, and PurgeFrame) use an  $Sn$  defining the frame sent out. We need at least one buffer to hold this frame until we are sure that it is received safe and sound. Line 10 shows that before the frame is sent, it is stored. The copy is used for resending a corrupt or lost frame. We are still using the canSend variable to prevent the network layer from making a request before the previous frame is received safe and sound. If the frame is not corrupted and the ackNo of the ACK frame matches the sequence number of the next frame to send, we stop the timer and purge the copy of the data frame we saved. Otherwise, we just ignore this event and wait for the next event to happen. After each frame is sent, a timer is started. When the timer expires (line 28), the frame is resent and the timer is restarted.

Algorithm 11.6 shows the procedure at the receiver site.

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

```

1   = 0;           // Frame 0 expected to arrive first
2   while (true)
3   {
4       WaitForEvent(); // Sleep until an event occurs

```

Algorithm 11.6 Receiver-site algorithm for Stop-and-Wait ARQ Protocol (continued)

```

5   if(Event(ArrivalNotification)» //Data frame arrives
6   {
7       ReceiveFrame()i
8       if(corrupted(frame)»i
9           sleep() i
10      if(seqNo == Rn) //Valid data frame
11      {
12          ExtractData();
13          DeliverData()i //Deliver data
14          Rn = Rn + 1;
15      }
16      SendFrame(Rn); //Send an ACK
17  }
18 }

```

**Analysis** This is noticeably different from Algorithm 11.4. First, all arrived data frames that are corrupted are ignored. If the seqNo of the frame is the one that is expected ( $R_n$ ), the frame is accepted, the data are delivered to the network layer, and the value of  $R_n$  is incremented. However, there is one subtle point here. Even if the sequence number of the data frame does not match the next frame expected, an ACK is sent to the sender. This ACK, however, just reconfirms the previous ACK instead of confirming the frame received. This is done because the receiver assumes that the previous ACK might have been lost; the receiver is sending a duplicate frame. The resent ACK may solve the problem before the time-out does it.

### Example 11.3

Figure 11.11 shows an example of Stop-and-Wait ARQ. Frame 2 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 2 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.

### Efficiency

The Stop-and-Wait ARQ discussed in the previous section is very inefficient if our channel is *thick* and *long*. By *thick*, we mean that our channel has a large bandwidth; by *long*, we mean the round-trip delay is long. The product of these two is called the bandwidth-delay product, as we discussed in Chapter 3. We can think of the channel as a pipe. The bandwidth-delay product then is the volume of the pipe in bits. The pipe is always there. If we do not use it, we are inefficient. The 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.

### 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?

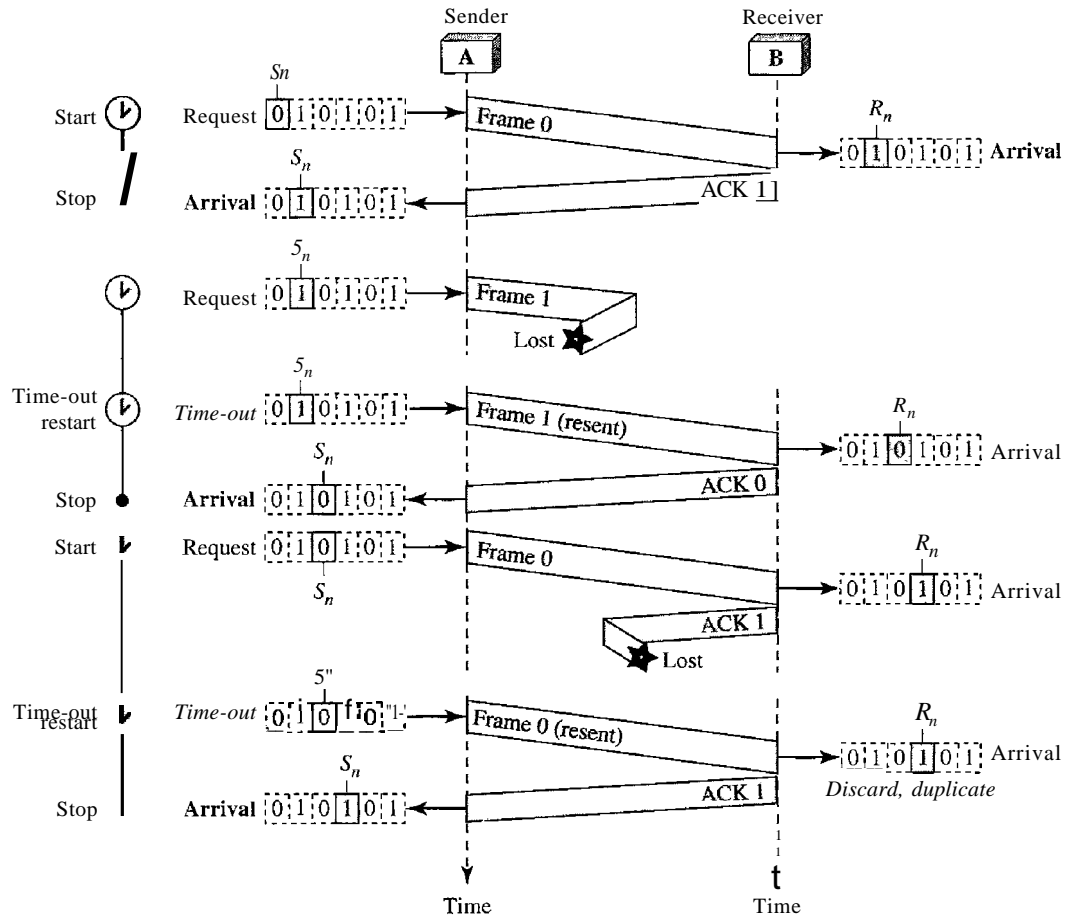
### Solution

The bandwidth-delay product is

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



Figure 11.11 Flow diagram for Example 11.3



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.

### 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 because we need to wait for a frame to reach the destination and be acknowledged before the next frame can be sent. However, pipelining does apply to our next two protocols because

several frames can be sent before we receive news about the previous frames. 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 Automatic Repeat Request

To improve the efficiency of transmission (filling the pipe), multiple frames must be in transition while waiting for acknowledgment. In other words, we need to let more than one frame be outstanding to keep the channel busy while the sender is waiting for acknowledgment. In this section, we discuss one protocol that can achieve this goal; in the next section, we discuss a second.

The first is called Go-Back-N Automatic Repeat Request (the rationale for the name will become clear later). In this protocol we can send several frames before receiving acknowledgments; we keep a copy of these frames until the acknowledgments arrive.

### *Sequence Numbers*

Frames from a sending station are numbered sequentially. However, because we need to include the sequence number of each frame in the header, we need to set a limit. If the header of the frame allows  $m$  bits for the sequence number, the sequence numbers range from 0 to  $2^m - 1$ . For example, if  $m$  is 4, the only sequence numbers are 0 through 15 inclusive. However, we can repeat the sequence. So the sequence numbers are

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, ...

In other words, the sequence numbers are modulo- $2^m$ .

---

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.

---

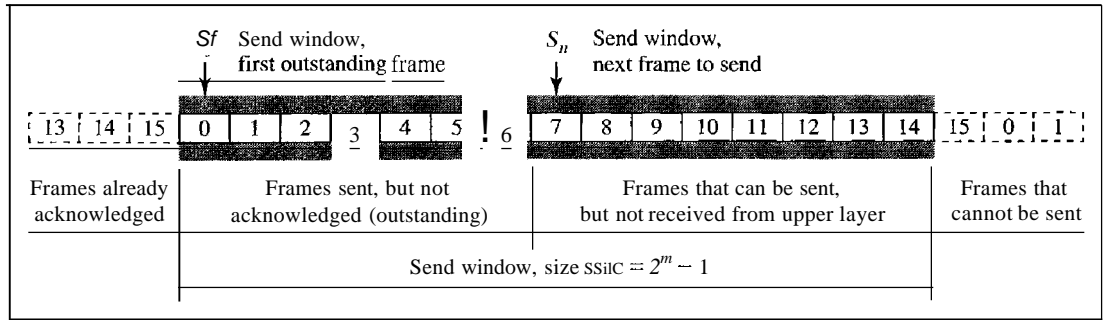
### *Sliding Window*

In this protocol (and the next), the sliding window is an abstract concept that defines the range of sequence numbers that is the concern of the sender and receiver. In other words, the sender and receiver need to deal with only part of the possible sequence numbers. The range which is the concern of the sender is called the send sliding window; the range that is the concern of the receiver is called the receive sliding window. We discuss both here.

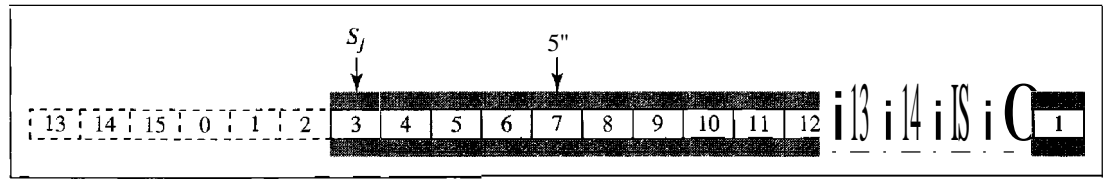
The send window is an imaginary box covering the sequence numbers of the data frames which can be in transit. In each window position, some of these sequence numbers define the frames that have been sent; others define those that can be sent. The maximum size of the window is  $2^m - 1$  for reasons that we discuss later. In this chapter, we let the size be fixed and set to the maximum value, but we will see in future chapters that some protocols may have a variable window size. Figure 11.12 shows a sliding window of size 15 ( $m=4$ ).

The window at any time divides the possible sequence numbers into four regions. The first region, from the far left to the left wall of the window, defines the sequence

Figure 11.12 Send window for Go-Back-NARQ



a. Send window before sliding



b. Send window after sliding

numbers belonging to frames that are already acknowledged. The sender does not worry about these frames and keeps no copies of them. The second region, colored in Figure 11.12a, defines the range of sequence numbers belonging to the frames that are sent and have an unknown status. The sender needs to wait to find out if these frames have been received or were lost. We call these outstanding frames. The third range, white in the figure, defines the range of sequence numbers for frames that can be sent; however, the corresponding data packets have not yet been received from the network layer. Finally, the fourth region defines sequence numbers that cannot be used until the window slides, as we see next.

The window itself is an abstraction; three variables define its size and location at any time. We call these variables  $S_f$  (send window, the first outstanding frame),  $S_n$  (send window, the next frame to be sent), and  $S_{size}$  (send window, size). The variable  $S_f$  defines the sequence number of the first (oldest) outstanding frame. The variable  $S_n$  holds the sequence number that will be assigned to the next frame to be sent. Finally, the variable  $S_{size}$  defines the size of the window, which is fixed in our protocol.

---

The send window is an abstract concept defining an imaginary box of size  $2^m - 1$  with three variables:  $S_f$ ,  $S_n$ , and  $S_{size}$

---

Figure 11.12b shows how a send window can slide one or more slots to the right when an acknowledgment arrives from the other end. As we will see shortly, the acknowledgments in this protocol are cumulative, meaning that more than one frame can be acknowledged by an ACK frame. In Figure 11.12b, frames 0, 1, and 2 are acknowledged, so the window has slid to the right three slots. Note that the value of  $S_f$  is 3 because frame 3 is now the first outstanding frame.

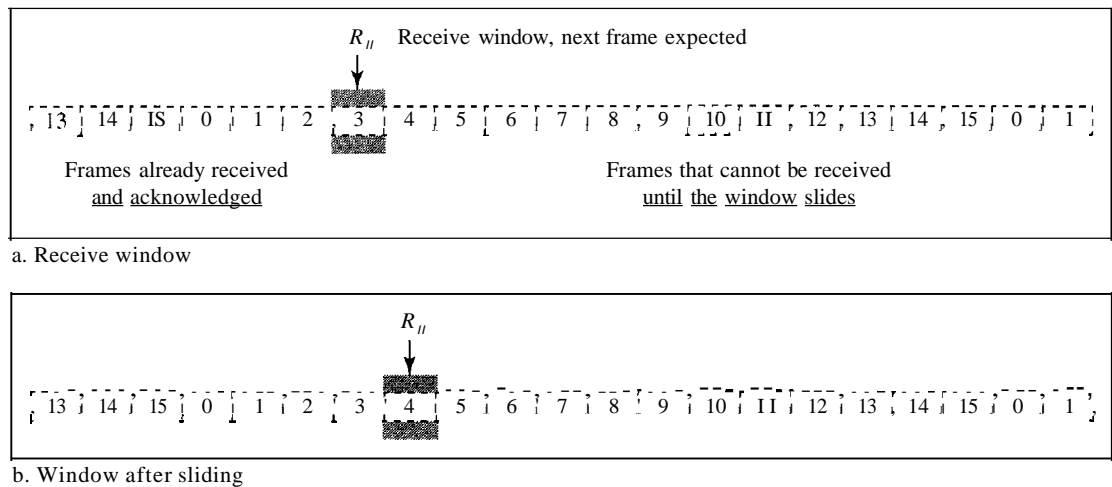
---

The send window can slide one or more slots when a valid acknowledgment arrives.

---

The receive window makes sure that the correct data frames are received and that the correct acknowledgments are sent. The size of the receive window is always 1. The receiver is always looking for the arrival of a specific frame. Any frame arriving out of order is discarded and needs to be resent. Figure 11.13 shows the receive window.

Figure 11.13 Receive window for Go-Back-NARQ



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.

Note that we need only one variable  $R_n$  (receive window, next frame expected) to define this abstraction. The sequence numbers to the left of the window belong to the frames already received and acknowledged; the sequence numbers to the right of this window define the frames that cannot be received. Any received frame with a sequence number in these two regions is discarded. Only a frame with a sequence number matching the value of  $R_n$  is accepted and acknowledged.

The receive window also slides, but only one slot at a time. When a correct frame is received (and a frame is received only one at a time), the window slides.

### Timers

Although there can be a timer for each frame that is sent, in our protocol we use only one. The reason is that the timer for the first outstanding frame always expires first; we send all outstanding frames when this timer expires.

### Acknowledgment

The receiver sends a positive acknowledgment if a frame has arrived safe and sound and in order. If a frame is damaged or is received out of order, the receiver is silent and will discard all subsequent frames until it receives the one it is expecting. The silence of

the receiver causes the timer of the unacknowledged frame at the sender site to expire. This, in turn, causes the sender to go back and resend all frames, beginning with the one with the expired timer. The receiver does not have to acknowledge each frame received. It can send one cumulative acknowledgment for several frames.

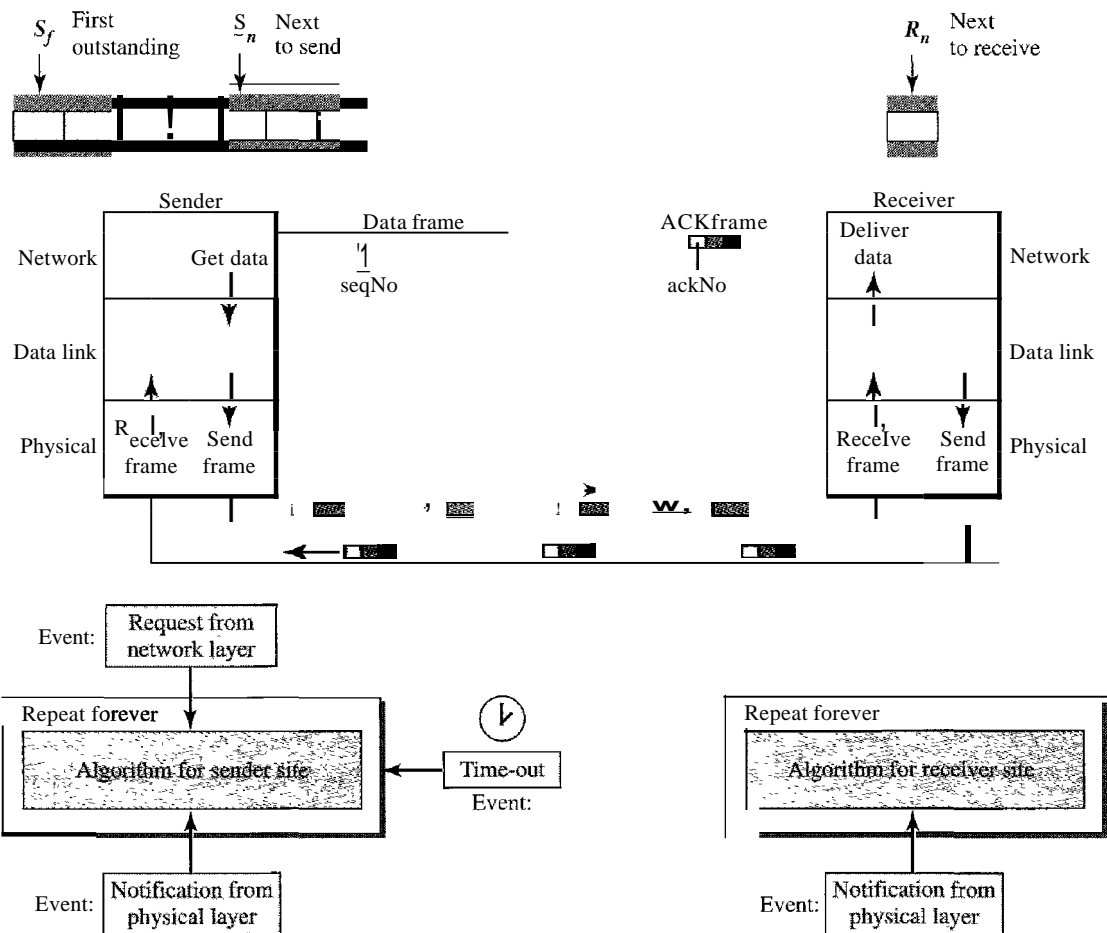
*Resending a Frame*

When the timer expires, the sender resends all outstanding frames. For example, suppose the sender has already sent frame 6, but the timer for frame 3 expires. This means that frame 3 has not been acknowledged; the sender goes back and sends frames 3, 4, 5, and 6 again. That is why the protocol is called *Go-Back-NARQ*.

*Design*

Figure 11.14 shows the design for this protocol. As we can see, multiple frames can be in transit in the forward direction, and multiple acknowledgments in the reverse direction. The idea is similar to Stop-and-Wait ARQ; the difference is that the send

Figure 11.14 Design of Go-Back-NARQ

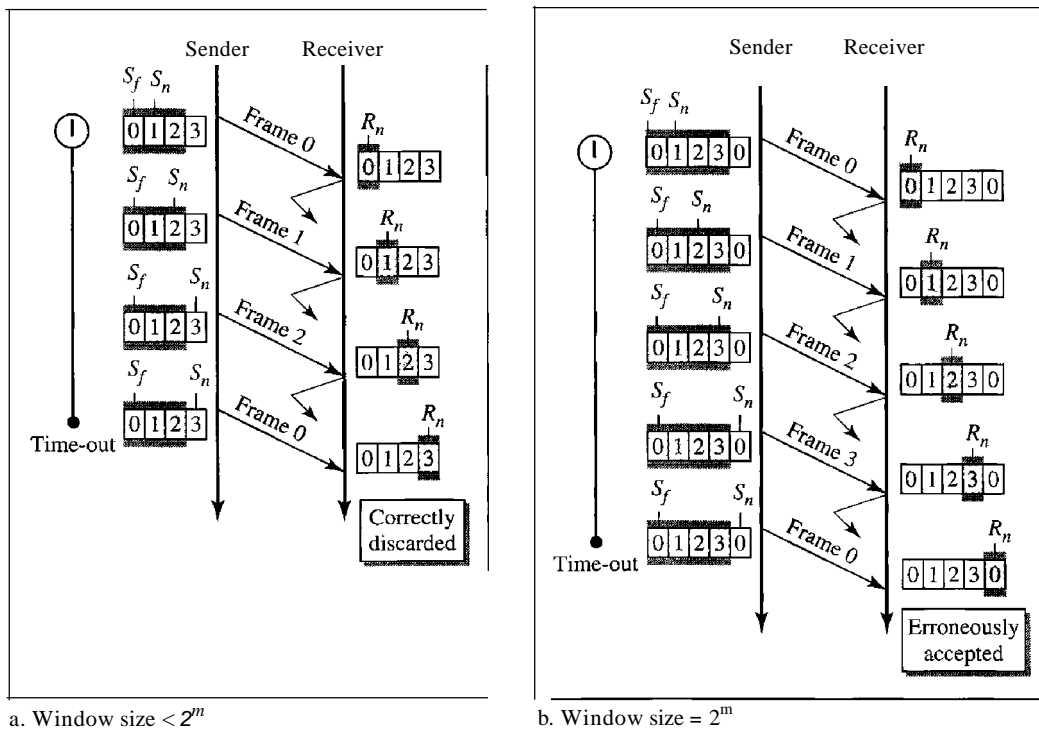


window allows us to have as many frames in transition as there are slots in the send window.

*Send Window Size*

We can now show why the size of the send window must be less than  $2^m$ . As an example, we choose  $m = 2$ , which means the size of the window can be  $2^m - 1$ , or 3. Figure 11.15 compares a window size of 3 against a window size of 4. If the size of the window is 3 (less than  $2^2$ ) and all three acknowledgments are lost, the frame timer expires and all three frames are resent. The receiver is now expecting frame 3, not frame 0, so the duplicate frame is correctly discarded. On the other hand, if the size of the window is 4 (equal to  $2^2$ ) and all acknowledgments are lost, the sender will send a duplicate of frame 0. However, this time the window of the receiver expects to receive frame 0, so it accepts frame 0, not as a duplicate, but as the first frame in the next cycle. This is an error.

Figure 11.15 Window size for Go-Back-NARQ



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

*Algorithms*

Algorithm 11.7 shows the procedure for the sender in this protocol.

Algorithm 11.7 *Go-Back-N sender algorithm*

```

1 Sw = 2m - 1;
2 Sf = 0;
3 Sn = 0
4
5 while (true) //Repeat forever
6 {
7   WaitForEvent();
8   if(Event(RequestToSend) //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    if(timer not running)
18      StartTimer();
19  }
20
21  if{Event{ArrivalNotification} //ACK arrives
22  {
23    Receive(ACK);
24    if{corrupted{ACK}
25      Sleep();
26    if{{ackNo>sf}&&{ackNO<=Sn} //If a valid ACK
27    While(Sf <= ackNo)
28      {
29        PurgeFrame{Sf);
30        Sf = Sf + 1;
31      }
32    StopTimer();
33  }
34
35  if{Event{TimeOut} //The timer expires
36  {
37    StartTimer();
38    Temp = Sf;
39    while{Temp < Sn);
40    {
41      SendFrame(Sf);
42      Sf = Sf + 1;
43    }
44  }
45 }

```

**Analysis** This algorithm first initializes three variables. Unlike Stop-and-Wait ARQ, this protocol allows several requests from the network layer without the need for other events to occur; we just need to be sure that the window is not full (line 12). In our approach, if the window is full,

the request is just ignored and the network layer needs to try again. Some implementations use other methods such as enabling or disabling the network layer. The handling of the arrival event is more complex than in the previous protocol. If we receive a corrupted ACK, we ignore it. If the ackNa belongs to one of the outstanding frames, we use a loop to purge the buffers and move the left wall to the right. The time-out event is also more complex. We first start a new timer. We then resend all outstanding frames.

Algorithm 11.8 is the procedure at the receiver site.

Algorithm 11.8 *Go-Back-N receiver algorithm*

```

1  Rn = 0;
2
3  while (true)                                II Repeat forever
4  {
5      WaitForEvent();
6
7      if(Event{ArrivalNotification} >> /Data frame arrives
8      (
9          Receive(Frame);
10         if(corrupted(Frame) >>
11             Sleep();
12         if(seqNo == Rn)                    III If expected frame
13         {
14             DeliverData(i)                 IID Deliver data
15             Rn = Rn + 1;                   IISlide window
16             SendACK(Rn);
17         }
18     }
19 }
```

**Analysis** This algorithm is simple. We ignore a corrupt or out-of-order frame. If a frame arrives with an expected sequence number, we deliver the data, update the value of  $R_n$ , and send an ACK with the ackNa showing the next frame expected.

*Example 11.6*

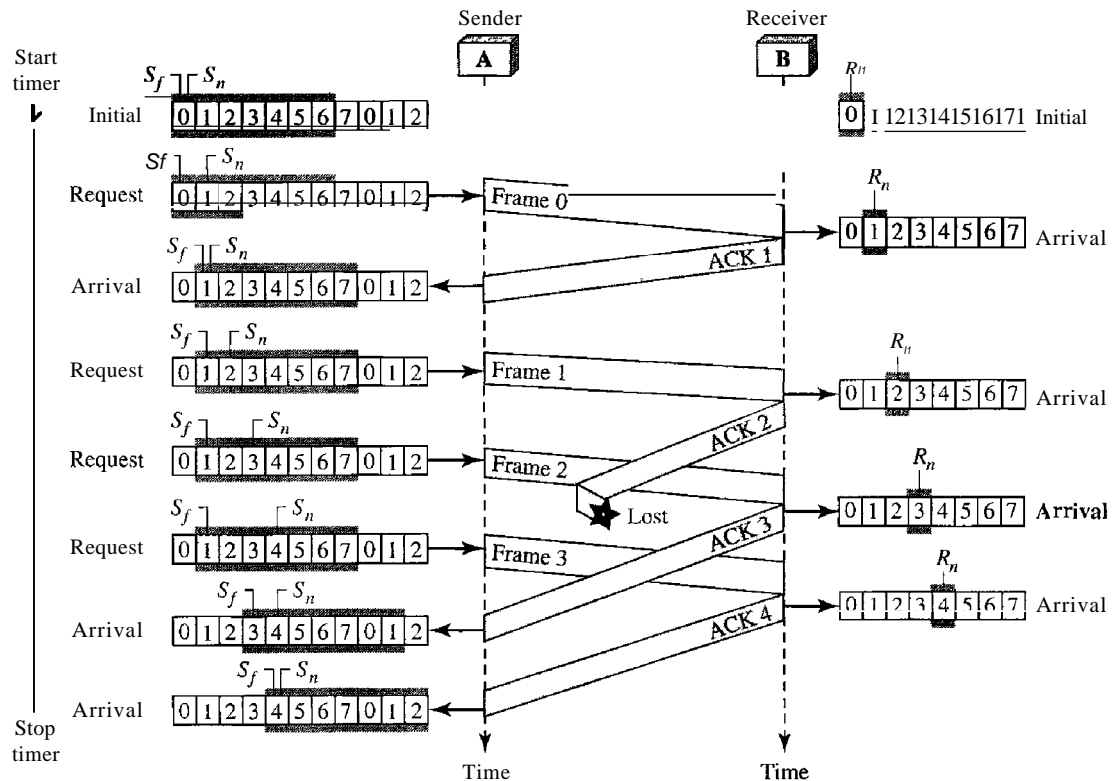
Figure 11.16 shows an example of Go-Back-N. This is an example of a case where the forward channel is reliable, but the reverse is not. No data frames are lost, but some ACKs are delayed and one is lost. The example also shows how cumulative acknowledgments can help if acknowledgments are delayed or 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.

There are four receiver events, all triggered by the arrival of frames from the physical layer.



Figure 11.16 Flow diagram for Example 11.6



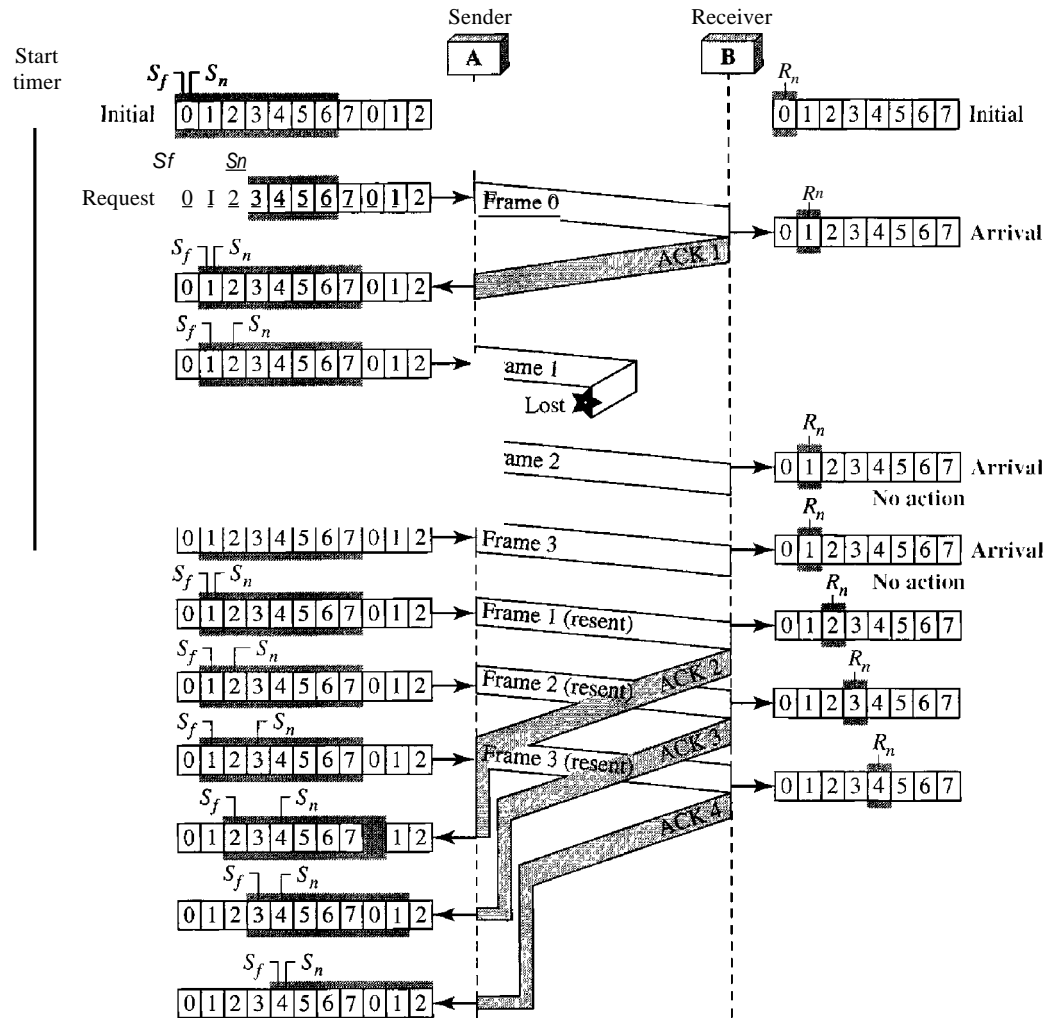
### Example 11.7

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 (frame 1 is expected). 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-NARQ Versus Stop-and-Wait ARQ

The reader may find that there is a similarity between *Go-Back-NARQ* and *Stop-and-Wait ARQ*. We can say that the *Stop-and-Wait ARQ* Protocol is actually a *Go-Back-NARQ* in which there are only two sequence numbers and the send window size is 1. In other words,  $m = 1$ ,  $2^m - 1 = 1$ . In *Go-Back-NARQ*, we said that the addition is modulo- $2^m$ ; in *Stop-and-Wait ARQ* it is 2, which is the same as  $2^m$  when  $m = 1$ .

Figure 11.17 Flow diagram for Example 11.7



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 Automatic Repeat Request

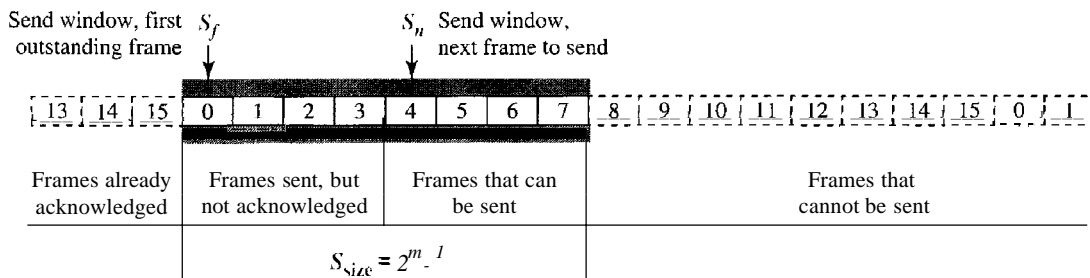
Go-Back-N ARQ simplifies the process at the receiver site. The receiver keeps track of only one variable, and there is no need to buffer out-of-order frames; they are simply discarded. However, this protocol is very inefficient for a noisy link. In a noisy link a frame has a higher probability of damage, which means the resending of multiple frames. This resending uses up the bandwidth and slows down the transmission. For noisy links, there is another mechanism that does not resend  $N$  frames when just one frame is damaged; only the damaged frame is resent. This mechanism is called Selective Repeat ARQ. It is more efficient for noisy links, but the processing at the receiver is more complex.

Windows

The Selective Repeat Protocol also uses two windows: a send window and a receive window. However, there are differences between the windows in this protocol and the ones in Go-Back-N. First, the size of the send window is much smaller; it is  $2^{m-1}$ . The reason for this will be discussed later. Second, the receive window is the same size as the send window.

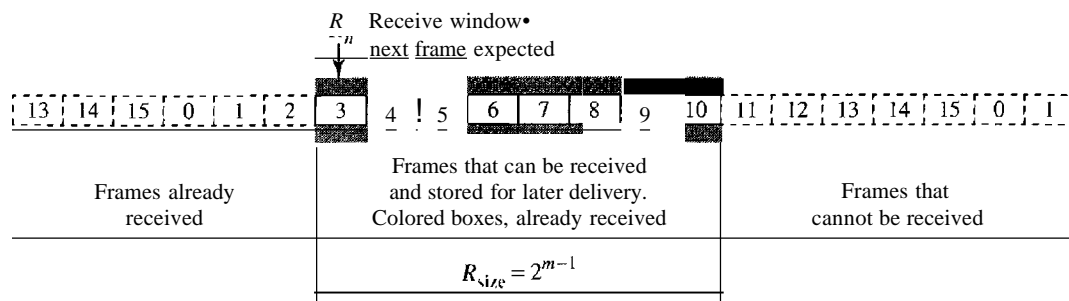
The send window maximum size can be  $2^{m-1}$ . For example, if  $m = 4$ , the sequence numbers go from 0 to 15, but the size of the window is just 8 (it is 15 in the *Go-Back-N* Protocol). The smaller window size means less efficiency in filling the pipe, but the fact that there are fewer duplicate frames can compensate for this. The protocol uses the same variables as we discussed for Go-Back-N. We show the Selective Repeat send window in Figure 11.18 to emphasize the size. Compare it with Figure 11.12.

**Figure 11.18** Send window for Selective Repeat ARQ



The receive window in Selective Repeat is totally different from the one in Go-Back-N. First, the size of the receive window is the same as the size of the send window ( $2^{m-1}$ ). 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. Because the sizes of the send window and receive window are the same, all the frames in the send frame can arrive out of order and be stored until they can be delivered. We need, however, to mention that the receiver never delivers packets out of order to the network layer. Figure 11.19 shows the receive window in this

**Figure 11.19** Receive window for Selective Repeat ARQ

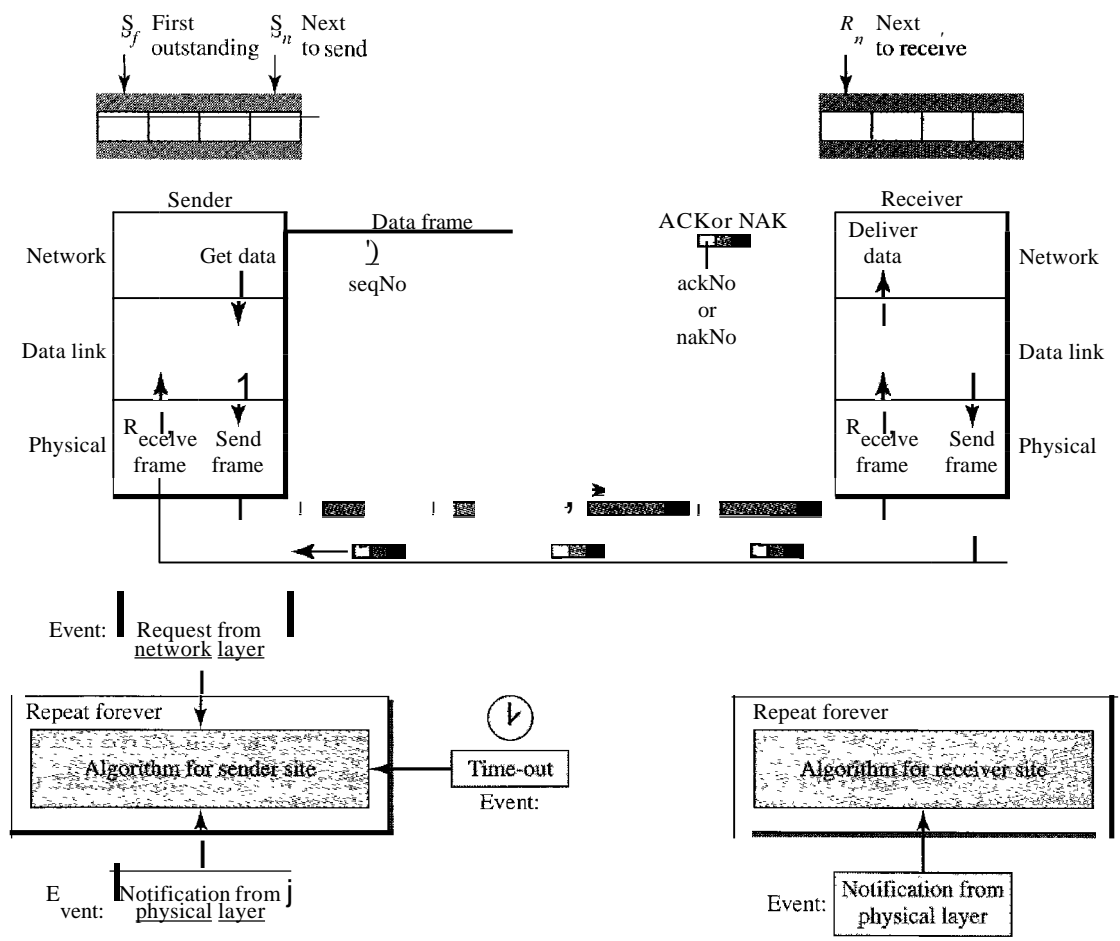


protocol. Those slots inside the window that are colored define frames that have arrived out of order and are waiting for their neighbors to arrive before delivery to the network layer.

*Design*

The design in this case is to some extent similar to the one we described for the 00-Back-N, but more complicated, as shown in Figure 11.20.

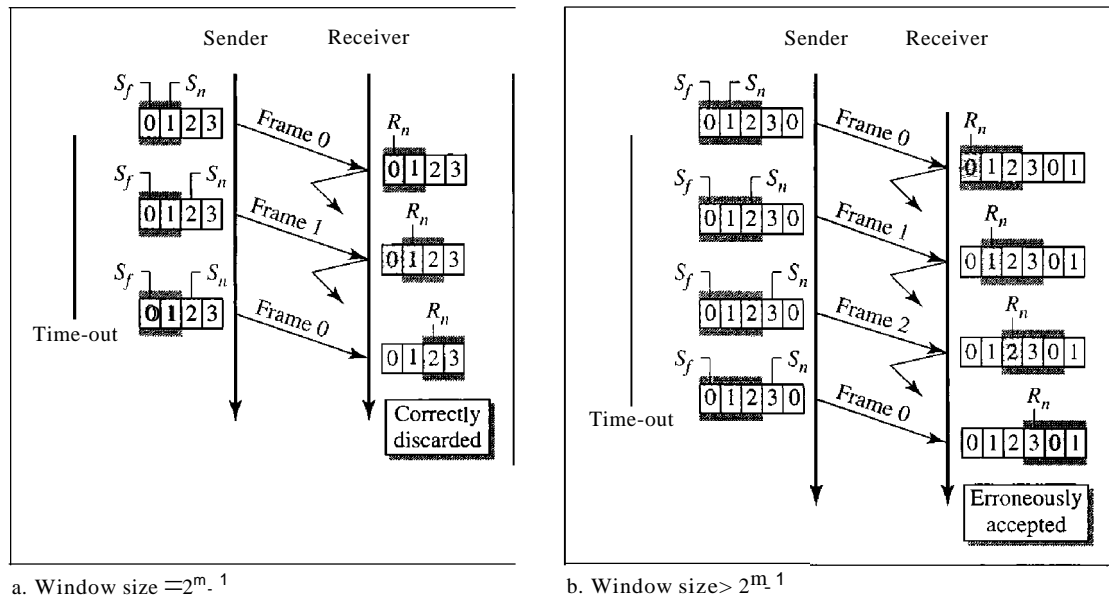
Figure 11.20 Design of Selective Repeat ARQ



*Window Sizes*

We can now show why the size of the sender and receiver windows must be at most one-half of  $2^m$ . For an example, we choose  $m = 2$ , which means the size of the window is  $2^m/2$ , or 2. Figure 11.21 compares a window size of 2 with a window size of 3.

If the size of the window is 2 and all acknowledgments are lost, the timer for frame 0 expires and frame 0 is resent. However, the window of the receiver is now expecting

Figure 11.21 *Selective Repeat ARQ, window size*

frame 2, not frame 0, so this duplicate frame is correctly discarded. When the size of the window is 3 and all acknowledgments are lost, the sender sends a duplicate of frame 0. However, this time, the window of the receiver expects to receive frame 0 (0 is part of the window), so it accepts frame 0, not as a duplicate, but as the first frame in the next cycle. This is clearly an error.

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

### Algorithms

Algorithm 11.9 shows the procedure for the sender.

#### Algorithm 11.9 *Sender-site Selective Repeat algorithm*

```

1   =  $2^{m-1} - 1$ 
2   =  $O_i$ 
3   =  $O_i$ 
4
5   hile (true)                               //Repeat forever
6   {
7     WaitForEvent()
8     if(Event(RequestToSend))                 //There is a packet to sen
9     {

```

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

```

10     if{Sn-S;E >= Sw}                I/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
20 if(Event{ArrivalNotification}» IIACK arrives
21 {
22     Receive{frame};                I/Receive ACK or NAK
23     if{corrupted{frame}»
24         Sleep{};
25     if (FrameType == NAK)
26         if (nakNo between Sf and So)
27         {
28             resend{nakNo};
29             StartTimer{nakNo};
30         }
31     if (FrameType == ACK)
32         if (ackNo between Sf and So)
33         {
34             while{sf < ackNo}
35             {
36                 Purge(sf);
37                 stopTimer(Sf);
38                 Sf = Sf + 1;
39             }
40         }
41 }
42
43 if(Event{TimeOut{t}»}                liThe timer expires
44 {
45     StartTimer{t};
46     SendFrame{t};
47 }
48 }
```

**Analysis** The handling of the request event is similar to that of the previous protocol except that one timer is started for each frame sent. The arrival event is more complicated here. An ACK or a NAK frame may arrive. If a valid NAK frame arrives, we just resend the corresponding frame. If a valid ACK arrives, we use a loop to purge the buffers, stop the corresponding timer, and move the left wall of the window. The time-out event is simpler here; only the frame which times out is resent.

Algorithm 11.10 shows the procedure for the receiver.

Algorithm 11.10 Receiver-site Selective Repeat algorithm

```

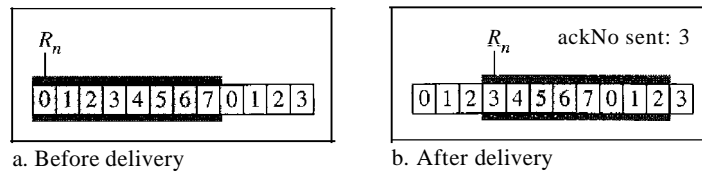
1  Rn = 0;
2  NakSent = false;
3  AckNeeded = false;
4  Repeat(for all slots)
5      Marked(slot) = false;
6
7  !while (true)                                IIRepeat forever
8  {
9      WaitForEvent(i)
10
11     if{Event{ArrivalNotification}»          jData frame arrives
12     {
13         Receive(Frame);
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);
23             NakSent = true;
24             if {(seqNo in window)&&(IMarked(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 }

```

**Analysis** Here we need more initialization. In order not to overwhelm the other side with NAKs, we use a variable called NakSent. To know when we need to send an ACK, we use a variable called AckNeeded. Both of these are initialized to false. We also use a set of variables to

mark the slots in the receive window once the corresponding frame has arrived and is stored. If we receive a corrupted frame and a NAK has not yet been sent, we send a NAK to tell the other site that we have not received the frame we expected. If the frame is not corrupted and the sequence number is in the window, we store the frame and mark the slot. If contiguous frames, starting from  $R_n$  have been marked, we deliver their data to the network layer and slide the window. Figure 11.22 shows this situation.

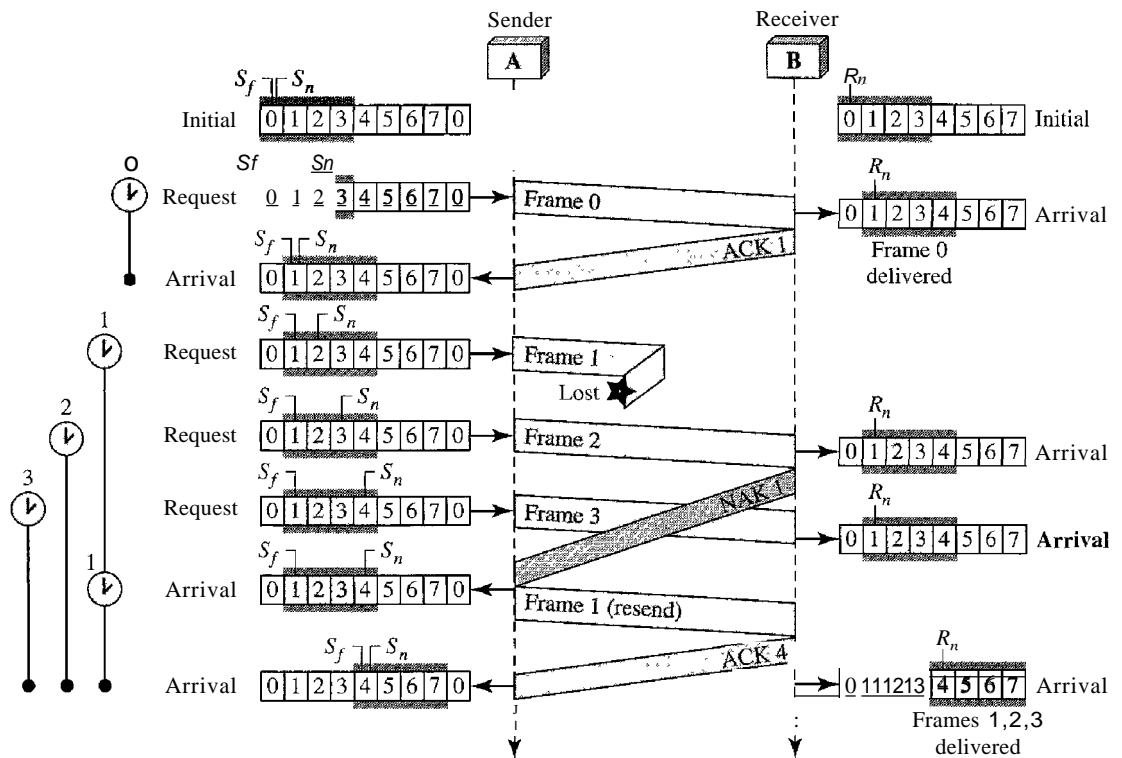
Figure 11.22 Delivery of data in Selective Repeat ARQ



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.

Figure 11.23 Flow diagram for Example 11.8





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.

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 (colored slot), 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. After the first arrival, there was only one frame and it started from the beginning of the window. After the last arrival, there are three frames and the first one starts from the beginning of the window.

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 a NAK 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.

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.

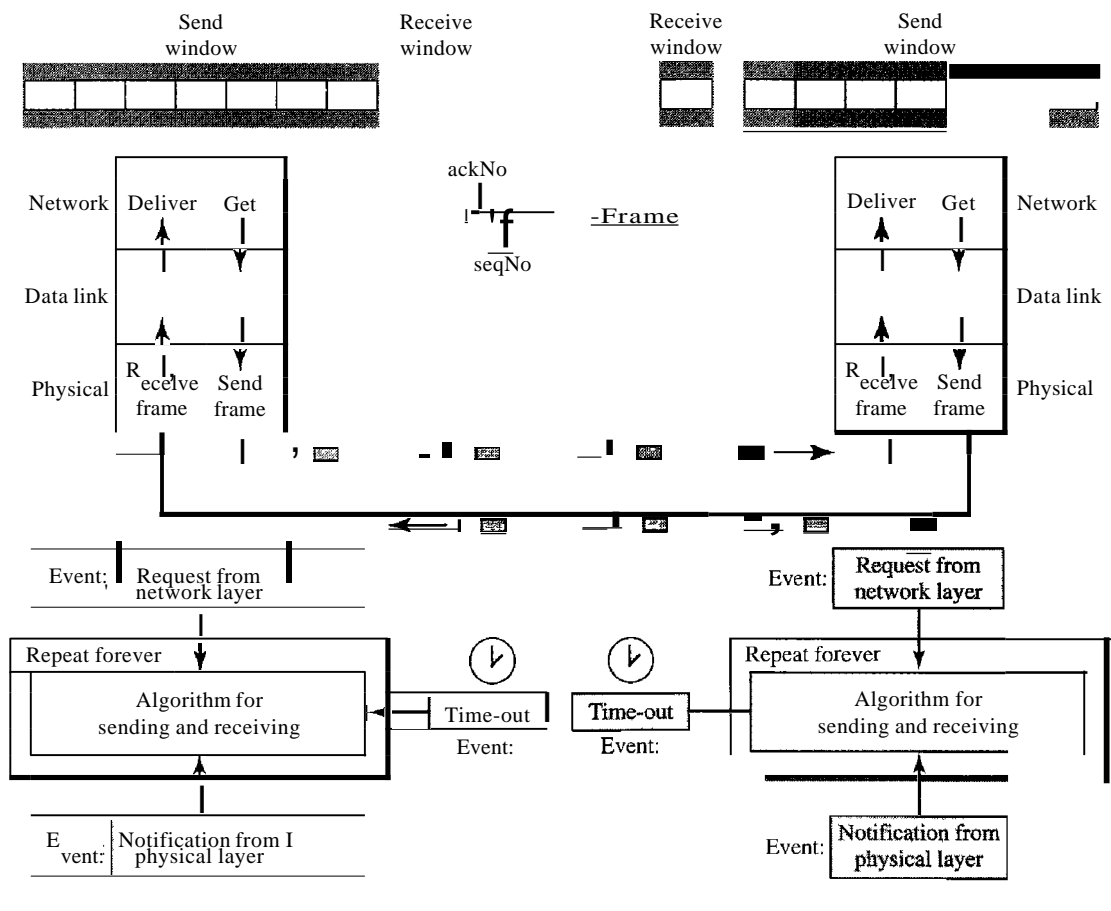
## Piggybacking

The three protocols we discussed in this section are all unidirectional: data frames flow in only one direction although control information such as ACK and NAK frames can travel in the other direction. In real life, data frames are normally flowing in both directions: from node A to node B and from node B to node A. This means that the control information also needs to flow in both directions. A technique called **piggybacking** is used to improve the efficiency of the bidirectional protocols. When a frame is carrying data from A to B, it can also carry control information about arrived (or lost) frames from B; when a frame is carrying data from B to A, it can also carry control information about the arrived (or lost) frames from A.

We show the design for a Go-Back-N ARQ using piggybacking in Figure 11.24. Note that each node now has two windows: one send window and one receive window. Both also need to use a timer. Both are involved in three types of events: request, arrival, and time-out. However, the arrival event here is complicated; when a frame arrives, the site needs to handle control information as well as the frame itself. Both of these concerns must be taken care of in one event, the arrival event. The request event uses only the send window at each site; the arrival event needs to use both windows.

An important point about piggybacking is that both sites must use the same algorithm. This algorithm is complicated because it needs to combine two arrival events into one. We leave this task as an exercise.

Figure 11.24 Design of piggybacking in Go-Back-NARQ



## 11.6 HDLC

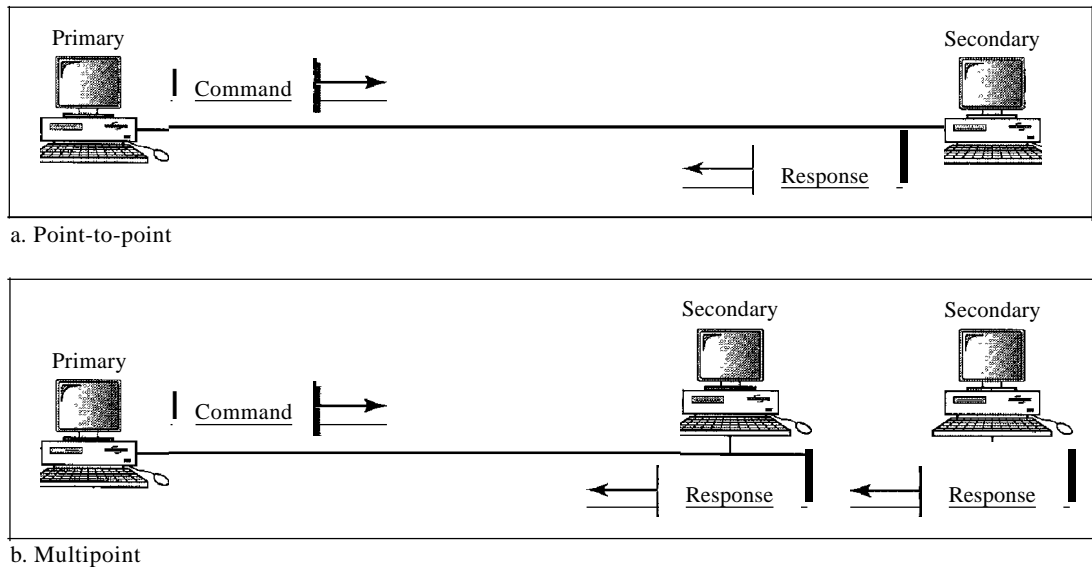
High-level Data Link Control (HDLC) is a bit-oriented protocol for communication over point-to-point and multipoint links. It implements the ARQ mechanisms we discussed in this chapter.

### Configurations and Transfer Modes

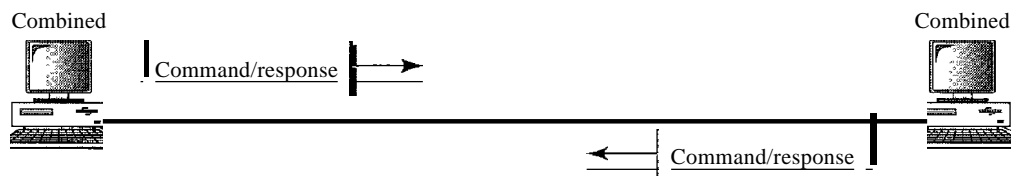
HDLC provides two common transfer modes that can be used in different configurations: normal response mode (NRM) and asynchronous balanced mode (ABM).

#### Normal Response Mode

In normal response mode (NRM), the station configuration is unbalanced. We have one primary station and multiple secondary stations. A primary station can send commands; a secondary station can only respond. The NRM is used for both point-to-point and multiple-point links, as shown in Figure 11.25.

Figure 11.25 *Normal response mode**Asynchronous Balanced Mode*

In asynchronous balanced mode (ABM), the configuration is balanced. The link is point-to-point, and each station can function as a primary and a secondary (acting as peers), as shown in Figure 11.26. This is the common mode today.

Figure 11.26 *Asynchronous balanced mode***Frames**

To provide the flexibility necessary to support all the options possible in the modes and configurations just described, HDLC defines three types of frames: information frames (I-frames), supervisory frames (S-frames), and unnumbered frames (V-frames). Each type of frame serves as an envelope for the transmission of a different type of message. I-frames are used to transport user data and control information relating to user data (piggybacking). S-frames are used only to transport control information. V-frames are reserved for system management. Information carried by V-frames is intended for managing the link itself.

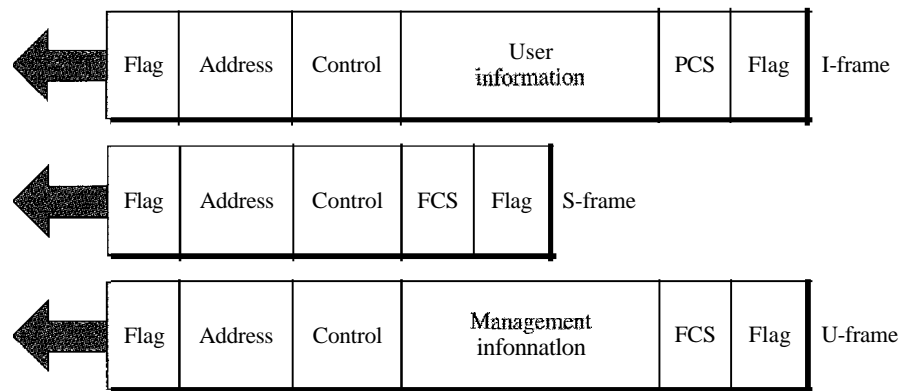
*Frame Format*

Each frame in HDLC may contain up to six fields, as shown in Figure 11.27: a beginning flag field, an address field, a control field, an information field, a frame check sequence (FCS) field, and an ending flag field. In multiple-frame transmissions, the ending flag of one frame can serve as the beginning flag of the next frame.

---

Figure 11.27 HDLC frames

---

*Fields*

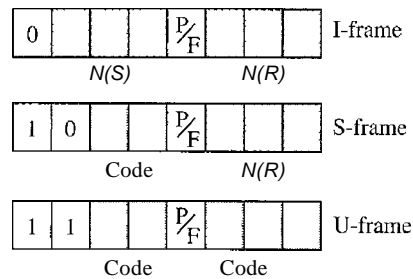
Let us now discuss the fields and their use in different frame types.

- Flag field. The flag field of an HDLC frame 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. The second field of an HDLC frame contains the address of the secondary station. If a primary station created the frame, it contains a *to* address. If a secondary creates the frame, it contains *afrom* address. An address field can be 1 byte or several bytes long, depending on the needs of the network. One byte can identify up to 128 stations (1 bit is used for another purpose). Larger networks require multiple-byte address fields. If the address field is only 1 byte, the last bit is always a 1. If the address is more than 1 byte, all bytes but the last one will end with 0; only the last will end with 1. Ending each intermediate byte with 0 indicates to the receiver that there are more address bytes to come.
- Control field. The control field is a 1- or 2-byte segment of the frame used for flow and error control. The interpretation of bits in this field depends on the frame type. We discuss this field later and describe its format for each frame type.
- Information field. The information field contains the user's data from the network layer or management information. Its length can vary from one network to another.
- FCS field. The frame check sequence (FCS) is the HDLC error detection field. It can contain either a 2- or 4-byte ITU-T CRC.

## Control Field

The control field determines the type of frame and defines its functionality. So let us discuss the format of this field in greater detail. The format is specific for the type of frame, as shown in Figure 11.28.

Figure 11.28 Control field format for the different frame types



### Control Field for I-Frames

I-frames are designed to carry user data from the network layer. In addition, they can include flow and error control information (piggybacking). The subfields in the control field are used to define these functions. The subfields in the control field are used to define these functions. The first bit defines the type. If the first bit of the control field is 0, this means the frame is an I-frame. The next 3 bits, called  $N(S)$ , define the sequence number of the frame. Note that with 3 bits, we can define a sequence number between 0 and 7; but in the extension format, in which the control field is 2 bytes, this field is larger. The last 3 bits, called  $N(R)$ , correspond to the acknowledgment number when piggybacking is used. The single bit between  $N(S)$  and  $N(R)$  is called the  $PIF$  bit. The  $PIF$  field is a single bit with a dual purpose. It has meaning only when it is set (bit = 1) and can mean poll or final. It means *poll* when the frame is sent by a primary station to a secondary (when the address field contains the address of the receiver). It means *final* when the frame is sent by a secondary to a primary (when the address field contains the address of the sender).

### Control Field for S-Frames

Supervisory frames are used for flow and error control whenever piggybacking is either impossible or inappropriate (e.g., when the station either has no data of its own to send or needs to send a command or response other than an acknowledgment). S-frames do not have information fields. If the first 2 bits of the control field is 10, this means the frame is an S-frame. The last 3 bits, called  $N(R)$ , corresponds to the acknowledgment number (ACK) or negative acknowledgment number (NAK) depending on the type of S-frame. The 2 bits called code is used to define the type of S-frame itself. With 2 bits, we can have four types of S-frames, as described below:

- Receive ready (RR). If the value of the code subfield is 00, it is an RR S-frame. This kind of frame acknowledges the receipt of a safe and sound frame or group of frames. In this case, the value  $N(R)$  field defines the acknowledgment number.

- Receive not ready (RNR). If the value of the code subfield is 10, it is an RNR S-frame. This kind of frame is an RR frame with additional functions. It acknowledges the receipt of a frame or group of frames, and it announces that the receiver is busy and cannot receive more frames. It acts as a kind of congestion control mechanism by asking the sender to slow down. The value of *NCR* is the acknowledgment number.
- Reject (REJ). If the value of the code subfield is 01, it is a REJ S-frame. This is a NAK frame, but not like the one used for Selective Repeat ARQ. It is a NAK that can be used in *Go-Back-N* ARQ to improve the efficiency of the process by informing the sender, before the sender time expires, that the last frame is lost or damaged. The value of *NCR* is the negative acknowledgment number.
- Selective reject (SREJ). If the value of the code subfield is 11, it is an SREJ S-frame. This is a NAK frame used in Selective Repeat ARQ. Note that the HDLC Protocol uses the term *selective reject* instead of *selective repeat*. The value of *N(R)* is the negative acknowledgment number.

#### *Control Field for V-Frames*

Unnumbered frames are used to exchange session management and control information between connected devices. Unlike S-frames, U-frames contain an information field, but one used for system management information, not user data. As with S-frames, however, much of the information carried by U-frames is contained in codes included in the control field. U-frame codes are divided into two sections: a 2-bit prefix before the PtF bit and a 3-bit suffix after the PtF bit. Together, these two segments (5 bits) can be used to create up to 32 different types of U-frames. Some of the more common types are shown in Table 11.1.

Table 11.1 *U-frame control command and response*

<i>Code</i>	<i>Command</i>	<i>Response</i>	<i>Meaning</i>
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

*Example 11.9: Connection/Disconnection*

Figure 11.29 shows how V-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 (VA) 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 VA (unnumbered acknowledgment).

Figure 11.29 Example of connection and disconnection

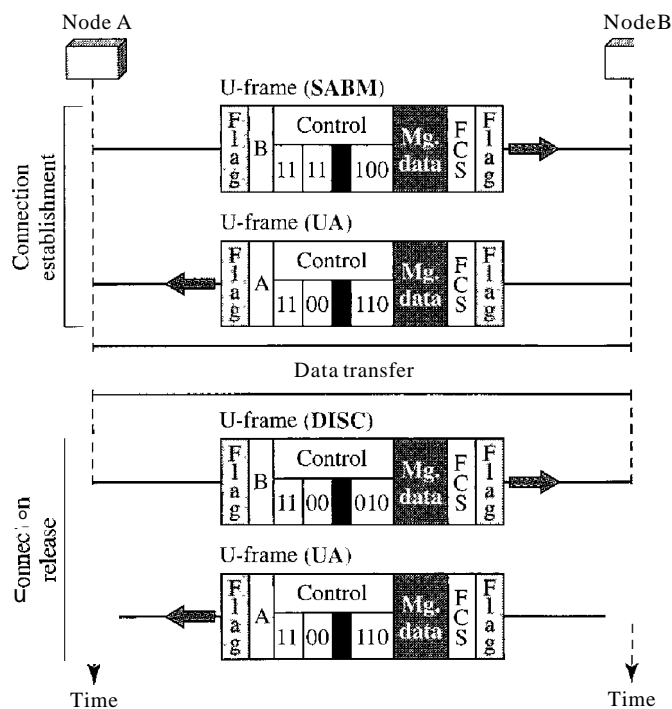
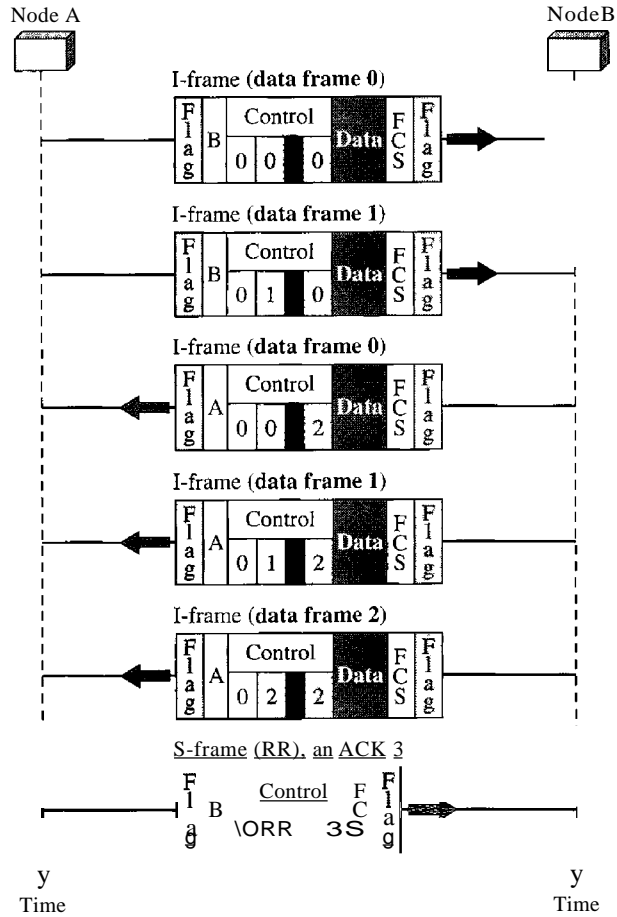
*Example 11.10: Piggybacking without Error*

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. 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.

Figure 11.30 Example of piggybacking without error



*Example 11.11: Piggybacking with Error*

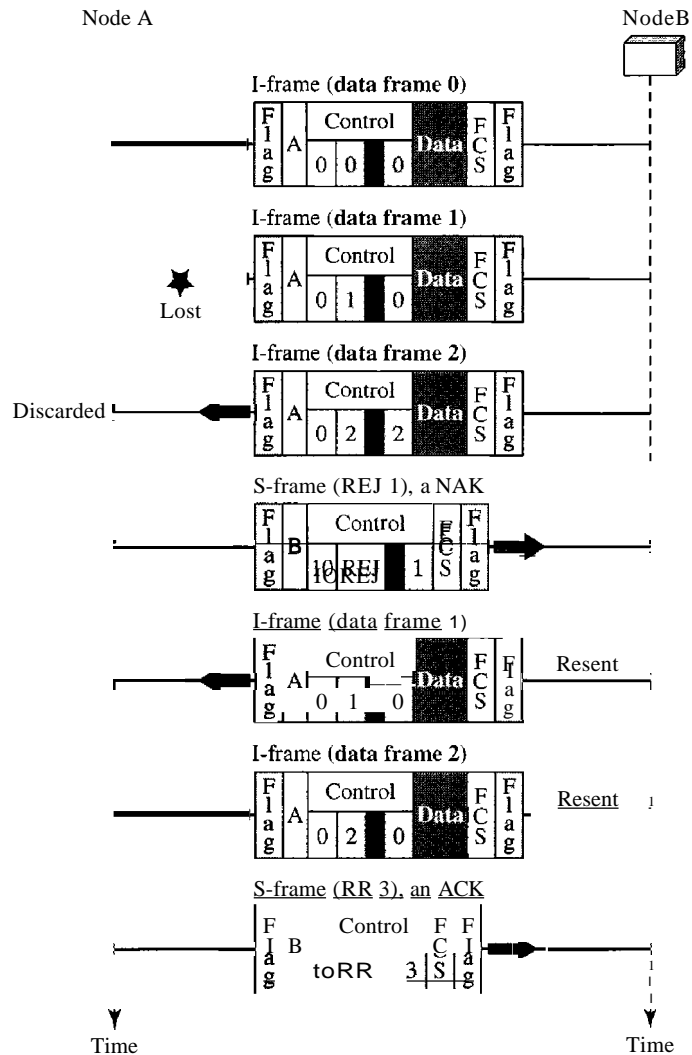
Figure 11.31 shows an exchange in which a frame is lost. Node B sends three data frames (0, 1, and 2), but frame 1 is lost. When node A receives frame 2, it discards it and sends a *REI* frame for frame 1. Note that the protocol being used is *Go-Back-N* with the special use of an REI frame as a NAK 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 REI frame, resends frames 1 and 2. Node A acknowledges the receipt by sending an RR frame (ACK) with acknowledgment number 3.

## 11.7 POINT-TO-POINT PROTOCOL

Although HDLC is a general protocol that can be used for both point-to-point and multi-point configurations, one of the most common protocols for point-to-point access is the Point-to-Point Protocol (PPP). Today, millions of Internet users who need to connect their home computers to the server of an Internet service provider use PPP. The majority of these users have a traditional modem; they are connected to the Internet through a telephone line, which provides the services of the physical layer. But to control and



Figure 11.31 Example of piggybacking with error



manage the transfer of data, there is a need for a point-to-point protocol at the data link layer. PPP is by far the most common.

PPP provides several services:

1. PPP defines the format of the frame to be exchanged between devices.
2. PPP defines how two devices can negotiate the establishment of the link and the exchange of data.
3. PPP defines how network layer data are encapsulated in the data link frame.
4. PPP defines how two devices can authenticate each other.
5. PPP provides multiple network layer services supporting a variety of network layer protocols.
6. PPP provides connections over multiple links.
7. PPP provides network address configuration. This is particularly useful when a home user needs a temporary network address to connect to the Internet.

On the other hand, to keep PPP simple, several services are missing:

1. PPP does not provide flow control. A sender can send several frames one after another with no concern about overwhelming the receiver.
2. PPP has a very simple mechanism for error control. A CRC field is used to detect errors. If the frame is corrupted, it is silently discarded; the upper-layer protocol needs to take care of the problem. Lack of error control and sequence numbering may cause a packet to be received out of order.
3. PPP does not provide a sophisticated addressing mechanism to handle frames in a multipoint configuration.

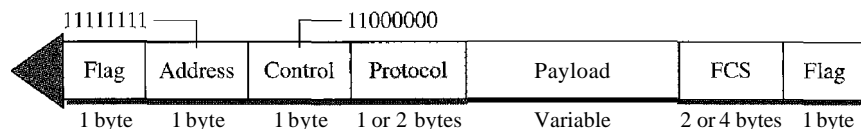
## Framing

PPP is a byte-oriented protocol. Framing is done according to the discussion of byte-oriented protocols at the beginning of this chapter.

### Frame Format

Figure 11.32 shows the format of a PPP frame. The description of each field follows:

Figure 11.32 PPP frame format



- **Flag.** A PPP frame starts and ends with a 1-byte flag with the bit pattern 01111110. Although this pattern is the same as that used in HDLC, there is a big difference. PPP is a byte-oriented protocol; HDLC is a bit-oriented protocol. The flag is treated as a byte, as we will explain later.
- **Address.** The address field in this protocol is a constant value and set to 11111111 (broadcast address). During negotiation (discussed later), the two parties may agree to omit this byte.
- **Control.** This field is set to the constant value 11000000 (imitating unnumbered frames in HDLC). As we will discuss later, PPP does not provide any flow control. Error control is also limited to error detection. This means that this field is not needed at all, and again, the two parties can agree, during negotiation, to omit this byte.
- **Protocol.** The protocol field defines what is being carried in the data field: either user data or other information. We discuss this field in detail shortly. This field is by default 2 bytes long, but the two parties can agree to use only 1 byte.
- **Payload field.** This field carries either the user data or other information that we will discuss shortly. The data field is a sequence of bytes with the default of a maximum of 1500 bytes; but this can be changed during negotiation. The data field is byte-stuffed if the flag byte pattern appears in this field. Because there is no field defining the size of the data field, padding is needed if the size is less than the maximum default value or the maximum negotiated value.
- **FCS.** The frame check sequence (FCS) is simply a 2-byte or 4-byte standard CRC.

### Byte Stuffing

The similarity between PPP and HDLe ends at the frame format. PPP, as we discussed before, is a byte-oriented protocol totally different from HDLC. As a byte-oriented protocol, the flag in PPP is a byte and needs to be escaped whenever it appears in the data section of the frame. The escape byte is 01111101, which means that every time the flaglike pattern appears in the data, this extra byte is stuffed to tell the receiver that the next byte is not a flag.

---

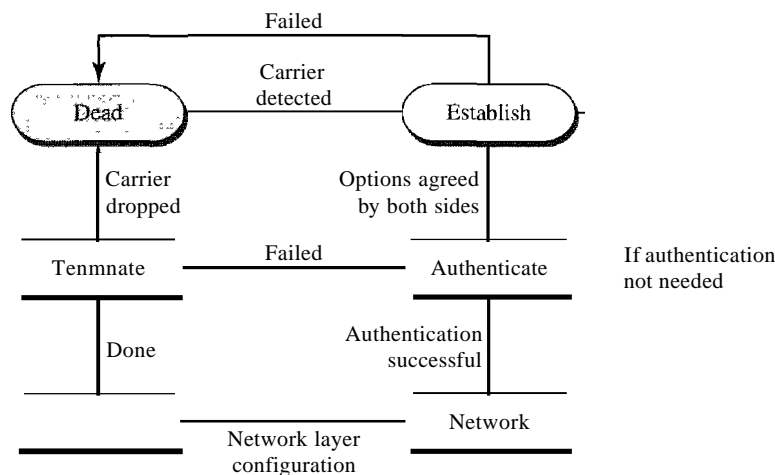
**PPP** is a byte-oriented protocol using byte stuffing with the escape byte 01111101.

---

### Transition Phases

A PPP connection goes through phases which can be shown in a transition phase diagram (see Figure 11.33).

Figure 11.33 *Transition phases*



- D **Dead.** In the dead phase the link is not being used. There is no active carrier (at the physical layer) and the line is quiet.
- D **Establish.** When one of the nodes starts the communication, the connection goes into this phase. In this phase, options are negotiated between the two parties. If the negotiation is successful, the system goes to the authentication phase (if authentication is required) or directly to the networking phase. The link control protocol packets, discussed shortly, are used for this purpose. Several packets may be exchanged here.
- D **Authenticate.** The authentication phase is optional; the two nodes may decide, during the establishment phase, not to skip this phase. However, if they decide to proceed with authentication, they send several authentication packets, discussed later. If the result is successful, the connection goes to the networking phase; otherwise, it goes to the termination phase.
- D **Network.** In the network phase, negotiation for the network layer protocols takes place. PPP specifies that two nodes establish a network layer agreement before data at

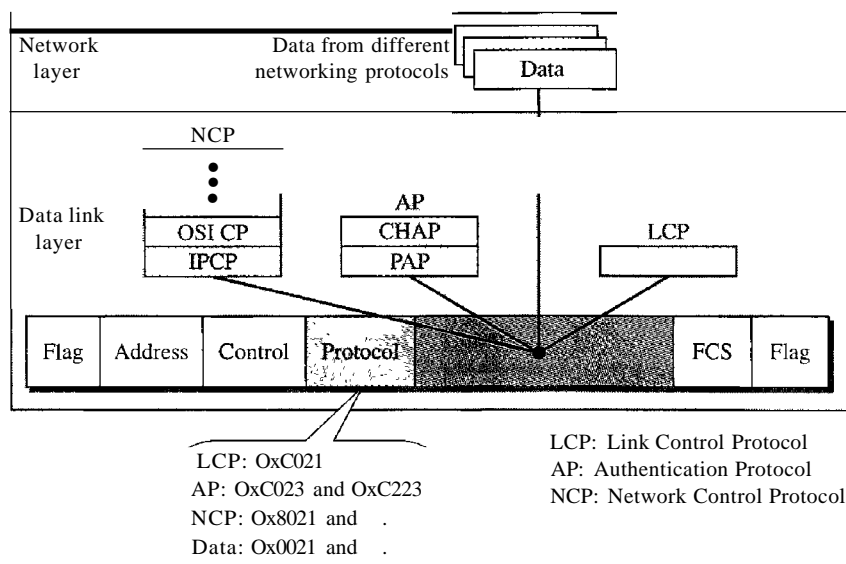
the network layer can be exchanged. The reason is that PPP supports multiple protocols at the network layer. If a node is running multiple protocols simultaneously at the network layer, the receiving node needs to know which protocol will receive the data.

- **Open.** In the open phase, data transfer takes place. When a connection reaches this phase, the exchange of data packets can be started. The connection remains in this phase until one of the endpoints wants to terminate the connection.
- **Terminate.** In the termination phase the connection is terminated. Several packets are exchanged between the two ends for house cleaning and closing the link.

## Multiplexing

Although PPP is a data link layer protocol, PPP uses another set of other protocols to establish the link, authenticate the parties involved, and carry the network layer data. Three sets of protocols are defined to make PPP powerful: the Link Control Protocol (LCP), two Authentication Protocols (APs), and several Network Control Protocols (NCPs). At any moment, a PPP packet can carry data from one of these protocols in its data field, as shown in Figure 11.34. Note that there is one LCP, two APs, and several NCPs. Data may also come from several different network layers.

Figure 11.34 Multiplexing in PPP



### Link Control Protocol

The **Link Control Protocol (LCP)** is responsible for establishing, maintaining, configuring, and terminating links. It also provides negotiation mechanisms to set options between the two endpoints. Both endpoints of the link must reach an agreement about the options before the link can be established. See Figure 11.35.

All LCP packets are carried in the payload field of the PPP frame with the protocol field set to C021 in hexadecimal.

The code field defines the type of LCP packet. There are 11 types of packets as shown in Table 11.2.

Figure 11.35 LCP packet encapsulated in a frame

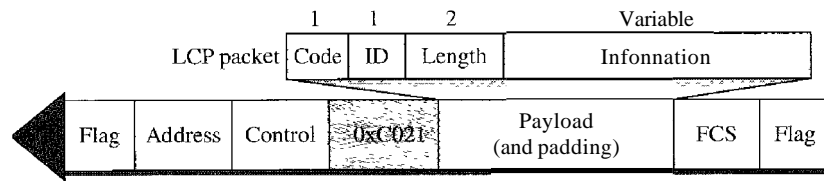


Table 11.2 LCP packets

<i>Code</i>	<i>Packet Type</i>	<i>Description</i>
0x01	Configure-request	Contains the list of proposed options and their values
0x02	Configure-ack	Accepts all options proposed
0x03	Configure-nak	Announces that some options are not acceptable
0x04	Configure-reject	Announces that some options are not recognized
0x05	Terminate-request	Request to shut down the line
0x06	Terminate-ack	Accept the shutdown request
0x07	Code-reject	Announces an unknown code
0x08	Protocol-reject	Announces an unknown protocol
0x09	Echo-request	A type of hello message to check if the other end is alive
0x0A	Echo-reply	The response to the echo-request message
0x0B	Discard-request	A request to discard the packet

There are three categories of packets. The first category, comprising the first four packet types, is used for link configuration during the establish phase. The second category, comprising packet types 5 and 6, is used for link termination during the termination phase. The last five packets are used for link monitoring and debugging.

The ID field holds a value that matches a request with a reply. One endpoint inserts a value in this field, which will be copied into the reply packet. The length field defines the length of the entire LCP packet. The information field contains information, such as options, needed for some LCP packets.

There are many options that can be negotiated between the two endpoints. Options are inserted in the information field of the configuration packets. In this case, the information field is divided into three fields: option type, option length, and option data. We list some of the most common options in Table 11.3.

Table 11.3 Common options

<i>Option</i>	<i>Default</i>
Maximum receive unit (payload field size)	1500
Authentication protocol	None
Protocol field compression	Off
Address and control field compression	Off

*Authentication Protocols*

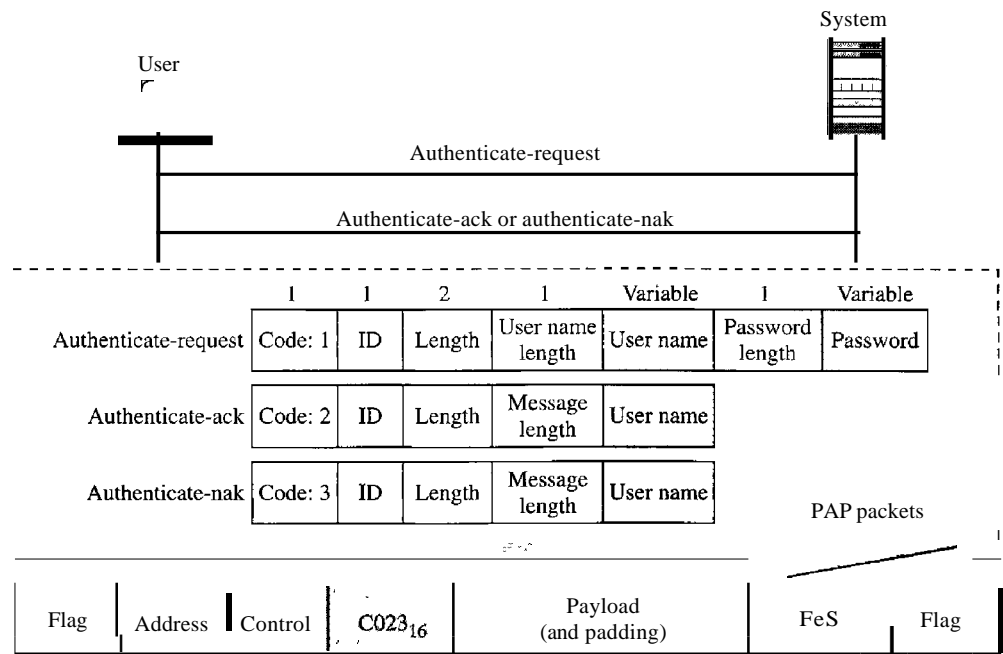
Authentication plays a very important role in PPP because PPP is designed for use over dial-up links where verification of user identity is necessary. **Authentication** means validating the identity of a user who needs to access a set of resources. PPP has created two protocols for authentication: Password Authentication Protocol and Challenge Handshake Authentication Protocol. Note that these protocols are used during the authentication phase.

**PAP** The **Password Authentication Protocol (PAP)** is a simple authentication procedure with a two-step process:

1. The user who wants to access a system sends an authentication identification (usually the user name) and a password.
2. The system checks the validity of the identification and password and either accepts or denies connection.

Figure 11.36 shows the three types of packets used by PAP and how they are actually exchanged. When a PPP frame is carrying any PAP packets, the value of the protocol field is OxCO23. The three PAP packets are authenticate-request, authenticate-ack, and authenticate-nak. The first packet is used by the user to send the user name and password. The second is used by the system to allow access. The third is used by the system to deny access.

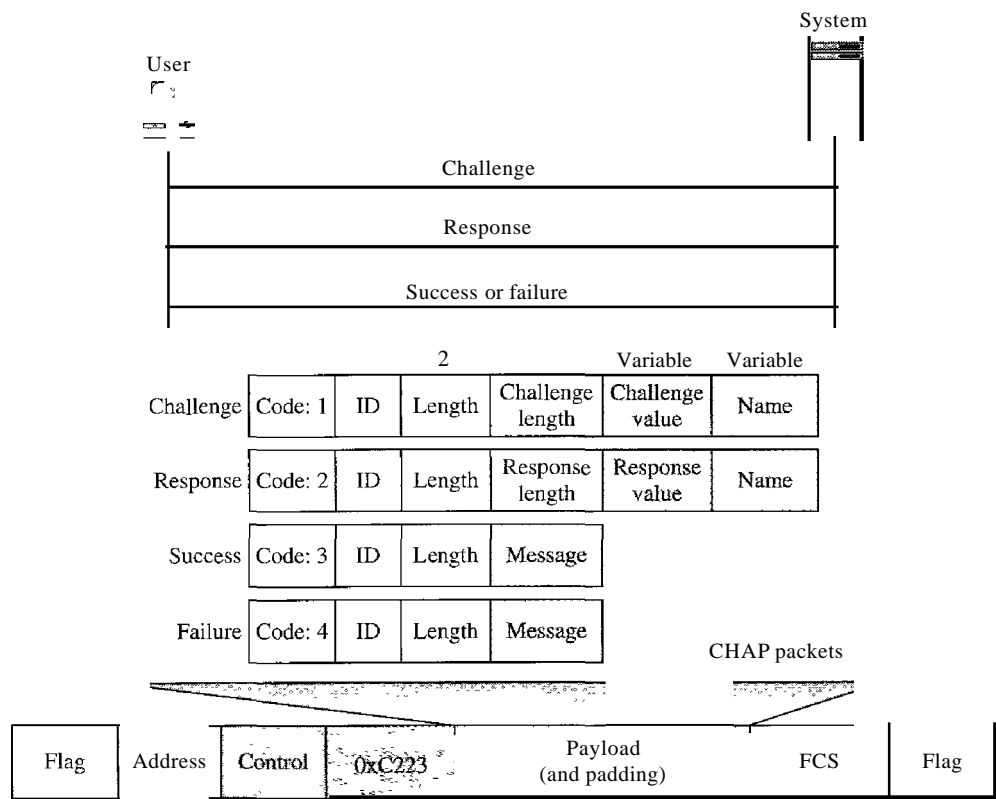
**Figure 11.36** PAP packets encapsulated in a PPP frame



**CHAP** The **Challenge Handshake Authentication Protocol (CHAP)** is a three-way hand-shaking authentication protocol that provides greater security than PAP. In this method, the password is kept secret; it is never sent online.

1. The system sends the user a challenge packet containing a challenge value, usually a few bytes.
2. The user applies a predefined function that takes the challenge value and the user's own password and creates a result. The user sends the result in the response packet to the system.
3. The system does the same. It applies the same function to the password of the user (known to the system) and the challenge value to create a result. If the result created is the same as the result sent in the response packet, access is granted; otherwise, it is denied. CHAP is more secure than PAP, especially if the system continuously changes the challenge value. Even if the intruder learns the challenge value and the result, the password is still secret. Figure 11.37 shows the packets and how they are used.

Figure 11.37 CHAP packets encapsulated in a PPP frame



CHAP packets are encapsulated in the PPP frame with the protocol value C223 in hexadecimal. There are four CHAP packets: challenge, response, success, and failure. The first packet is used by the system to send the challenge value. The second is used by the user to return the result of the calculation. The third is used by the system to allow access to the system. The fourth is used by the system to deny access to the system.

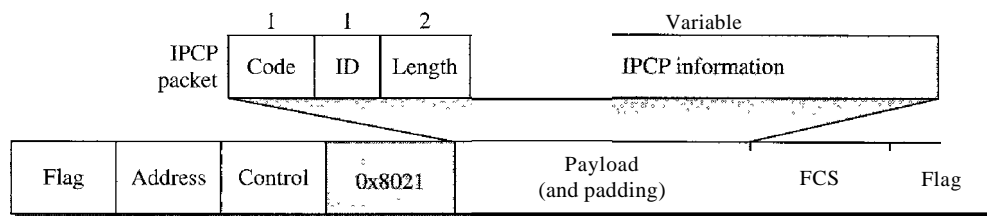
### Network Control Protocols

PPP is a multiple-network layer protocol. It can carry a network layer data packet from protocols defined by the Internet, OSI, Xerox, DECnet, AppleTalk, Novel, and so on.

To do this, PPP has defined a specific Network Control Protocol for each network protocol. For example, IPCP (Internet Protocol Control Protocol) configures the link for carrying IP data packets. Xerox CP does the same for the Xerox protocol data packets, and so on. Note that none of the NCP packets carry network layer data; they just configure the link at the network layer for the incoming data.

**IPCP** One NCP protocol is the Internet Protocol Control Protocol (IPCP). This protocol configures the link used to carry IP packets in the Internet. IPCP is especially of interest to us. The format of an IPCP packet is shown in Figure 11.38. Note that the value of the protocol field in hexadecimal is 8021.

Figure 11.38 *fPCP packet encapsulated in PPP frame*



IPCP defines seven packets, distinguished by their code values, as shown in Table 11.4.

Table 11.4 *Code value for IPCP packets*

Code	IPCP Packet
0x01	Configure-request
0x02	Configure-ack
0x03	Configure-nak
0x04	Configure-reject
0x05	Terminate-request
0x06	Terminate-ack
0x07	Code-reject

**Other Protocols** There are other NCP protocols for other network layer protocols. The OSI Network Layer Control Protocol has a protocol field value of 8023; the Xerox NS IDP Control Protocol has a protocol field value of 8025; and so on. The value of the code and the format of the packets for these other protocols are the same as shown in Table 11.4.

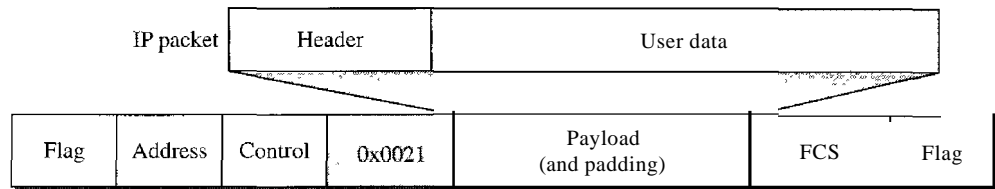
*Data/rom the Network Layer*

After the network layer configuration is completed by one of the NCP protocols, the users can exchange data packets from the network layer. Here again, there are different



protocol fields for different network layers. For example, if PPP is carrying data from the IP network layer, the field value is 0021 (note that the three rightmost digits are the same as for IPCP). If PPP is carrying data from the OSI network layer, the value of the protocol field is 0023, and so on. Figure 11.39 shows the frame for IP.

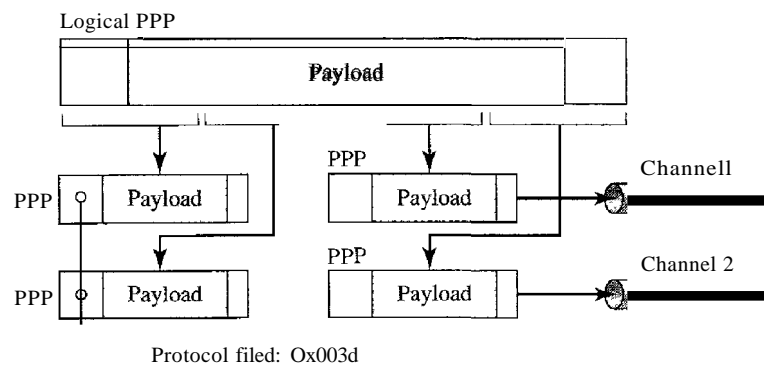
Figure 11.39 IP datagram encapsulated in a PPP frame



## Multilink PPP

PPP was originally designed for a single-channel point-to-point physical link. The availability of multiple channels in a single point-to-point link motivated the development of Multilink PPP. In this case, a logical PPP frame is divided into several actual PPP frames. A segment of the logical frame is carried in the payload of an actual PPP frame, as shown in Figure 11.40. To show that the actual PPP frame is carrying a fragment of a

Figure 11.40 Multilink PPP

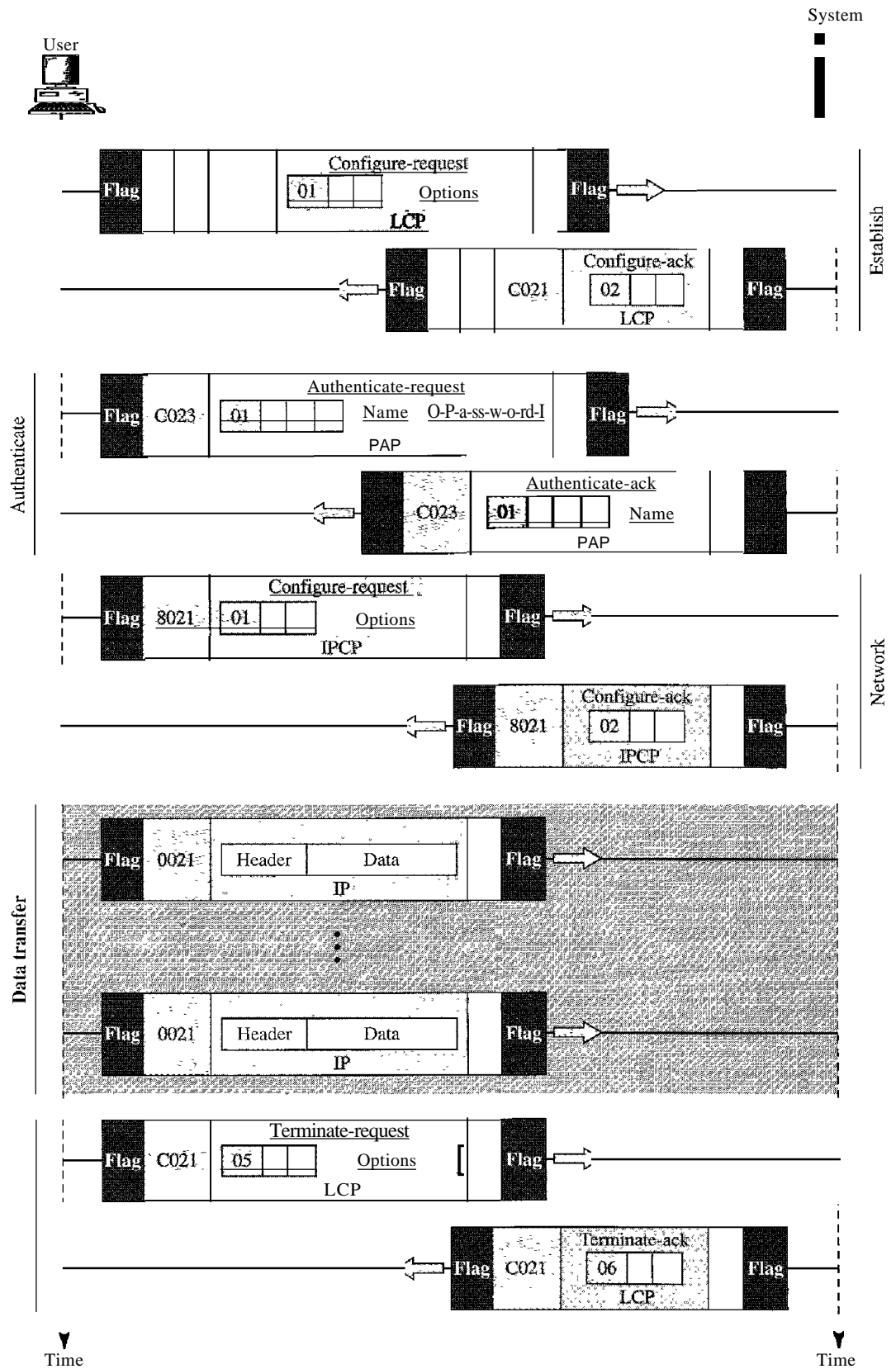


logical PPP frame, the protocol field is set to 0x003d. This new development adds complexity. For example, a sequence number needs to be added to the actual PPP frame to show a fragment's position in the logical frame.

### Example 11.12

Let us go through the phases followed by a network layer packet as it is transmitted through a PPP connection. Figure 11.41 shows the steps. For simplicity, we assume unidirectional movement of data from the user site to the system site (such as sending an e-mail through an ISP).

Figure 11.41 An example



The first two frames show link establishment. We have chosen two options (not shown in the figure): using PAP for authentication and suppressing the address control fields. Frames 3 and 4 are for authentication. Frames 5 and 6 establish the network layer connection using IPCP.

The next several frames show that some IP packets are encapsulated in the PPP frame. The system (receiver) may have been running several network layer protocols, but it knows that the incoming data must be delivered to the IP protocol because the NCP protocol used before the data transfer was IPCP.

After data transfer, the user then terminates the data link connection, which is acknowledged by the system. Of COURse the user or the system could have chosen to terminate the network layer IPCP and keep the data link layer running if it wanted to run another NCP protocol.

The example is trivial, but it points out the similarities of the packets in LCP, AP, and NCP. It also shows the protocol field values and code numbers for particular protocols.

## 11.8 RECOMMENDED READING

For more details about subjects discussed in this chapter, we recommend the following books. The items in brackets [...] refer to the reference list at the end of the text.

### Books

A discussion of data link control can be found in [GW04], Chapter 3 of [Tan03], Chapter 7 of [Sta04], Chapter 12 of [Kes97], and Chapter 2 of [PD03]. More advanced materials can be found in [KMK04].

## 11.9 KEY TERMS

acknowledgment (ACK)	flow control
asynchronous balanced mode (ABM)	framing
automatic repeat request (ARQ)	<i>Go-Back-N</i> ARQ Protocol
bandwidth-delay product	High-level Data Link Control (HDLC)
bit-oriented protocol	information frame (I-frame)
bit stuffing	Internet Protocol Control Protocol (IPCP)
byte stuffing	Link Control Protocol (LCP)
Challenge Handshake Authentication Protocol (CHAP)	negative acknowledgment (NAK)
character-oriented protocol	noiseless channel
data link control	noisy channel
error control	normal response mode (NRM)
escape character (ESC)	Password Authentication Protocol (PAP)
event	piggybacking
fixed-size framing	pipelining
flag	Point-to-Point Protocol (PPP)
	primary station

receive sliding window	sliding window
secondary station	Stop-and-Wait ARQ Protocol
Selective Repeat ARQ Protocol	Stop-and-Wait Protocol
send sliding window	supervisory frame (S-frame)
sequence number	transition phase
Simplest Protocol	unnumbered frame (D-frame)
	variable-size framing

---

## 11.10 SUMMARY

- O Data link control deals with the design and procedures for communication between two adjacent nodes: node-to-node communication.
- O Framing in the data link layer separates a message from one source to a destination, or from other messages going from other sources to other destinations,
- O 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.
- O 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.
- O 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.
- O 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.
- O 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.
- O For the noiseless channel, we discussed two protocols: the Simplest Protocol and the Stop-and-Wait Protocol. The first protocol has neither flow nor error control; the second has no error control. In the Simplest Protocol, the sender sends its frames one after another with no regards to the receiver. In the Stop-and-Wait Protocol, the sender sends one frame, stops until it receives confirmation from the receiver, and then sends the next frame.
- D For the noisy channel, we discussed three protocols: Stop-and-Wait ARQ, 00-Back-N, and Selective Repeat ARQ. The Stop-and-Wait ARQ Protocol, adds a simple error control mechanism to the Stop-and-Wait Protocol. In the 00-Back-N ARQ Protocol, we can send several frames before receiving acknowledgments, improving the efficiency of transmission. In the Selective Repeat ARQ protocol we avoid unnecessary transmission by sending only frames that are corrupted.
- D Both 00-Back-N and Selective-Repeat Protocols use a sliding window. In 00-Back-N ARQ, if  $m$  is the number of bits for the sequence number, then the size of

the send window must be less than  $2^m$ ; the size of the receiver window is always 1. In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of  $2^m$ .

- O A technique called piggybacking is used to improve the efficiency of the bidirectional protocols. When a frame is carrying data from A to B, it can also carry control information about frames from B; when a frame is carrying data from B to A, it can also carry control information about frames from A.
- O 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.

## 11.11 PRACTICE SET

### Review Questions

1. Briefly describe the services provided by the data link layer.
2. Define framing and the reason for its need.
3. Compare and contrast byte-oriented and bit-oriented protocols. Which category has been popular in the past (explain the reason)? Which category is popular now (explain the reason)?
4. Compare and contrast byte-stuffing and bit-stuffing. Which technique is used in byte-oriented protocols? Which technique is used in bit-oriented protocols?
5. Compare and contrast flow control and error control.
6. What are the two protocols we discussed for noiseless channels in this chapter?
7. What are the three protocols we discussed for noisy channels in this chapter?
8. Explain the reason for moving from the Stop-and-Wait ARQ Protocol to the 00-Back-NARQ Protocol.
9. Compare and contrast the Go-Back-NARQ Protocol with Selective-RepeatARQ.
10. Compare and contrast HDLC with PPP. Which one is byte-oriented; which one is bit-oriented?
11. Define piggybacking and its usefulness.
12. Which of the protocols described in this chapter utilize pipelining?

### Exercises

13. Byte-stuff the data in Figure 11.42.

Figure 11.42 *Exercise 13*

ESC			Flag			ESC	ESC	ESC		Flag	
-----	--	--	------	--	--	-----	-----	-----	--	------	--

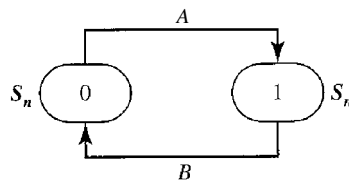
14. Bit-stuff the data in Figure 11.43.

**Figure 11.43** Exercise 14

10001111111001111101000111111111111000011111 |

15. Design two simple algorithms for byte-stuffing. The first adds bytes at the sender; the second removes bytes at the receiver.
16. Design two simple algorithms for bit-stuffing. The first adds bits at the sender; the second removes bits at the receiver.
17. A sender sends a series of packets to the same destination using 5-bit sequence numbers. If the sequence number starts with 0, what is the sequence number after sending 100 packets?
18. Using 5-bit sequence numbers, what is the maximum size of the send and receive windows for each of the following protocols?
- Stop-and-Wait ARQ
  - Go-Back-N ARQ
  - Selective-Repeat ARQ
19. Design a bidirectional algorithm for the Simplest Protocol using piggybacking. Note that the both parties need to use the same algorithm.
20. Design a bidirectional algorithm for the Stop-and-Wait Protocol using piggybacking. Note that both parties need to use the same algorithm.
21. Design a bidirectional algorithm for the Stop-and-Wait ARQ Protocol using piggybacking. Note that both parties need to use the same algorithm.
22. Design a bidirectional algorithm for the Go-Back-N ARQ Protocol using piggybacking. Note that both parties need to use the same algorithm.
23. Design a bidirectional algorithm for the Selective-Repeat ARQ Protocol using piggybacking. Note that both parties need to use the same algorithm.
24. Figure 11.44 shows a state diagram to simulate the behavior of Stop-and-Wait ARQ at the sender site.

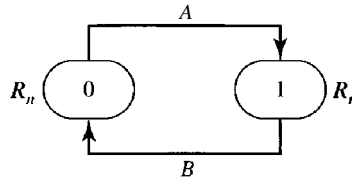
**Figure 11.44** Exercise 24



The states have a value of  $S_n$  (0 or 1). The arrows show the transitions. Explain the events that cause the two transitions labeled A and B.

25. Figure 11.45 shows a state diagram to simulate the behavior of Stop-and-Wait ARQ at the receiver site.

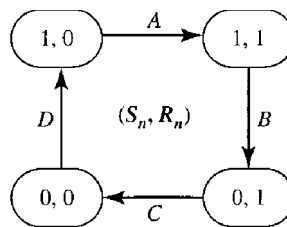
Figure 11.45 Exercise 25



The states have a value of  $R_n$  (0 or 1). The arrows shows the transitions. Explain the events that cause the two transitions labeled A and B.

26. In Stop-and-Wait ARQ, we can combine the state diagrams of the sender and receiver in Exercises 24 and 25. One state defines the combined values of  $R_n$  and  $S_n$ . This means that we can have four states, each defined by  $(x, y)$ , where  $x$  defines the value of  $S_n$  and  $y$  defines the value of  $R_n$ . In other words, we can have the four states shown in Figure 11.46. Explain the events that cause the four transitions labeled A, B, C, and D.

Figure 11.46 Exercise 26



27. The timer of a system using the Stop-and-Wait ARQ Protocol has a time-out of 6 ms. Draw the flow diagram similar to Figure 11.11 for four frames if the round trip delay is 4 ms. Assume no data frame or control frame is lost or damaged.
28. Repeat Exercise 27 if the time-out is 4 ms and the round trip delay is 6.
29. Repeat Exercise 27 if the first frame (frame 0) is lost.
30. A system uses the Stop-and-Wait ARQ Protocol. If each packet carries 1000 bits of data, how long does it take to send 1 million bits of data if the distance between the sender and receiver is 5000 Km and the propagation speed is  $2 \times 10^8$  m? Ignore transmission, waiting, and processing delays. We assume no data or control frame is lost or damaged.
31. Repeat Exercise 30 using the *Go-back-N* ARQ Protocol with a window size of 7. Ignore the overhead due to the header and trailer.
32. Repeat Exercise 30 using the Selective-Repeat ARQ Protocol with a window size of 4. Ignore the overhead due to the header and the trailer.