3.1.1 Services Provided to the Network Layer

The function of the data link layer is to provide services to the network layer. The principal service is transferring data from the network layer on the source machine to the network layer on the destination machine. On the source machine is an entity, call it a process, in the network layer that hands some bits to the data

link layer for transmission to the destination. The job of the data link layer is to transmit the bits to the destination machine so they can be handed over to the network layer there, as shown in Fig. 3-2(a). The actual transmission follows the path of Fig. 3-2(b), but it is easier to think in terms of two data link layer processes communicating using a data link protocol. For this reason, we will implicitly use the model of Fig. 3-2(a) throughout this chapter.

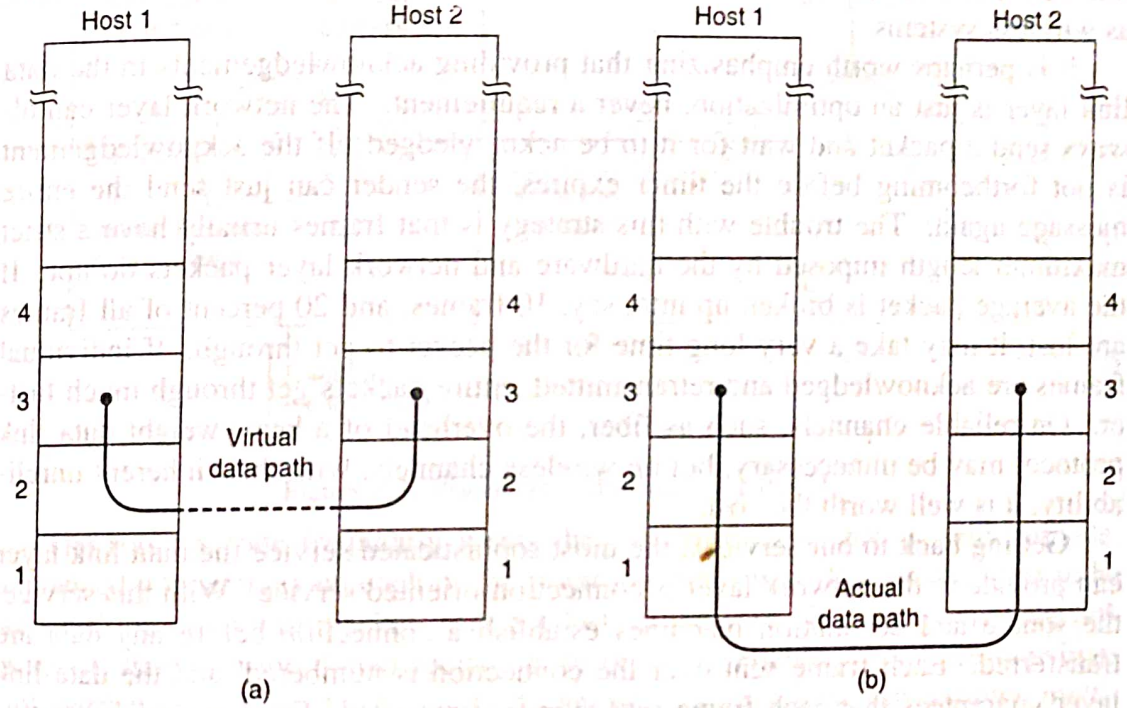

Figure 3-2. (a) Virtual communication. (b) Actual communication.

The data link layer can be designed to offer various services. The actual services offered can vary from system to system. Three reasonable possibilities that are commonly provided are

1. Unacknowledged connectionless service.
2. Acknowledged connectionless service.
3. Acknowledged connection-oriented service.

Let us consider each of these in turn.

Unacknowledged connectionless service consists of having the source machine send independent frames to the destination machine without having the destination machine acknowledge them. No logical connection is established beforehand or released afterward. If a frame is lost due to noise on the line, no attempt is made to detect the loss or recover from it in the data link layer. This class of service is appropriate when the error rate is very low so that recovery is left to higher layers. It is also appropriate for real-time traffic, such as voice, in

which late data are worse than bad data. Most LANs use unacknowledged connectionless service in the data link layer.

The next step up in terms of reliability is acknowledged connectionless service. When this service is offered, there are still no logical connections used, but each frame sent is individually acknowledged. In this way, the sender knows whether a frame has arrived correctly. If it has not arrived within a specified time interval, it can be sent again. This service is useful over unreliable channels, such as wireless systems.

It is perhaps worth emphasizing that providing acknowledgements in the data link layer is just an optimization, never a requirement. The network layer can always send a packet and wait for it to be acknowledged. If the acknowledgement is not forthcoming before the timer expires, the sender can just send the entire message again. The trouble with this strategy is that frames usually have a strict maximum length imposed by the hardware and network layer packets do not. If the average packet is broken up into, say, 10 frames, and 20 percent of all frames are lost, it may take a very long time for the packet to get through. If individual frames are acknowledged and retransmitted, entire packets get through much faster. On reliable channels, such as fiber, the overhead of a heavyweight data link protocol may be unnecessary, but on wireless channels, with their inherent unreliability, it is well worth the cost.

Getting back to our services, the most sophisticated service the data link layer can provide to the network layer is connection-oriented service. With this service, the source and destination machines establish a connection before any data are transferred. Each frame sent over the connection is numbered, and the data link layer guarantees that each frame sent is indeed received. Furthermore, it guarantees that each frame is received exactly once and that all frames are received in the right order. With connectionless service, in contrast, it is conceivable that a lost acknowledgement causes a packet to be sent several times and thus received several times. Connection-oriented service, in contrast, provides the network layer processes with the equivalent of a reliable bit stream.

When connection-oriented service is used, transfers go through three distinct phases. In the first phase, the connection is established by having both sides initialize variables and counters needed to keep track of which frames have been received and which ones have not. In the second phase, one or more frames are actually transmitted. In the third and final phase, the connection is released, freeing up the variables, buffers, and other resources used to maintain the connection.

Consider a typical example: a WAN subnet consisting of routers connected by point-to-point leased telephone lines. When a frame arrives at a router, the hardware checks it for errors (using techniques we will study late in this chapter), then passes the frame to the data link layer software (which might be embedded in a chip on the network interface board). The data link layer software checks to see if this is the frame expected, and if so, gives the packet contained in the payload field to the routing software. The routing software then chooses the appropriate

outgoing line and passes the packet back down to the data link layer software, which then transmits it. The flow over two routers is shown in Fig. 3-3.

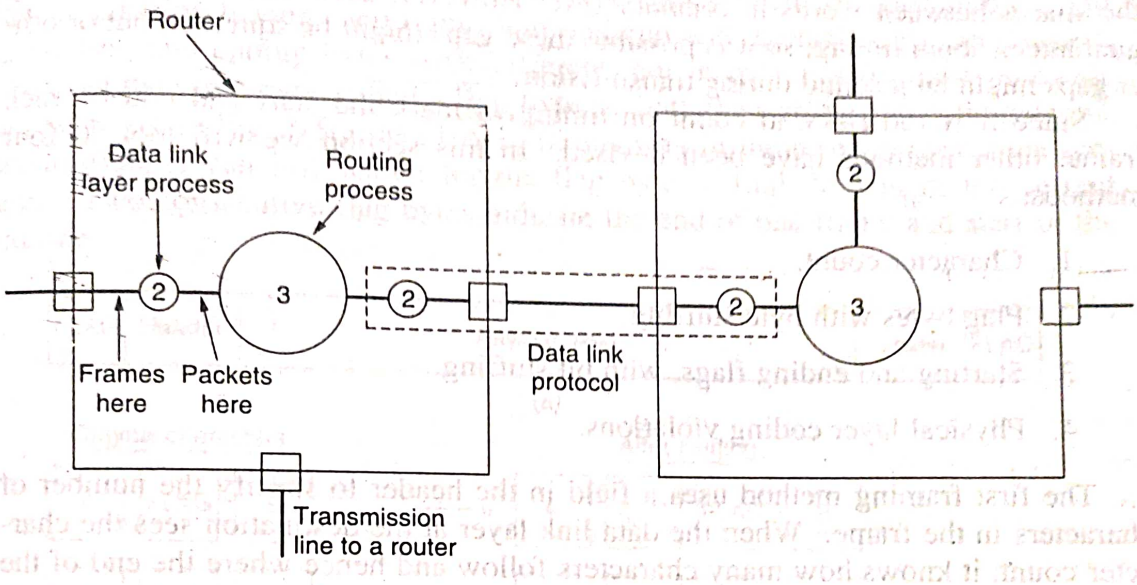

Figure 3-3. Placement of the data link protocol.

The routing code frequently wants the job done right, that is, with reliable, sequenced connections on each of the point-to-point lines. It does not want to be bothered too often with packets that got lost on the way. It is up to the data link protocol, shown in the dotted rectangle, to make unreliable communication lines look perfect or, at least, fairly good. As an aside, although we have shown multiple copies of the data link layer software in each router, in fact, one copy handles all the lines, with different tables and data structures for each one.

3.1.2 Framing

To provide service to the network layer, the data link layer must use the service provided to it by the physical layer. What the physical layer does is accept a raw bit stream and attempt to deliver it to the destination. This bit stream is not guaranteed to be error free. The number of bits received may be less than, equal to, or more than the number of bits transmitted, and they may have different values. It is up to the data link layer to detect and, if necessary, correct errors.

The usual approach is for the data link layer to break the bit stream up into discrete frames and compute the checksum for each frame. (Checksum algorithms will be discussed later in this chapter.) When a frame arrives at the destination, the checksum is recomputed. If the newly-computed checksum is different from the one contained in the frame, the data link layer knows that an error has occurred and takes steps to deal with it (e.g., discarding the bad frame and possibly also sending back an error report).

Breaking the bit stream up into frames is more difficult than it at first appears. One way to achieve this framing is to insert time gaps between frames, much like the spaces between words in ordinary text. However, networks rarely make any guarantees about timing, so it is possible these gaps might be squeezed out or other gaps might be inserted during transmission.

Since it is too risky to count on timing to mark the start and end of each frame, other methods have been devised. In this section we will look at four methods:

1. Character count.
2. Flag bytes with byte stuffing.
3. Starting and ending flags, with bit stuffing.
4. Physical layer coding violations.

The first framing method uses a field in the header to specify the number of characters in the frame. When the data link layer at the destination sees the character count, it knows how many characters follow and hence where the end of the frame is. This technique is shown in Fig. 3-4(a) for four frames of sizes 5, 5, 8, and 8 characters, respectively.

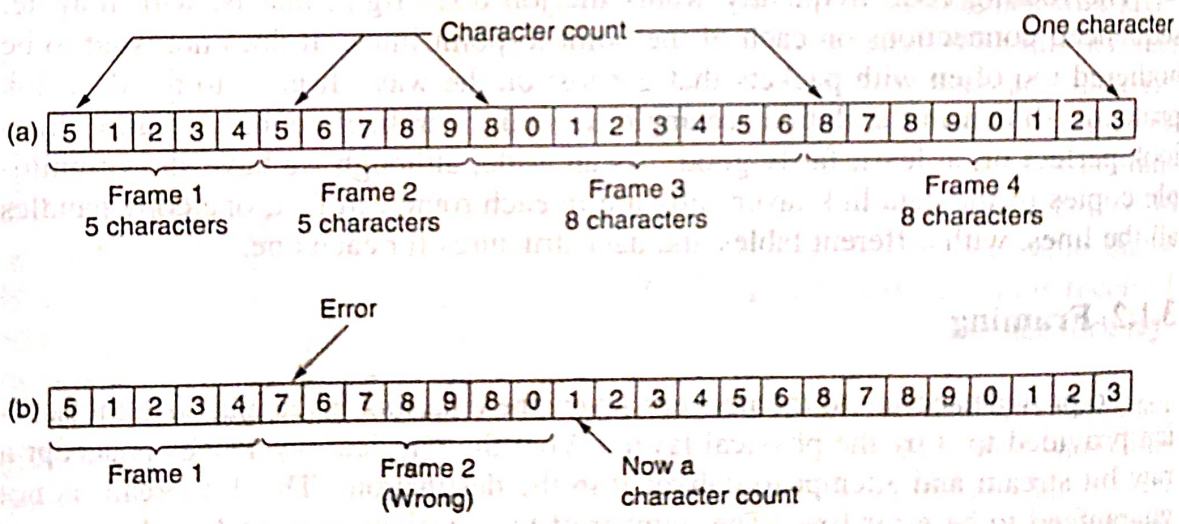

Figure 3-4. A character stream. (a) Without errors. (b) With one error.

The trouble with this algorithm is that the count can be garbled by a transmission error. For example, if the character count of 5 in the second frame of Fig. 3-4(b) becomes a 7, the destination will get out of synchronization and will be unable to locate the start of the next frame. Even if the checksum is incorrect so the destination knows that the frame is bad, it still has no way of telling where the next frame starts. Sending a frame back to the source asking for a retransmission does not help either, since the destination does not know how many characters to

skip over to get to the start of the retransmission. For this reason, the character count method is rarely used anymore.

The second framing method gets around the problem of resynchronization after an error by having each frame start and end with special bytes. In the past, the starting and ending bytes were different, but in recent years most protocols have used the same byte, called a **flag byte**, as both the starting and ending delimiter, as shown in Fig. 3-5(a) as FLAG. In this way, if the receiver ever loses synchronization, it can just search for the flag byte to find the end of the current frame. Two consecutive flag bytes indicate the end of one frame and start of the next one.

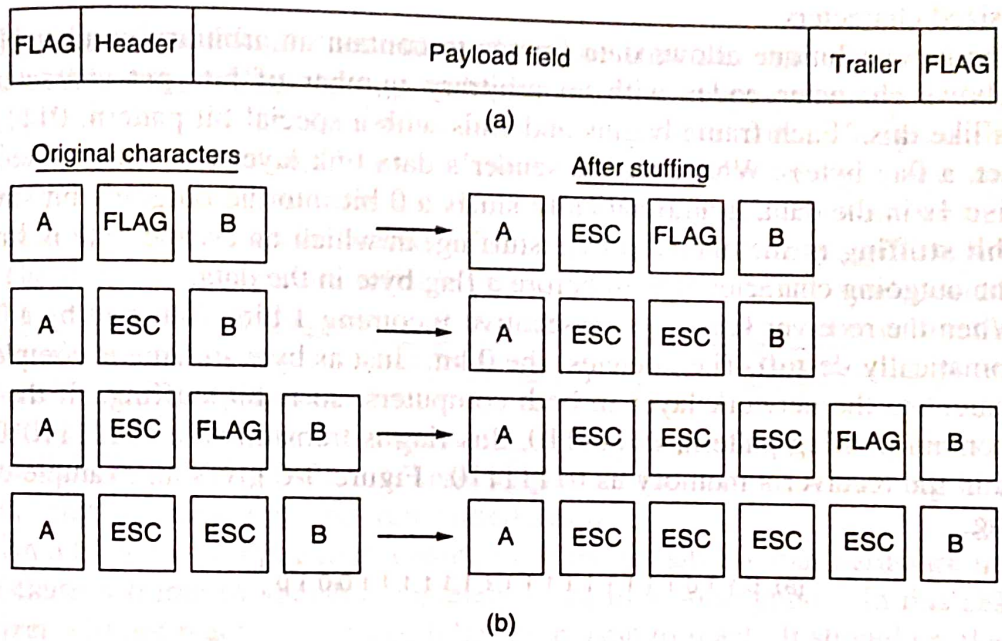

Figure 3-5. (a) A frame delimited by flag bytes. (b) Four examples of byte sequences before and after byte stuffing.

A serious problem occurs with this method when binary data, such as object programs or floating-point numbers, are being transmitted. It may easily happen that the flag byte's bit pattern occurs in the data. This situation will usually interfere with the framing. One way to solve this problem is to have the sender's data link layer insert a special escape byte (ESC) just before each "accidental" flag byte in the data. The data link layer on the receiving end removes the escape byte before the data are given to the network layer. This technique is called **byte stuffing** or **character stuffing**. Thus, a framing flag byte can be distinguished from one in the data by the absence or presence of an escape byte before it.

Of course, the next question is: What happens if an escape byte occurs in the middle of the data? The answer is that it, too, is stuffed with an escape byte. Thus, any single escape byte is part of an escape sequence, whereas a doubled one indicates that a single escape occurred naturally in the data. Some examples are

shown in Fig. 3-5(b). In all cases, the byte sequence delivered after destuffing is exactly the same as the original byte sequence.

The byte-stuffing scheme depicted in Fig. 3-5 is a slight simplification of the one used in the PPP protocol that most home computers use to communicate with their Internet service provider. We will discuss PPP later in this chapter.

A major disadvantage of using this framing method is that it is closely tied to the use of 8-bit characters. Not all character codes use 8-bit characters. For example, UNICODE uses 16-bit characters, As networks developed, the disadvantages of embedding the character code length in the framing mechanism became more and more obvious, so a new technique had to be developed to allow arbitrary sized characters.

The new technique allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary number of bits per character. It works like this. Each frame begins and ends with a special bit pattern, 01111110 (in fact, a flag byte). Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a 0 bit into the outgoing bit stream. This **bit stuffing** is analogous to byte stuffing, in which an escape byte is stuffed into the outgoing character stream before a flag byte in the data.

When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically destuffs (i.e., deletes) the 0 bit. Just as byte stuffing is completely transparent to the network layer in both computers, so is bit stuffing. If the user data contain the flag pattern, 01111110, this flag is transmitted as 011111010 but stored in the receiver's memory as 01111110. Figure 3-6 gives an example of bit stuffing.

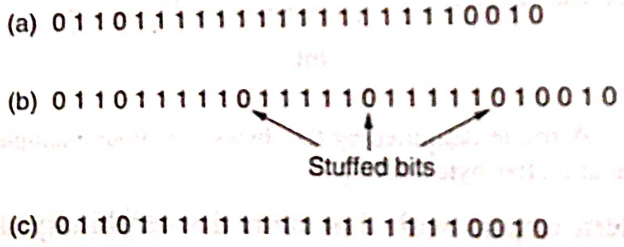

Figure 3-6. Bit stuffing. (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing.

With bit stuffing, the boundary between two frames can be unambiguously recognized by the flag pattern. Thus, if the receiver loses track of where it is, all it has to do is scan the input for flag sequences, since they can only occur at frame boundaries and never within the data.

The last method of framing is only applicable to networks in which the encoding on the physical medium contains some redundancy. For example, some LANs encode 1 bit of data by using 2 physical bits. Normally, a 1 bit is a high-low pair and a 0 bit is a low-high pair. The scheme means that every data bit has a transi-

tion in the middle, making it easy for the receiver to locate the bit boundaries. The combinations high-high and low-low are not used for data but are used for delimiting frames in some protocols.

As a final note on framing, many data link protocols use a combination of a character count with one of the other methods for extra safety. When a frame arrives, the count field is used to locate the end of the frame. Only if the appropriate delimiter is present at that position and the checksum is correct is the frame accepted as valid. Otherwise, the input stream is scanned for the next delimiter.

3.1.3 Error Control

Having solved the problem of marking the start and end of each frame, we come to the next problem: how to make sure all frames are eventually delivered to the network layer at the destination and in the proper order. Suppose that the sender just kept outputting frames without regard to whether they were arriving properly. This might be fine for unacknowledged connectionless service, but would most certainly not be fine for reliable, connection-oriented service.

The usual way to ensure reliable delivery is to provide the sender with some feedback about what is happening at the other end of the line. Typically, the protocol calls for the receiver to send back special control frames bearing positive or negative acknowledgements about the incoming frames. If the sender receives a positive acknowledgement about a frame, it knows the frame has arrived safely. On the other hand, a negative acknowledgement means that something has gone wrong, and the frame must be transmitted again.

An additional complication comes from the possibility that hardware troubles may cause a frame to vanish completely (e.g., in a noise burst). In this case, the receiver will not react at all, since it has no reason to react. It should be clear that a protocol in which the sender transmits a frame and then waits for an acknowledgement, positive or negative, will hang forever if a frame is ever lost due to, for example, malfunctioning hardware.

This possibility is dealt with by introducing timers into the data link layer. When the sender transmits a frame, it generally also starts a timer. The timer is set to expire after an interval long enough for the frame to reach the destination, be processed there, and have the acknowledgement propagate back to the sender. Normally, the frame will be correctly received and the acknowledgement will get back before the timer runs out, in which case the timer will be canceled.

However, if either the frame or the acknowledgement is lost, the timer will go off, alerting the sender to a potential problem. The obvious solution is to just transmit the frame again. However, when frames may be transmitted multiple times there is a danger that the receiver will accept the same frame two or more times and pass it to the network layer more than once. To prevent this from happening, it is generally necessary to assign sequence numbers to outgoing frames, so that the receiver can distinguish retransmissions from originals.

The whole issue of managing the timers and sequence numbers so as to ensure that each frame is ultimately passed to the network layer at the destination exactly once, no more and no less, is an important part of the data link layer's duties. Later in this chapter, we will look at a series of increasingly sophisticated examples to see how this management is done.

3.1.4 Flow Control

Another important design issue that occurs in the data link layer (and higher layers as well) is what to do with a sender that systematically wants to transmit frames faster than the receiver can accept them. This situation can easily occur when the sender is running on a fast (or lightly loaded) computer and the receiver is running on a slow (or heavily loaded) machine. The sender keeps pumping the frames out at a high rate until the receiver is completely swamped. Even if the transmission is error free, at a certain point the receiver will simply be unable to handle the frames as they arrive and will start to lose some. Clearly, something has to be done to prevent this situation.

Two approaches are commonly used. In the first one, **feedback-based flow control**, the receiver sends back information to the sender giving it permission to send more data or at least telling the sender how the receiver is doing. In the second one, **rate-based flow control**, the protocol has a built-in mechanism that limits the rate at which senders may transmit data, without using feedback from the receiver. In this chapter we will study feedback-based flow control schemes because rate-based schemes are never used in the data link layer. We will look at rate-based schemes in Chap. 5.

Various feedback-based flow control schemes are known, but most of them use the same basic principle. The protocol contains well-defined rules about when a sender may transmit the next frame. These rules often prohibit frames from being sent until the receiver has granted permission, either implicitly or explicitly. For example, when a connection is set up, the receiver might say: "You may send me n frames now, but after they have been sent, do not send any more until I have told you to continue." We will examine the details shortly.

3.4 SLIDING WINDOW PROTOCOLS

In the previous protocols, data frames were transmitted in one direction only. In most practical situations, there is a need for transmitting data in both directions. One way of achieving full-duplex data transmission is to have two separate communication channels and use each one for simplex data traffic (in different directions). If this is done, we have two separate physical circuits, each with a "forward" channel (for data) and a "reverse" channel (for acknowledgements). In both cases the bandwidth of the reverse channel is almost entirely wasted. In effect, the user is paying for two circuits but using only the capacity of one.

A better idea is to use the same circuit for data in both directions. After all, in protocols 2 and 3 it was already being used to transmit frames both ways, and the reverse channel has the same capacity as the forward channel. In this model the data frames from *A* to *B* are intermixed with the acknowledgement frames from *A* to *B*. By looking at the *kind* field in the header of an incoming frame, the receiver can tell whether the frame is data or acknowledgement.

Although interleaving data and control frames on the same circuit is an improvement over having two separate physical circuits, yet another improvement is possible. When a data frame arrives, instead of immediately sending a separate

control frame, the receiver restrains itself and waits until the network layer passes it the next packet. The acknowledgement is attached to the outgoing data frame (using the *ack* field in the frame header). In effect, the acknowledgement gets a free ride on the next outgoing data frame. The technique of temporarily delaying outgoing acknowledgements so that they can be hooked onto the next outgoing data frame is known as **piggybacking**.

The principal advantage of using piggybacking over having distinct acknowledgement frames is a better use of the available channel bandwidth. The *ack* field in the frame header costs only a few bits, whereas a separate frame would need a header, the acknowledgement, and a checksum. In addition, fewer frames sent means fewer "frame arrival" interrupts, and perhaps fewer buffers in the receiver, depending on how the receiver's software is organized. In the next protocol to be examined, the piggyback field costs only 1 bit in the frame header. It rarely costs more than a few bits.

However, piggybacking introduces a complication not present with separate acknowledgements. How long should the data link layer wait for a packet onto which to piggyback the acknowledgement? If the data link layer waits longer than the sender's timeout period, the frame will be retransmitted, defeating the whole purpose of having acknowledgements. If the data link layer were an oracle and could foretell the future, it would know when the next network layer packet was going to come in and could decide either to wait for it or send a separate acknowledgement immediately, depending on how long the projected wait was going to be. Of course, the data link layer cannot foretell the future, so it must resort to some ad hoc scheme, such as waiting a fixed number of milliseconds. If a new packet arrives quickly, the acknowledgement is piggybacked onto it; otherwise, if no new packet has arrived by the end of this time period, the data link layer just sends a separate acknowledgement frame.

The next three protocols are bidirectional protocols that belong to a class called **sliding window** protocols. The three differ among themselves in terms of efficiency, complexity, and buffer requirements, as discussed later. In these, as in all sliding window protocols, each outbound frame contains a sequence number, ranging from 0 up to some maximum. The maximum is usually $2^n - 1$ so the sequence number fits exactly in an n-bit field. The stop-and-wait sliding window protocol uses $n = 1$, restricting the sequence numbers to 0 and 1, but more sophisticated versions can use arbitrary n.

The essence of all sliding window protocols is that at any instant of time, the sender maintains a set of sequence numbers corresponding to frames it is permitted to send. These frames are said to fall within the **sending window**. Similarly, the receiver also maintains a **receiving window** corresponding to the set of frames it is permitted to accept. The sender's window and the receiver's window need not have the same lower and upper limits or even have the same size. In some protocols they are fixed in size, but in others they can grow or shrink over the course of time as frames are sent and received.

Although these protocols give the data link layer more freedom about the order in which it may send and receive frames, we have definitely not dropped the requirement that the protocol must deliver packets to the destination network layer in the same order they were passed to the data link layer on the sending machine. Nor have we changed the requirement that the physical communication channel is "wire-like," that is, it must deliver all frames in the order sent.

The sequence numbers within the sender's window represent frames that have been sent or can be sent but are as yet not acknowledged. Whenever a new packet arrives from the network layer, it is given the next highest sequence number, and the upper edge of the window is advanced by one. When an acknowledgement comes in, the lower edge is advanced by one. In this way the window continuously maintains a list of unacknowledged frames. Figure 3-13 shows an example.

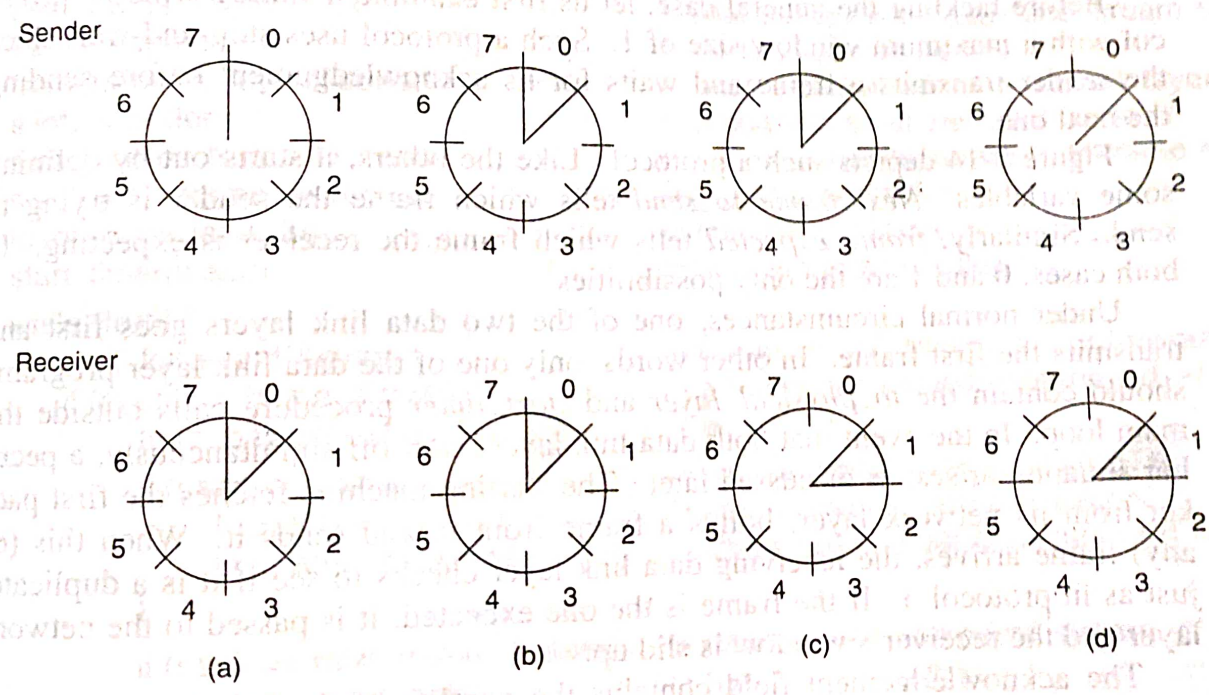

Figure 3-13. A sliding window of size 1, with a 3-bit sequence number. (a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgement has been received.

Since frames currently within the sender's window may ultimately be lost or damaged in transit, the sender must keep all these frames in its memory for possible retransmission. Thus, if the maximum window size is n, the sender needs n buffers to hold the unacknowledged frames. If the window ever grows to its maximum size, the sending data link layer must forcibly shut off the network layer until another buffer becomes free.

The receiving data link layer's window corresponds to the frames it may accept. Any frame falling outside the window is discarded without comment. When a frame whose sequence number is equal to the lower edge of the window

is received, it is passed to the network layer, an acknowledgement is generated, and the window is rotated by one. Unlike the sender's window, the receiver's window always remains at its initial size. Note that a window size of 1 means that the data link layer only accepts frames in order, but for larger windows this is not so. The network layer, in contrast, is always fed data in the proper order, regardless of the data link layer's window size.

Figure 3-13 shows an example with a maximum window size of 1. Initially, no frames are outstanding, so the lower and upper edges of the sender's window are equal, but as time goes on, the situation progresses as shown.