Second Edition

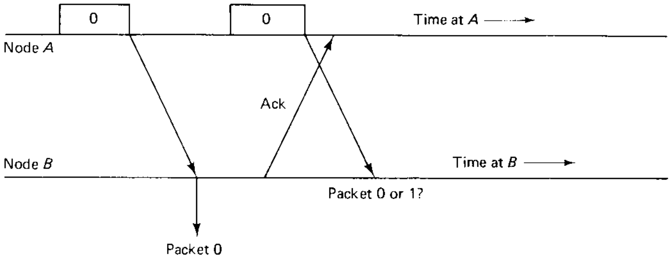
Data Networks

DIMITRI BERTSEKAS
Massachusetts Institute of Technology

ROBERT GALLAGER
Massachusetts Institute of Technology

PRENTICE HALL, Englewood Cliffs, New Jersey 07632

2



Point-to-Point Protocols and Links

2.1 INTRODUCTION

This chapter first provides an introduction to the physical communication links that constitute the building blocks of data networks. The major focus of the chapter is then data link control (*i.e.*, the point-to-point protocols needed to control the passage of data over a communication link). Finally, a number of point-to-point protocols at the network, transport, and physical layers are discussed. There are many similarities between the point-to-point protocols at these different layers, and it is desirable to discuss them together before addressing the more complex network-wide protocols for routing, flow control, and multiaccess control.

The treatment of physical links in Section 2.2 is a brief introduction to a very large topic. The reason for the brevity is not that the subject lacks importance or inherent interest, but rather, that a thorough understanding requires a background in linear system theory, random processes, and modern communication theory. In this section we provide a sufficient overview for those lacking this background and provide a review and perspective for those with more background.

In dealing with the physical layer in Section 2.2, we discuss both the actual communication channels used by the network and whatever interface modules are required at the ends of the channels to transmit and receive digital data (see Fig 2.1). We refer to these modules as modems (digital data modulators and demodulators), although in many cases no modulation or demodulation is required. In sending data, the modem converts the incoming binary data into a signal suitable for the channel. In receiving, the modem converts the received signal back into binary data. To an extent, the combination of modem, physical link, modem can be ignored; one simply recognizes that the combination appears to higher layers as a virtual bit pipe.

There are several different types of virtual bit pipes that might be implemented by the physical layer. One is the *synchronous* bit pipe, where the sending side of the data link control (DLC) module supplies bits to the sending side of the modem at a synchronous rate (*i.e.*, one bit each T seconds for some fixed T). If the DLC module temporarily has no data to send, it must continue to send dummy bits, called *idle fill*, until it again has data. The receiving modem recovers these bits synchronously (with delay and occasional errors) and releases the bits, including idle fill, to the corresponding DLC module.

The *intermittent synchronous* bit pipe is another form of bit pipe in which the sending DLC module supplies bits synchronously to the modem when it has data to send, but supplies nothing when it has no data. The sending modem sends no signal during these idle intervals, and the receiving modem detects the idle intervals and releases nothing to the receiving DLC module. This somewhat complicates the receiving modem, since it must distinguish between 0, 1, and idle in each time interval, and it must also regain synchronization at the end of an idle period. In Chapter 4 it will be seen that the capability to transmit nothing is very important for multiaccess channels.

A final form of virtual bit pipe is the *asynchronous character* pipe, where the bits within a character are sent at a fixed rate, but successive characters can be separated by

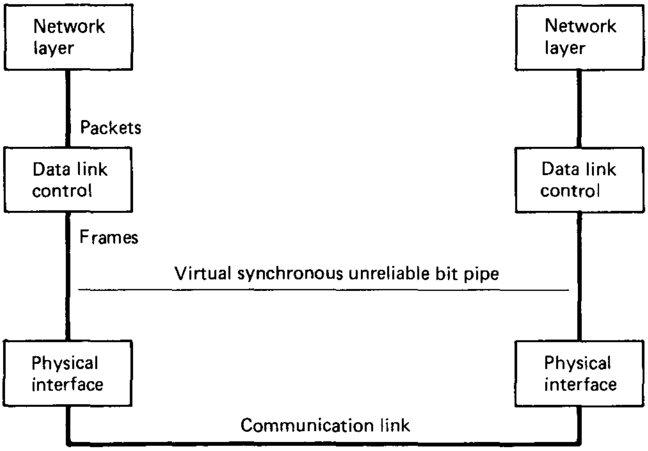


Figure 2.1 Data link control (DLC) layer with interfaces to adjacent layers.

variable delays, subject to a given minimum. As will be seen in the next section, this is used only when high data rates are not an important consideration.

In Sections 2.3 to 2.7 we treat the DLC layer, which is the primary focus of this chapter. For each point-to-point link, there are two DLC peer modules, one module at each end of the link. For traffic in a given direction, the sending DLC module receives packets from the network layer module at that node. The peer DLC modules employ a distributed algorithm, or protocol, to transfer these packets to the receiving DLC and thence to the network layer module at that node. In most cases, the objective is to deliver the packets in the order of arrival, with neither repetitions nor errors. In accomplishing this task, the DLC modules make use of the virtual bit pipe (with errors) provided by the physical layer.

One major problem in accomplishing the objective above is that of correcting the bit errors that occur on the virtual bit pipe. This objective is generally accomplished by a technique called ARQ (*a*utomatic *r*epeat re*q*uest). In this technique, errors are first detected at the receiving DLC module and then repetitions are requested from the transmitting DLC module. Both the detection of errors and the requests for retransmission require a certain amount of control overhead to be communicated on the bit pipe. This overhead is provided by adding a given number of bits (called a *header*) to the front of each packet and adding an additional number of bits (called a *trailer*) to the rear (see Fig. 2.2). A packet, extended by this header and trailer, is called a *frame*. From the standpoint of the DLC, a packet is simply a string of bits provided as a unit by the network layer; examples arise later where single "packets," in this sense, from the network layer contain pieces of information from many different sessions, but this makes no difference to the DLC layer.

Section 2.3 treats the problem of error detection and shows how a set of redundant bits in the trailer can be used for error detection. Section 2.4 then deals with retransmission requests. This is not as easy as it appears, first because the requests must be embedded into the data traveling in the opposite direction, and second because the opposite direction is also subject to errors. This provides our first exposure to a real distributed algorithm, which is of particular conceptual interest since it must operate in the presence of errors.

In Section 2.5 we discuss framing. The issue here is for the receiving DLC module to detect the beginning and end of each successive frame. For a synchronous bit pipe, the bits within a frame must contain the information to distinguish the end of the frame; also the idle fill between frames must be uniquely recognizable. This problem becomes even more interesting in the presence of errors.

A widely accepted standard of data link control is the HDLC protocol. This is discussed in Section 2.6, followed in Section 2.7 by a general discussion of how to initialize and to disconnect DLC protocols. This topic might appear to be trivial, but on closer inspection, it requires careful thought.

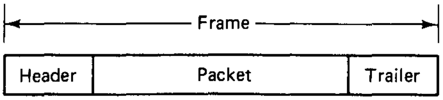


Figure 2.2 Frame structure. A packet from the network layer is extended in the DLC layer with control bits in front and in back of the packet.

Section 2.8 then treats a number of network layer issues, starting with addressing. End-to-end error recovery, which can be done at either the network or transport layer, is discussed next, along with a general discussion of why error recovery should or should not be done at more than one layer. The section ends with a discussion first of the X.25 network layer standard and next of the Internet Protocol (IP). IP was originally developed to connect the many local area networks in academic and research institutions to the ARPANET and is now a defacto standard for the internet sublayer.

A discussion of the transport layer is then presented in Section 2.9. This focuses on the transport protocol, TCP, used on top of IP. The combined use of TCP and IP (usually called TCP/IP) gives us an opportunity to explore some of the practical consequences of particular choices of protocols.

The chapter ends with an introduction to broadband integrated service data networks. In these networks, the physical layer is being implemented as a packet switching network called ATM, an abbreviation for "asynchronous transfer mode." It is interesting to compare how ATM performs its various functions with the traditional layers of conventional data networks.

2.2 THE PHYSICAL LAYER: CHANNELS AND MODEMS

As discussed in Section 2.1, the virtual channel seen by the data link control (DLC) layer has the function of transporting bits or characters from the DLC module at one end of the link to the module at the other end (see Fig. 2.1). In this section we survey how communication channels can be used to accomplish this function. We focus on point-to-point channels (*i.e.*, channels that connect just two nodes), and postpone consideration of multiaccess channels (*i.e.*, channels connecting more than two nodes) to Chapter 4. We also focus on communication in one direction, thus ignoring any potential interference between simultaneous transmission in both directions.

There are two broad classes of point-to-point channels: digital channels and analog channels. From a black box point of view, a digital channel is simply a bit pipe, with a bit stream as input and output. An analog channel, on the other hand, accepts a waveform (*i.e.*, an arbitrary function of time) as input and produces a waveform as output. We discuss analog channels first since digital channels are usually implemented on top of an underlying analog structure.

A module is required at the input of an analog channel to map the digital data from the DLC module into the waveform sent over the channel. Similarly, a module is required at the receiver to map the received waveform back into digital data. These modules will be referred to as modems (digital data *mo*dulator and *dem*odulator). The term *modem* is used broadly here, not necessarily implying any modulation but simply referring to the required mapping operations.

Let $s(t)$ denote the analog channel input as a function of time; $s(t)$ could represent a voltage or current waveform. Similarly, let $r(t)$ represent the voltage or current waveform at the output of the analog channel. The output $r(t)$ is a distorted, delayed, and attenuated version of $s(t)$, and our objective is to gain some intuitive appreciation of how to map the digital data into $s(t)$ so as to minimize the deleterious effects of this distortion. Note that a black box viewpoint of the physical channel is taken here, considering the input–output relation rather than the internal details of the analog channel. For example, if an ordinary voice telephone circuit (usually called a voice-grade circuit) is used as the analog channel, the physical path of the channel will typically pass through multiple switches, multiplexers, demultiplexers, modulators, and demodulators. For dial-up lines, this path will change on each subsequent call, although the specification of tolerances on the input–output characterization remain unchanged.

2.2.1 Filtering

One of the most important distorting effects on most analog channels is linear time-invariant filtering. Filtering occurs not only from filters inserted by the channel designer but also from the inherent behavior of the propagation medium. One effect of filtering is to "smooth out" the transmitted signal $s(t)$. Figure 2.3 shows two examples in which $s(t)$ is first a single rectangular pulse and then a sequence of rectangular pulses. The defining properties of linear time-invariant filters are as follows:

1. If input $s(t)$ yields output $r(t)$, then for any τ, input $s(t-\tau)$ yields output $r(t-\tau)$.
2. If $s(t)$ yields $r(t)$, then for any real number α, $\alpha s(t)$ yields $\alpha r(t)$.
3. If $s_1(t)$ yields $r_1(t)$ and $s_2(t)$ yields $r_2(t)$, then $s_1(t)+s_2(t)$ yields $r_1(t)+r_2(t)$.

The first property above expresses the time invariance: namely, if an input is delayed by τ, the corresponding output remains the same except for the delay of τ. The second and third properties express the linearity. That is, if an input is scaled by α, the corresponding output remains the same except for being scaled by α. Also, the

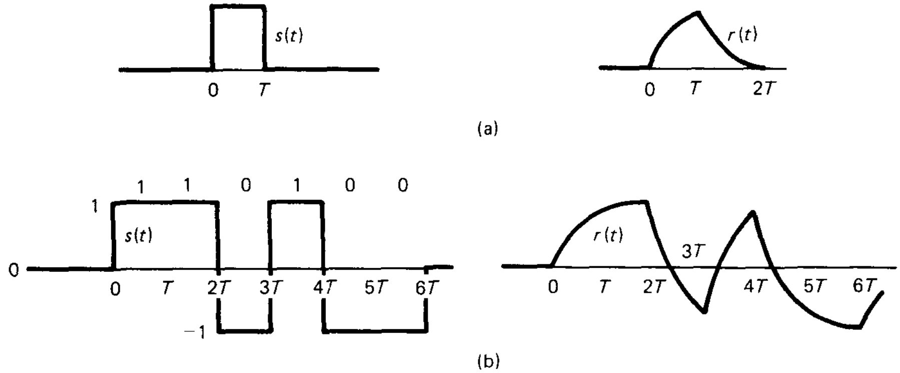


Figure 2.3 Relation of input and output waveforms for a communication channel with filtering. Part (a) shows the response $r(t)$ to an input $s(t)$ consisting of a rectangular pulse, and part (b) shows the response to a sequence of pulses. Part (b) also illustrates the NRZ code in which a sequence of binary inputs (1 1 0 1 0 0) is mapped into rectangular pulses. The duration of each pulse is equal to the time between binary inputs.

output due to the sum of two inputs is simply the sum of the corresponding outputs. As an example of these properties, the output for Fig. 2.3(b) can be calculated from the output in Fig. 2.3(a). In particular, the response to a negative-amplitude rectangular pulse is the negative of that from a positive-amplitude pulse (property 2), the output from a delayed pulse is delayed from the original output (property 1), and the output from the sum of the pulses is the sum of the outputs for the original pulses (property 3).

Figure 2.3 also illustrates a simple way to map incoming bits into an analog waveform $s(t)$. The virtual channel accepts a new bit from the DLC module each T seconds; the bit value 1 is mapped into a rectangular pulse of amplitude +1, and the value 0 into amplitude -1. Thus, Fig. 2.3(b) represents $s(t)$ for the data sequence 110100. This mapping scheme is often referred to as a nonreturn to zero (NRZ) code. The notation NRZ arises because of an alternative scheme in which the pulses in $s(t)$ have a shorter duration than the bit time T, resulting in $s(t)$ returning to 0 level for some interval between each pulse. The merits of these schemes will become clearer later.

Next, consider changing the rate at which bits enter the virtual channel. Figure 2.4 shows the effect of increasing the rate by a factor of 4 (*i.e.*, the signaling interval is reduced from T to $T/4$). It can be seen that output $r(t)$ is more distorted than before. The problem is that the response to a single pulse lasts much longer than a pulse time, so that the output at a given t depends significantly on the polarity of several input pulses; this phenomenon is called *intersymbol interference*.

From a more general viewpoint, suppose that $h(t)$ is the channel output corresponding to an infinitesimally narrow pulse of unit area at time 0. We call $h(t)$ the impulse response of the channel. Think of an arbitrary input waveform $s(t)$ as a superposition of very narrow pulses as shown in Fig. 2.5. The pulse from $s(\tau)$ to $s(\tau + \delta)$ can be viewed as a small impulse of area $\delta s(\tau)$ at time τ; this gives rise to the output $\delta s(\tau)h(t - \tau)$ at

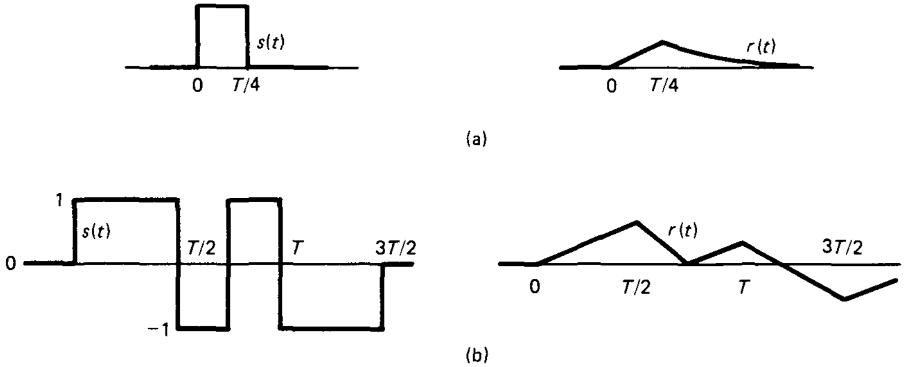


Figure 2.4 Relation of input and output waveforms for the same channel as in Fig. 2.3. Here the binary digits enter at 4 times the rate of Fig. 2.3, and the rectangular pulses last one-fourth as long. Note that the output $r(t)$ is more distorted and more attenuated than that in Fig. 2.3.

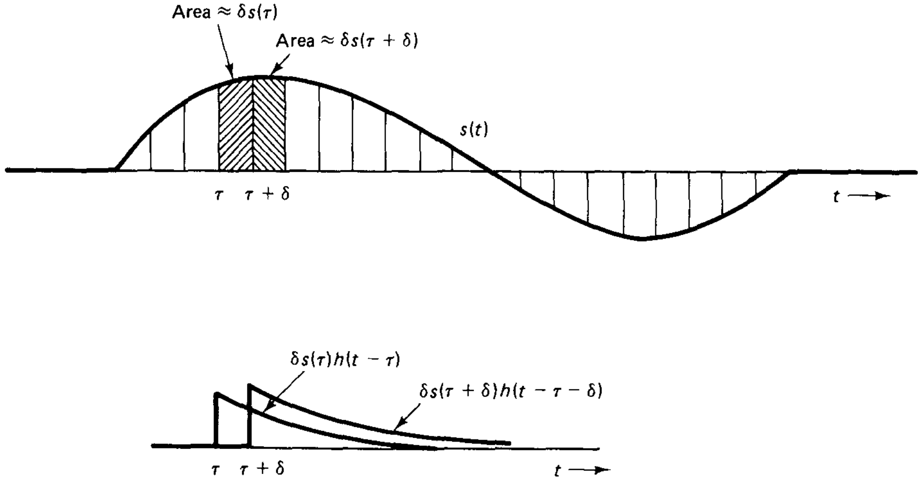

Area ≈ $\delta s(\tau)$
Area ≈ $\delta s(\tau + \delta)$
$s(t)$
τ
$\tau + \delta$
t →
$\delta s(\tau)h(t - \tau)$
$\delta s(\tau + \delta)h(t - \tau - \delta)$
τ
$\tau + \delta$
t →

Figure 2.5 Graphical interpretation of the convolution equation. Input $s(t)$ is viewed as the superposition of narrow pulses of width δ. Each such pulse yields an output $\delta s(\tau)h(t - \tau)$. The overall output is the sum of these pulse responses.

time t. Adding the responses at time t from each of these input pulses and going to the limit $\delta \to 0$ shows that

$$r(t) = \int_{-\infty}^{+\infty} s(\tau)h(t - \tau)d\tau \tag{2.1}$$

This formula is called the *convolution integral*, and $r(t)$ is referred to as the convolution of $s(t)$ and $h(t)$. Note that this formula asserts that the filtering aspects of a channel are completely characterized by the impulse response $h(t)$; given $h(t)$, the output $r(t)$ for any input $s(t)$ can be determined. For the example in Fig. 2.3, it has been assumed that $h(t)$ is 0 for $t < 0$ and $\alpha e^{-\alpha t}$ for $t \geq 0$, where $\alpha = 2/T$. Given this, it is easy to calculate the responses in the figure.

From Eq. (2.1), note that the output at a given time t depends significantly on the input $s(\tau)$ over the interval where $h(t - \tau)$ is significantly nonzero. As a result, if this interval is much larger than the signaling interval T between successive input pulses, significant amounts of intersymbol interference will occur.

Physically, a channel cannot respond to an input before the input occurs, and therefore $h(t)$ should be 0 for $t < 0$; thus, the upper limit of integration in Eq. (2.1) could be taken as t. It is often useful, however, to employ a different time reference at the receiver than at the transmitter, thus eliminating the effect of propagation delay. In this case, $h(t)$ could be nonzero for $t < 0$, which is the reason for the infinite upper limit of integration.

2.2.2 Frequency Response

To gain more insight into the effects of filtering, it is necessary to look at the frequency domain as well as the time domain; that is, one wants to find the effect of filtering on sinusoids of different frequencies. It is convenient analytically to take a broader viewpoint here and allow the channel input $s(t)$ to be a complex function of t; that is, $s(t) = \text{Re}[s(t)] + j\ \text{Im}[s(t)]$, where $j = \sqrt{-1}$. The actual input to a channel is always real, of course. However, if $s(t)$ is allowed to be complex in Eq. (2.1), $r(t)$ will also be complex, but the output corresponding to $\text{Re}[s(t)]$ is simply $\text{Re}[r(t)]$ [assuming that $h(t)$ is real], and the output corresponding to $\text{Im}[s(t)]$ is $\text{Im}[r(t)]$. For a given frequency f, let $s(\tau)$ in Eq. (2.1) be the complex sinusoid $e^{j2\pi f\tau} = \cos(2\pi f\tau) + j\sin(2\pi f\tau)$. Integrating Eq. (2.1) (see Problem 2.3) yields

$$r(t) = H(f)e^{j2\pi ft} \tag{2.2}$$

where

$$H(f) = \int_{-\infty}^{\infty} h(\tau)e^{-j2\pi f\tau}d\tau \tag{2.3}$$

Thus, the response to a complex sinusoid of frequency f is a complex sinusoid of the same frequency, scaled by the factor $H(f)$. $H(f)$ is a complex function of the frequency f and is called the *frequency response* of the channel. It is defined for both positive and negative f. Let $|H(f)|$ be the magnitude and $\angle H(f)$ the phase of $H(f)$ [*i.e.*, $H(f) = |H(f)|e^{j\angle H(f)}$]. The response $r_1(t)$ to the real sinusoid $\cos(2\pi ft)$ is given by the real part of Eq. (2.2), or

$$r_1(t) = |H(f)|\cos[2\pi ft + \angle H(f)] \tag{2.4}$$

Thus, a real sinusoidal input at frequency f gives rise to a real sinusoidal output at the same freqency; $|H(f)|$ gives the amplitude and $\angle H(f)$ the phase of that output relative to the phase of the input. As an example of the frequency response, if $h(t) = \alpha e^{-\alpha t}$ for $t \geq 0$, then integrating Eq. (2.3) yields

$$H(f) = \frac{\alpha}{\alpha + j2\pi f} \tag{2.5}$$

Equation (2.3) maps an arbitrary time function $h(t)$ into a frequency function $H(f)$; mathematically, $H(f)$ is the *Fourier transform* of $h(t)$. It can be shown that the time function $h(t)$ can be recovered from $H(f)$ by the *inverse Fourier transform*

$$h(t) = \int_{-\infty}^{\infty} H(f)e^{j2\pi ft}df \tag{2.6}$$

Equation (2.6) has an interesting interpretation; it says that an (essentially) arbitrary function of time $h(t)$ can be represented as a superposition of an infinite set of infinitesimal complex sinusoids, where the amount of each sinusoid per unit frequency is $H(f)$, as given by Eq. (2.3). Thus, the channel input $s(t)$ (at least over any finite interval) can also be represented as a frequency function by

$$S(f) = \int_{-\infty}^{\infty} s(t)e^{-j2\pi ft}dt \tag{2.7}$$

$$s(t) = \int_{-\infty}^{\infty} S(f)e^{j2\pi ft}df \tag{2.8}$$

The channel output $r(t)$ and its frequency function $R(f)$ are related in the same way. Finally, since Eq. (2.2) expresses the output for a unit complex sinusoid at frequency f, and since $s(t)$ is a superposition of complex sinusoids as given by Eq. (2.8), it follows from the linearity of the channel that the response to $s(t)$ is

$$r(t) = \int_{-\infty}^{\infty} H(f)S(f)e^{j2\pi ft}df \tag{2.9}$$

Since $r(t)$ is also the inverse Fourier transform of $R(f)$, the input–output relation in terms of frequency is simply

$$R(f) = H(f)S(f) \tag{2.10}$$

Thus, the convolution of $h(t)$ and $s(t)$ in the time domain corresponds to the multiplication of $H(f)$ and $S(f)$ in the frequency domain. One sees from Eq. (2.10) why the frequency domain provides insight into filtering.

If $H(f) = 1$ over some range of frequencies and $S(f)$ is nonzero only over those frequencies, then $R(f) = S(f)$, so that $r(t) = s(t)$. One is not usually so fortunate as to have $H(f) = 1$ over a desired range of frequencies, but one could filter the received signal by an additional filter of frequency response $H^{-1}(f)$ over the desired range; this additional filtering would yield a final filtered output satisfying $r(t) = s(t)$. Such filters can be closely approximated in practice subject to additional delay [*i.e.*, satisfying $r(t) = s(t - \tau)$, for some delay τ]. Such filters can even be made to adapt automatically to the actual channel response $H(f)$; filters of this type are usually implemented digitally, operating on a sampled version of the channel output, and are called *adaptive equalizers*. These filters can and often are used to overcome the effects of intersymbol interference.

The question now arises as to what frequency range should be used by the signal $s(t)$; it appears at this point that any frequency range could be used as long as $H(f)$ is nonzero. The difficulty with this argument is that the additive noise that is always present on a channel has been ignored. Noise is added to the signal at various points along the propagation path, including at the receiver. Thus, the noise is not filtered in the channel in the same way as the signal. If $|H(f)|$ is very small in some interval of frequencies, the signal is greatly attenuated at those frequencies, but typically the noise is not attenuated. Thus, if the received signal is filtered at the receiver by $H^{-1}(f)$, the signal is restored to its proper level, but the noise is greatly amplified.

The conclusion from this argument (assuming that the noise is uniformly distributed over the frequency band) is that the digital data should be mapped into $s(t)$ in such a way that $|S(f)|$ is large over those frequencies where $|H(f)|$ is large and $|S(f)|$ is small (ideally zero) elsewhere. The cutoff point between large and small depends on the noise level and signal level and is not of great relevance here, particularly since the argument above is qualitative and oversimplified. Coming back to the example where $h(t) = \alpha e^{-\alpha t}$ for $t \geq 0$, the frequency response was given in Eq. (2.5) as $H(f) = \alpha/(\alpha + j2\pi f)$. It is seen that $|H(f)|$ is approximately 1 for small f and decreases as $1/f$ for large f. We shall not attempt to calculate $S(f)$ for the NRZ code of Fig. 2.3, partly because $s(t)$ and $S(f)$ depend on the particular data sequence being encoded, but the effect of changing the signaling interval T can easily be found. If T is decreased, then $s(t)$ is compressed in time, and this correspondingly expands $S(f)$ in frequency (see Problem 2.5). The equalization at the receiver would then be required either to equalize $H(f)$ over a broader band of frequencies, thus increasing the noise, or to allow more intersymbol interference.

2.2.3 The Sampling Theorem

A more precise way of looking at the question of signaling rates comes from the sampling theorem. This theorem states that if a waveform $s(t)$ is low-pass limited to frequencies at most W [*i.e.*, $S(f) = 0$, for $|f| > W$], then [assuming that $S(f)$ does not contain an impulse at $f = W$], $s(t)$ is completely determined by its values each $1/(2W)$ seconds; in particular,

$$s(t) = \sum_{i=-\infty}^{\infty} s\left(\frac{i}{2W}\right) \frac{\sin[2\pi W(t - i/(2W))]}{2\pi W[t - i/(2W)]} \tag{2.11}$$

Also, for any choice of sample values at intervals 1/(2W), there is a low-pass waveform, given by Eq. (2.11), with those sample values. Figure 2.6 illustrates this result.

The impact of this result, for our purposes, is that incoming digital data can be mapped into sample values at a spacing of $1/(2W)$ and used to create a waveform of the given sample values that is limited to $|f| \leq W$. If this waveform is then passed through an ideal low-pass filter with $H(f) = 1$ for $|f| \leq W$ and $H(f) = 0$ elsewhere, the received waveform will be identical to the transmitted waveform (in the absence of noise); thus, its samples can be used to recreate the original digital data.

The NRZ code can be viewed as mapping incoming bits into sample values of $s(t)$, but the samples are sent as rectangular pulses rather than the ideal $(\sin x)/x$ pulse shape of Eq. (2.11). In a sense, the pulse shape used at the transmitter is not critically important

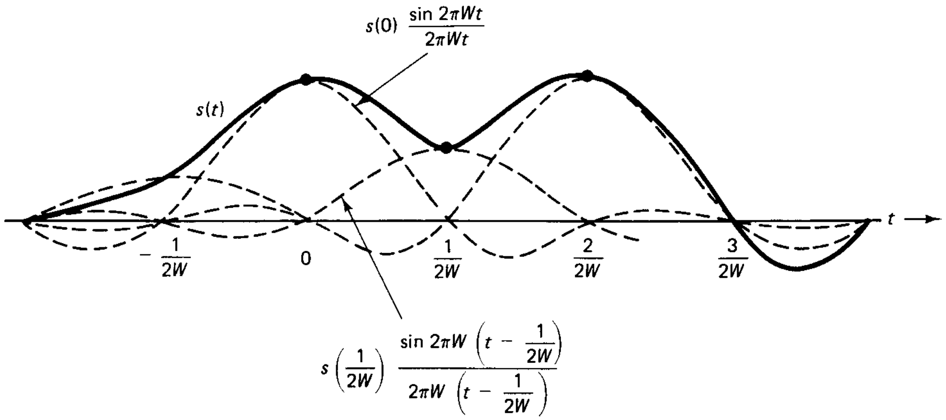

$s(0)\,\frac{\sin 2\pi Wt}{2\pi Wt}$
$s(t)$
t
$-\frac{1}{2W}$
0
$\frac{1}{2W}$
$\frac{2}{2W}$
$\frac{3}{2W}$
$s\left(\frac{1}{2W}\right)\frac{\sin 2\pi W\left(t-\frac{1}{2W}\right)}{2\pi W\left(t-\frac{1}{2W}\right)}$

Figure 2.6 Sampling theorem, showing a function $s(t)$ that is low-pass limited to frequencies at most W. The function is represented as a superposition of $(\sin x)/x$ functions. For each sample, there is one such function, centered at the sample and with a scale factor equal to the sample value.

since the pulse shape can be viewed as arising from filtering sample impulses. If the combination of the pulse-shaping filter at the input with the channel filter and equalizer filter has the $(\sin x)/x$ shape of Eq. (2.11) (or, equivalently, a combined frequency response equal to the ideal low-pass filter response), the output samples will re-create the input samples.

The ideal low-pass filter and the $(\sin x)/x$ pulse shape suggested by the sampling theorem are nice theoretically but not very practical. What is required in practice is a more realistic filter for which the sample values at the output replicate those at the input (*i.e.*, no intersymbol interference) while the high-frequency noise is filtered out.

An elegant solution to this problem was given in a classic paper by Nyquist [Nyq28]. He showed that intersymbol interference is avoided by using a filter $H'(f)$ with odd symmetry at the band edge; that is, $H'(f+W) = 1 - H'(f-W)$ for $|f| \leq W$, and $H'(f) = 0$ for $|f| > 2W$ (see Fig. 2.7). The filter $H'(f)$ here is the composite of the pulse-shaping filter at the transmitter, the channel filter, and the equalizer filter. In practice, such a filter usually cuts off rather sharply around $f = W$ to avoid high-frequency noise, but ideal filtering is not required, thereby allowing considerable design flexibility.

It is important to recognize that the sampling theorem specifies the number of samples per second that can be utilized on a low-pass channel, but it does not specify how many bits can be mapped into one sample. For example, two bits per sample could be achieved by the mapping $11 \rightarrow 3, 10 \rightarrow 1, 00 \rightarrow -1$, and $01 \rightarrow -3$. As discussed later, it is the noise that limits the number of bits per sample.

2.2.4 Bandpass Channels

So far, we have considered only low-pass channels for which $|H(f)|$ is large only for a frequency band around $f = 0$. Most physical channels do not fall into this category, and instead have the property that $|(H(f)|$ is significantly nonzero only within some frequency band $f_1 \leq |f| \leq f_2$, where $f_1 > 0$. These channels are called bandpass channels and many of them have the property that $H(0) = 0$. A channel or waveform with $H(0) = 0$ is said to have no dc component. From Eq. (2.3), it can be seen that this implies that $\int_{-\infty}^{\infty} h(t)dt = 0$. The impulse response for these channels fluctuates

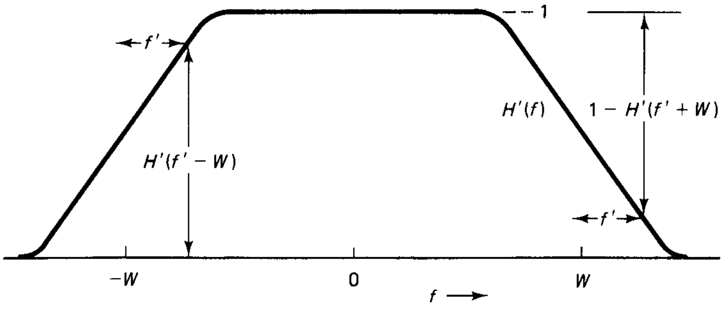


Figure 2.7 Frequency response $H'(f)$ that satisfies the Nyquist criterion for no intersymbol interference. Note that the response has odd symmetry around the point $f = W$.

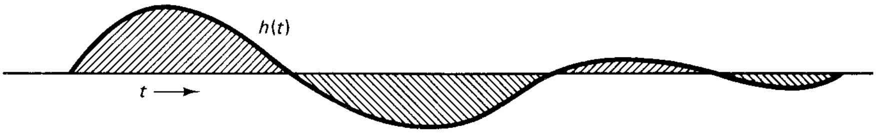


Figure 2.8 Impulse response $h(t)$ for which $H(f) = 0$ for $f = 0$. Note that the area over which $h(t)$ is positive is equal to that over which it is negative.

around 0, as illustrated in Fig. 2.8; this phenomenon is often called ringing. This type of impulse response suggests that the NRZ code is not very promising for a bandpass channel.

To avoid the foregoing problems, most modems for bandpass channels either directly encode digital data into signals with no dc component or else use modulation techniques. The best known direct encoding of this type is Manchester coding (see Fig. 2.9). As seen from the figure, the signals used to represent 0 and 1 each have no dc component and also have a transition (either 1 to -1 or -1 to 1) in the middle of each signaling interval. These transitions simplify the problem of timing recovery at the receiver (note that with the NRZ code, timing recovery could be difficult for a long run of 0's or 1's even in the absence of filtering and noise). The price of eliminating dc components in this way is the use of a much broader band of frequencies than required. Manchester coding is widely used in practice, particularly in the Ethernet system and the corresponding IEEE 802.3 standard described in Section 4.5. There are many other direct encodings, all of a somewhat ad hoc nature, with varying frequency characteristics and no dc component.

2.2.5 Modulation

One of the simplest modulation techniques is amplitude modulation (AM). Here a signal waveform $s(t)$ (called a baseband signal) is generated from the digital data as before, say by the NRZ code. This is then multiplied by a sinusoidal carrier, say $\cos(2\pi f_0 t)$, to generate a modulated signal $s(t)\cos(2\pi f_0 t)$. It is shown in Problem 2.6 that the frequency representation of this modulated signal is $[S(f - f_0) + S(f + f_0)]/2$ (see Fig. 2.10). At the receiver, the modulated signal is again multiplied by $\cos(2\pi f_0 t)$, yielding a received signal

$$\begin{aligned} r(t) &= s(t)\cos^2(2\pi f_0 t) \\ &= \frac{s(t)}{2} + \frac{s(t)\cos(4\pi f_0 t)}{2} \end{aligned} \tag{2.12}$$

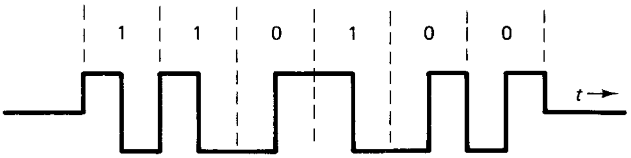


Figure 2.9 Manchester coding. A binary 1 is mapped into a positive pulse followed by a negative pulse, and a binary 0 is mapped into a negative pulse followed by a positive pulse. Note the transition in the middle of each signal interval.

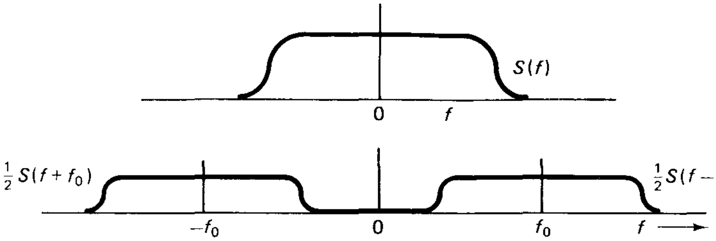


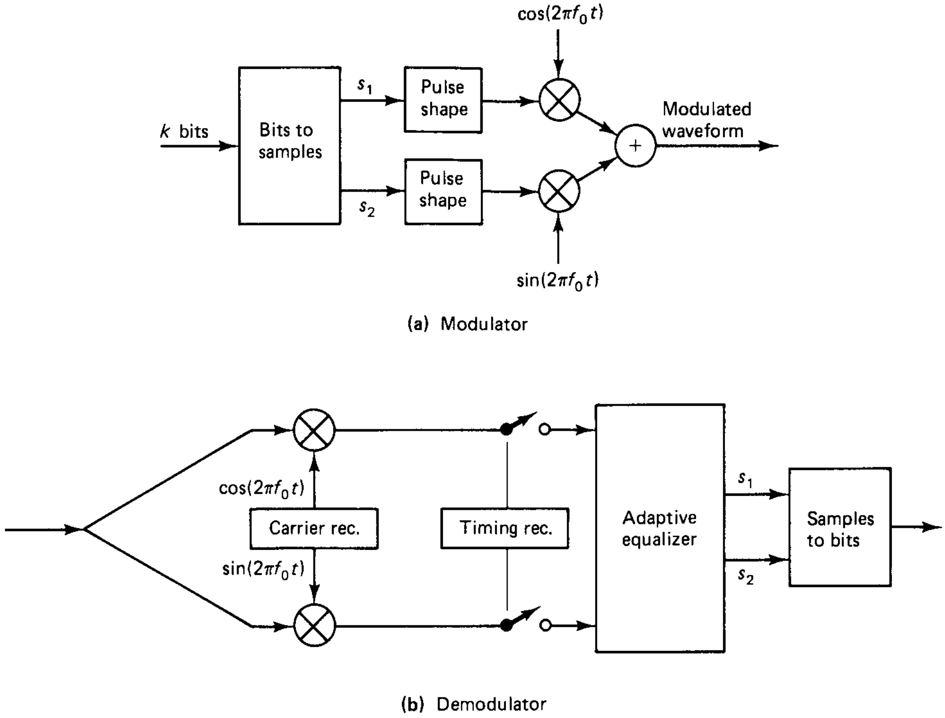
Figure 2.10 Amplitude modulation. The frequency characteristic of the waveform $s(t)$ is shifted up and down by f_0 in frequency.

The high-frequency component, at $2f_0$, is then filtered out at the receiver, leaving the demodulated signal $s(t)/2$, which is then converted back to digital data. In practice, the incoming bits are mapped into shorter pulses than those of NRZ and then filtered somewhat to remove high-frequency components. It is interesting to note that Manchester coding can be viewed as AM in which the NRZ code is multiplied by a square wave. Although a square wave is not a sinusoid, it can be represented as a sinusoid plus odd harmonics; the harmonics serve no useful function and are often partly filtered out by the channel.

AM is rather sensitive to the receiver knowing the correct phase of the carrier. For example, if the modulated signal $s(t)\cos(2\pi f_0 t)$ were multiplied by $\sin(2\pi f_0 t)$, the low-frequency demodulated waveform would disappear. This suggests, however, the possibility of transmitting twice as many bits by a technique known as quadrature amplitude modulation (QAM). Here, the incoming bits are mapped into two baseband signals, $s_1(t)$ and $s_2(t)$. Then $s_1(t)$ is multiplied by $\cos(2\pi f_0 t)$ and $s_2(t)$ by $\sin(2\pi f_0 t)$; the sum of these products forms the transmitted QAM signal (see Fig. 2.11). The received waveform is separately multiplied by $\cos(2\pi f_0 t)$ and $\sin(2\pi f_0 t)$ The first multiplication (after filtering out the high-frequency component) yields $s_1(t)/2$, and the second yields $s_2(t)/2$.

QAM is widely used in high-speed modems for voice-grade telephone channels. These channels have a useful bandwidth from about 500 to 2900 Hz (hertz, *i.e.*, cycles per second), and the carrier frequency is typically about 1700 Hz. The signals $s_1(t)$ and $s_2(t)$ are usually low-pass limited to 1200 Hz, thus allowing 2400 sample inputs per second for each baseband waveform. Adaptive equalization is usually used at the receiver (either before or after demodulation) to compensate for variations in the channel frequency response.

The next question is how to map the incoming bits into the low-pass waveforms $s_1(t)$ and $s_2(t)$. The pulse shape used in the mapping is not of central importance; it will be filtered anyway, and the equalizer will compensate for its shape automatically in attempting to eliminate intersymbol interference. Thus, what is important is mapping the input bits into sample amplitudes of $s_1(t)$ and $s_2(t)$. One could map one bit into a sample of s_1, mapping 1 into +1 and 0 into -1, and similarly map a second bit into a sample of s_2. This is shown in Fig. 2.12(a), viewed as a mapping from two bits into two amplitudes.



(a) Modulator

(b) Demodulator

Figure 2.11 Quadrature amplitude modulation. (a) Each sample period, k bits enter the modulator, are converted into quadrature amplitudes, are then modulated by sine and cosine functions, respectively, and are then added. (b) The reverse operations take place at the demodulator.

Similarly, for any given integer k, one can map k bits into two amplitudes. Each of the 2^k combinations of k bits map into a different amplitude pair. This set of 2^k amplitude pairs in a mapping is called a signal constellation (see Fig. 2.12 for a number of examples). The first two constellations in Fig. 2.12 are also referred to as phase-shift keying (PSK), since they can be viewed as simply changing the phase of the carrier rather than changing its amplitude. For voice-grade modems, where $W = 2400$ Hz, $k = 2$ [as in Fig. 2.12(a)] yields 4800 bits per second (bps), whereas $k = 4$ [as in Fig. 2.12(d)] yields 9600 bps.

To reduce the likelihood of noise causing one constellation point to be mistaken for another, it is desirable to place the signal points as far from each other as possible subject to a constraint on signal power (*i.e.*, on the mean-squared distance from origin to constellation point, taken over all constellation points). It can be shown that the constellation in Fig. 2.12(c) is preferable in this sense to that in Fig. 2.12(b) for $k = 3$.

Modems for voice-grade circuits are also available at rates of 14,400 and 19,200 bps. These use correspondingly larger signal constellations and typically also use error-correction coding. It is reasonable to ask at this point what kind of limit is imposed on data rate by channels subject to bandwidth constraints and noise. Shannon [Sha48] has

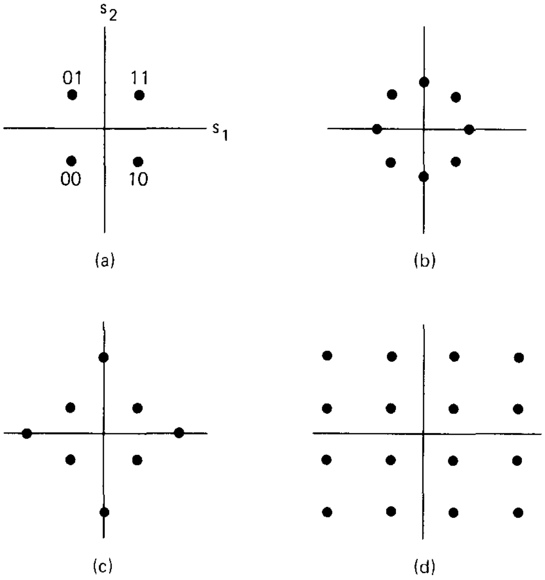


Figure 2.12 Signal constellations for QAM. Part (a) maps two binary digits into a quadrature amplitude sample. Parts (b) and (c) each map three binary digits, and part (d) maps four binary digits. Parts (a) and (b) also can be regarded as phase-shift keying.

shown that the capacity (*i.e.*, the maximum reliable data rate in bps) of such a channel is given by

$$C = W \log_2 \left(1 + \frac{S}{N_0 W}\right) \tag{2.13}$$

where W is the available bandwidth of the channel (essentially 2400 Hz for voice-grade telephone), S is the allowable signal power as seen by the receiver, and $N_0 W$ is the noise power within the bandwidth W seen by the receiver (*i.e.*, N_0 is the noise power per unit bandwidth, assumed uniformly distributed over W).

This is a deep and subtle result, and the machinery has not been developed here even to state it precisely, let alone to justify its validity. The description of QAM, however, at least provides some intuitive justification for the form of the result. The available sampling rate $1/T$ is equal to W by the sampling theorem [recall that W is twice the low-pass bandwidth of $s_1(t)$ and $s_2(t)$]. Since k bits are transmitted per sample, Wk bps can be sent. Next note that some minimum area must be provided around each point in the signal constellation to allow for noise, and this area is proportional to $N_0 W$. Also, the total area occupied by the entire constellation is proportional to the signal power S. Thus, 2^k, the number of points in the constellation, should be proportional to $S/(N_0 W)$, making k proportional to $\log_2[S/(N_0 W)]$.

Communication engineers usually express the signal-to-noise ratio [*i.e.*, $S/(N_0 W)$] in terms of decibels (dB), where the number of dB is defined as $10 \log_{10}[S/(N_0 W)]$. Thus, k, the number of bits per quadrature sample, is proportional to the signal-to-noise ratio in dB. The additional 1 in Eq. (2.13) is harder to explain; note, however, that it is required to keep the capacity positive when the signal-to-noise ratio is very small. The intuitive argument above would make it appear that the error probability depends on the spacing of points in the constellation and thus on k. The beauty of Shannon's theorem, on the other hand, is that it asserts that with the use of error-correction coding, any rate less than C can be achieved with arbitrarily small error probability. The capacity of voice-grade telephone channels is generally estimated to be about 25,000 bps, indicating that the data rates of voice-grade modems are remarkably close to the theoretical limit.

High-speed modems generally maintain signal timing, carrier phase, and adaptive equalization by slowly updating these quantities from the received signal. Thus, it is essential for these modems to operate synchronously; each T seconds, k bits must enter the modem, and the DLC unit must provide idle fill when there are no data to send. For low-speed modems, on the other hand, it is permissible for the physical channel to become idle in between frames or even between individual characters. What is regarded as high speed here depends on the bandwidth of the channel. For modems on 3 kHz voice-grade channels, for example, data rates of 2400 bps or more are usually synchronous, whereas lower rates might be intermittent synchronous or character asynchronous. For a coaxial cable, on the other hand, the bandwidth is so large that intermittent synchronous data rates of 10 megabits per second (Mbps) or more can be used. As mentioned before, this is essential for multiaccess systems such as Ethernet.

2.2.6 Frequency- and Time-Division Multiplexing

In previous subsections, bandwidth constraints imposed by filtering on the physical channel and by noise have been discussed. Often, however, a physical channel is shared by multiple signals each constrained to a different portion of the available bandwidth; this is called frequency-division multiplexing (FDM). The most common examples of this are AM, FM, and TV broadcasting, in which each station uses a different frequency band. As another common example, voice-grade channels are often frequency multiplexed together in the telephone network. In these examples, each multiplexed signal is constrained to its own allocated frequency band to avoid interference (sometimes called crosstalk) with the other signals. The modulation techniques of the preceding subsection are equally applicable whether a bandwidth constraint is imposed by FDM or by channel filtering.

FDM can be viewed as a technique for splitting a big channel into many little channels. Suppose that a physical channel has a usable bandwidth of W hertz and we wish to split it into m equal FDM subchannels. Then, each subchannel has W/m hertz available, and thus W/m available quadrature samples per second for sending data (in practice, the bandwidth per subchannel must be somewhat smaller to provide guard bands between the subchannels, but that is ignored here for simplicity). In terms of Eq. (2.13), each subchannel is allotted $1/m$ of the overall available signal power and is subject to $1/m$ of the overall noise, so each subchannel has $1/m$ of the total channel capacity.

Time-division multiplexing (TDM) is an alternative technique for splitting a big channel into many little channels. Here, one modem would be used for the overall bandwidth W. Given m equal rate streams of binary data to be transmitted, the m bit streams would be multiplexed together into one bit stream. Typically, this is done by sending the data in successive frames. Each frame contains m slots, one for each bit stream to be multiplexed; a slot contains a fixed number of bits, sometimes 1, sometimes 8, and sometimes more. Typically, each frame also contains extra bits to help the receiver maintain frame synchronization. For example, T1 carrier, which is in widespread telephone network use in the United States and Japan, multiplexes 24 data streams into slots of eight bits each with one extra bit per frame for synchronization. The overall data rate is 1.544 Mbps, with 64,000 bps for each of the multiplexed streams. In Europe, there is a similar system with 32 data streams and an overall data rate of 2.048 Mbps.

One way to look at FDM and TDM is that in each case one selects W quadrature samples per second (or more generally, $2W$ individual samples) for transmission; in one case, the samples are distributed in frequency and in the other they are distributed in time.

2.2.7 Other Channel Impairments

The effects of filtering and noise have been discussed in preceding subsections. There are a number of other possible impairments on physical channels. Sometimes there are multiple stages of modulation and demodulation internal to the physical channel. These can cause small amounts of phase jitter and carrier frequency offset in the received waveform. Amplification of the signals, within the physical channel and in the modems, can cause nonnegligible nonlinear distortion. Impulses of noise can occur due to lightning and switching effects. Repair personnel can short out or disconnect the channel for periods of time. Finally, there can be crosstalk from other frequency bands and from nearby wires.

To a certain extent, these effects can all be considered as extra noise sources. However, these noise sources have different statistical properties than the noise assumed by Shannon's theorem. (Technically, that noise is known as additive white Gaussian noise.) The principal difference is that the errors caused by these extra noise sources tend to occur in bursts of arbitrarily long length. As a result, one cannot, in practice, achieve the arbitrarily low error probabilities promised by Shannon's theorem. One must also use error detection and retransmission at the DLC layer; this is treated in Sections 2.3 and 2.4.

2.2.8 Digital Channels

In many cases, physical channels are designed to carry digital data directly, and the DLC unit can interface almost directly to the digital channel rather than to a modem mapping digital data into analog signals. To a certain extent, this is simply the question of whether the channel supplier or the channel user supplies the modem. This is an oversimplified view, however. A channel designed to carry digital data directly (such as the T1 carrier mentioned earlier) is often capable of achieving higher data rates at lower error probabilities than one carrying analog signals.

A major reason for this improved performance is the type of repeater used for digital channels. Repeaters are basically amplifiers inserted at various points in the propagation path to overcome attenuation. For a channel designed for analog signals, both the signal and the noise must be amplified at each repeater. Thus, the noise at the final receiver is an accumulation of noise over each stage of the path. For a digital channel, on the other hand, the digital signal can be recovered at each repeater. This means that the noise does not accumulate from stage to stage, and an error occurs only if the noise in a single stage is sufficient to cause an error. In effect, the noise is largely suppressed at each stage. Because of this noise suppression, and also because of the low cost of digital processing, it is increasingly common to use digital channels (such as the T1 carrier system) for the transmission of analog signals such as voice. Analog signals are sampled (typically at 8000 samples per second for voice) and then quantized (typically at eight bits per sample) to provide a digital input for a digital channel.

The telephone network can be separated into two parts—the local loops, which go from subscribers to local offices, and the internal network connecting local offices, central offices, and toll switches. Over the years, the internal network has become mostly digital, using TDM systems such as the T1 carrier just described and a higher-speed carrier system called T3, at 44.736 Mbps, that will multiplex 28 T1 signals.

There are two consequences to this increasing digitization. First, the telephone companies are anxious to digitize the local loops, thus allowing 64 kilobit per second (kbps) digital service over a single voice circuit. An added benefit of digitizing the local loops is that more than a single voice circuit could be provided to each subscriber. Such an extension of the voice network in which both voice and data are simultaneously available in an integrated way is called an *integrated services digital network* (ISDN). The other consequence is that data network designers can now lease T1 lines (called DS1 service) and T3 lines (called DS3 service) at modest cost and thus construct networks with much higher link capacities than are available on older networks. These higher-speed links are making it possible for wide area networks to send data at speeds comparable to local area nets.

With the increasing use of optical fiber, still higher link speeds are being standardized. *SONET* (Synchronous Optical Network) is the name for a standard family of interfaces for high-speed optical links. These start at 51.84 Mbps (called STS-1), and have various higher speeds of the form n times 51.84 Mbps (called STS-n) for $n = 1, 3, 9, 12, 18, 24, 36, 48$. Like the T1 and T3 line speeds, each of these speeds has a 125 μs frame structure which can be broken down into a very large number of 64 kbps voice circuits, each with one byte per frame. What is ingenious and interesting about these standards is the way that the links carrying these frames can be joined together, in the presence of slight clock drifts, without losing or gaining bits for any of the voice circuits. What is relevant for our purposes, however, is that 1.5 and 45 Mbps link speeds are now economically available, and much higher speeds will be economically available in the future. The other relevant fact is that optical fibers tend to be almost noise-free, with error rates of less than 10^{-10}.

ISDN As outlined briefly above, an integrated services digital network (ISDN) is a telephone network in which both the internal network and local loops are digital. The "integrated service" part of the name refers to the inherent capability of such a network to carry both voice and data in an integrated fashion. A telephone network with analog voice lines for local loops also allows both data and voice to be carried, but not easily at the same time and certainly not with much speed or convenience. There has been considerable effort to standardize the service provided by ISDN. Two standard types of service are available. The first, called *basic service*, provides two 64 kbps channels plus one 16 kbps channel to the user. Each 64 kbps channel can be used as an ordinary voice channel or as a point-to-point data channel. The 16 kbps channel is connected to the internal signaling network of the telephone system (*i.e.*, the network used internally for setting up and disconnecting calls, managing the network, etc.). Thus, the 16 kbps channel can be used to set up and control the two 64 kbps channels, but it could also be used for various low-data-rate services such as alarm and security systems.

In ISDN jargon, the 64 kbps channels are called B channels and the 16 kbps channel is called a D channel; the overall basic service is thus referred to as $2B + D$. With these facilities, a subscriber could conduct two telephone conversations simultaneously, or have one data session (at 64 kbps) plus one voice conversation. The latter capability appears to be rather useful to someone who works at home with a terminal but does not want to be cut off from incoming telephone calls. Most terminal users, of course, would be quite happy, after getting used to 300 bps terminals, to have the data capabilities of the 16 kbps D channel, and it is likely that subscribers will be offered a cheaper service consisting of just one B and one D channel.

The 64 kbps channels can be used in two ways: to set up a direct connection to some destination (thus using ISDN as a circuit switched network), and as an access line into a node of a packet switching network. On the one hand, a 64 kbps access into a packet switched network appears almost extravagant to one used to conventional networks. On the other hand, if one is trying to transfer a high-resolution graphics image of 10^9 bits, one must wait for over 4 hours using a 64 kbps access line. This illustrates a rather peculiar phenomenon. There are a very large number of applications of data networks that can be accomplished comfortably at very low data rates, but there are others (usually involving images) that require very high data rates.

Optical fiber will gradually be introduced into the local loops of the telephone network, and this will allow ISDN to operate at much higher data rates than those described above. Broadband ISDN (BISDN) is the name given to such high-speed ISDN networks. There has already been considerable standardization on broadband ISDN, including a standard user access rate of 155 Mbps (*i.e.*, the SONET STS-3 rate). These high data rates will allow for high-resolution TV as well as for fast image transmission, high-speed interconnection of supercomputers, and video conferencing. There are many interesting questions about how to build data networks handling such high data rates; one such strategy, called asynchronous transfer mode (ATM), is discussed briefly in Section 2.10.

The $2B + D$ basic service above is appropriate (or even lavish, depending on one's point of view) for the home or a very small office, but is inappropriate for larger offices used to using a PBX with a number of outgoing telephone lines. What ISDN (as proposed) offers here is something called *primary service* as opposed to basic service. This consists of 24 channels at 64 kbps each (in the United States and Japan) or 31 channels at 64 kbps each (in Europe). One of these channels is designated as the D channel and the others as B channels. The D channel is the one used for signaling and call setup and has a higher data rate here than in the basic service to handle the higher traffic levels. Subscribers can also obtain higher rate channels than the 64 kbps

B channels above. These higher rate channels are called H channels and come in 384, 1536, and 1920 kbps flavors. Such higher rate channels partially satisfy the need for the high data rate file transfers discussed above.

One of the interesting technical issues with ISDN is how to provide the required data rates over local loops consisting of a twisted wire pair. The approach being taken, for basic service, is to use time-division multiplexing of the B and D channels together with extra bits for synchronization and frequency shaping, thus avoiding a dc component. The most interesting part of this is that data must travel over the local loop in both directions, and that a node trying to receive the attenuated signal from the other end is frustrated by the higher-level signal being generated at the local end. For voice circuits, this problem has been solved in the past by a type of circuit called a hybrid which isolates the signal going out on the line from that coming in. At these high data rates, however, this circuit is not adequate and adaptive echo cancelers are required to get rid of the remaining echo from the local transmitted signal at the receiver.

2.2.9 Propagation Media for Physical Channels

The most common media for physical channels are twisted pair (*i.e.*, two wires twisted around each other so as to partially cancel out the effects of electromagnetic radiation from other sources), coaxial cable, optical fiber, radio, microwave, and satellite. For the first three, the propagated signal power decays exponentially with distance (*i.e.*, the attenuation in dB is linear with distance). Because of the attenuation, repeaters are used every few kilometers or so. The rate of attenuation varies with frequency, and thus as repeaters are spaced more closely, the useful frequency band increases, yielding a trade-off between data rate and the cost of repeaters. Despite this trade-off, it is helpful to have a ballpark estimate of typical data rates for channels using these media.

Twisted pair is widely used in the telephone network between subscribers and local stations and is increasingly used for data. One Mbps is a typical data rate for paths on the order of 1 km or less. Coaxial cable is widely used for local area networks, cable TV, and high-speed point-to-point links. Typical data rates are from 10 to several hundred Mbps. For optical fiber, data rates of 1000 Mbps or more are possible. Optical fiber is growing rapidly in importance, and the major problems lie in the generation, reception, amplification, and switching of such massive amounts of data.

Radio, microwave, and satellite channels use electromagnetic propagation in open space. The attenuation with distance is typically much slower than with wire channels, so repeaters can either be eliminated or spaced much farther apart than for wire lines. Frequencies below 1000 MHz are usually referred to as radio frequencies, and higher frequencies are referred to as microwave.

Radio frequencies are further divided at 30 MHz. Above 30 MHz, the ionosphere is transparent to electromagnetic waves, whereas below 30 MHz, the waves can be reflected by the ionosphere. Thus, above 30 MHz, propagation is on line-of-sight paths. The antennas for such propagation are frequently placed on towers or hills to increase the length of these line-of-sight paths, but the length of a path is still somewhat limited and repeaters are often necessary. This frequency range, from 30 to 1000 MHz, is used for UHF and VHF TV broadcast, for FM broadcast, and many specialized applications. Packet radio networks, discussed in Section 4.6, use this frequency band. Typical data rates in this band are highly variable; the DARPA (U.S. Department of Defense Advanced Research Projects Agency) packet radio network, for example, uses 100,000 and 400,000 bps.

Below 30 MHz, long-distance propagation beyond line-of-sight is possible by reflection from the ionosphere. Ironically, the 3 to 30 MHz band is called the high-frequency (HF) band, the terminology coming from the early days of radio. This band is very noisy, heavily used (*e.g.*, by ham radio), and subject to fading. Fading can be viewed as a channel filtering phenomenon with the frequency response changing relatively rapidly in time; this is caused by time-varying multiple propagation paths from source to destination. Typical data rates in this band are 2400 bps and less.

Microwave links (above 1000 MHz) must use line-of-sight paths. The antennas (usually highly directional dishes) yield typical path lengths of 10 to 200 km. Longer paths than this can be achieved by the use of repeaters. These links can carry 1000 Mbps or so and are usually multiplexed between long-distance telephony, TV program distribution, and data.

Satellite links use microwave frequencies with a satellite as a repeater. They have similar data rates and uses as microwave links. One satellite repeater can receive signals from many ground stations and broadcast back in another frequency band to all those ground stations. The satellite can contain multiple antenna beams, allowing it to act as a switch for multiple microwave links. In Chapter 4, multiaccess techniques for sharing individual frequency bands between different ground stations are studied.

This section has provided a brief introduction to physical channels and their use in data transmission. A link in a subnet might use any of these physical channels, or might share such a channel on a TDM or FDM basis with many other uses. Despite the complexity of this subject (which we have barely touched), these links can be regarded simply as unreliable bit pipes by higher layers.

2.3 ERROR DETECTION

The subject of the next four sections is data link control. This section treats the detection of transmission errors, and the next section treats retransmission requests. Assume initially that the receiving data link control (DLC) module knows where frames begin and end. The problem then is to determine which of those frames contain errors. From the layering viewpoint, the packets entering the DLC are arbitrary bit strings (*i.e.*, the function of the DLC layer is to provide error-free packets to the next layer up, no matter what the packet bit strings are). Thus, at the receiving DLC, any bit string is acceptable as a packet and errors cannot be detected by analysis of the packet itself. Note that a transformation on packets of K bits into some other representation of length K cannot help; there are 2^K possible packets and all possible bit strings of length K must be used to represent all possible packets. The conclusion is that extra bits must be appended to a packet to detect errors.

2.3.1 Single Parity Checks

The simplest example of error detection is to append a single bit, called a *parity check*, to a string of data bits. This parity check bit has the value 1 if the number of 1's in the bit string is odd, and has the value 0 otherwise (see Fig. 2.13). In other words, the parity check bit is the sum, modulo 2, of the bit values in the original bit string (k modulo j, for integer k and positive integer j, is the integer m, $0 \leq m < j$, such that $k - m$ is divisible by j).

In the ASCII character code, characters are mapped into strings of seven bits and then a parity check is appended as an eighth bit. One can also visualize appending a parity check to the end of a packet, but it will soon be apparent that this is not a sufficiently reliable way to detect errors.

Note that the total number of 1's in an encoded string (*i.e.*, the original bit string plus the appended parity check) is always even. If an encoded string is transmitted and a single error occurs in transmission, then, whether a 1 is changed to a 0 or a 0 to a 1, the resulting number of 1's in the string is odd and the error can be detected at the receiver. Note that the receiver cannot tell which bit is in error, nor how many errors occurred; it simply knows that errors occurred because of the odd number of 1's.

It is rather remarkable that for bit strings of any length, a single parity check enables the detection of any single error in the encoded string. Unfortunately, two errors in an encoded string always leave the number of 1's even so that the errors cannot be detected. In general, any odd number of errors are detected and any even number are undetected.

Despite the appealing simplicity of the single parity check, it is inadequate for reliable detection of errors; in many situations, it only detects errors in about half of the encoded strings where errors occur. There are two reasons for this poor behavior. The first is that many modems map several bits into a single sample of the physical channel input (see Section 2.2.5), and an error in the reception of such a sample typically causes several bit errors. The second reason is that many kinds of noise, such as lightning and temporarily broken connections, cause long bursts of errors. For both these reasons, when one or more errors occur in an encoded string, an even number of errors is almost as likely as an odd number and a single parity check is ineffective.

2.3.2 Horizontal and Vertical Parity Checks

Another simple and intuitive approach to error detection is to arrange a string of data bits in a two-dimensional array (see Fig. 2.14) with one parity check for each row and one for each column. The parity check in the lower right corner can be viewed as a parity check on the row parity checks, on the column parity checks, or on the data array. If an even number of errors are confined to a single row, each of them can be detected by the

s_1	s_2	s_3	s_4	s_5	s_6	s_7	c
1	0	1	1	0	0	0	1

Figure 2.13 Single parity check. Final bit c is the modulo 2 sum of s_1 to s_k, where $k = 7$ here.

1	0	0	1	0	1	0	1
0	1	1	1	0	1	0	0
1	1	1	0	0	0	1	0
1	0	0	0	1	1	1	0
0	0	1	1	0	0	1	1
1	0	1	1	1	1	1	0

Horizontal checks

Vertical checks

(a)

1	0	0	1	0	1	0	1
0	1	1	1	0	1	0	0
1	1	①	0	0	⓪	1	0
1	0	0	0	1	1	1	0
0	0	①	1	0	⓪	1	1
1	0	1	1	1	1	1	0

(b)

Figure 2.14 Horizontal and vertical parity checks. Each horizontal parity check checks its own row, and each column parity check checks its own column. Note, in part (b), that if each circled bit is changed, all parity checks are still satisfied.

corresponding column parity checks; similarly, errors in a single column can be detected by the row parity checks. Unfortunately, any pattern of four errors confined to two rows and two columns [*i.e.*, forming a rectangle as indicated in Fig. 2.14(b)] is undetectable.

The most common use of this scheme is where the input is a string of ASCII encoded characters. Each encoded character can be visualized as a row in the array of Fig. 2.14; the row parity check is then simply the last bit of the encoded character. The column parity checks can be trivially computed by software or hardware. The major weakness of this scheme is that it can fail to detect rather short bursts of errors (*e.g.*, two adjacent bits in each of two adjacent rows). Since adjacent errors are quite likely in practice, the likelihood of such failures is undesirably high.

2.3.3 Parity Check Codes

The nicest feature about horizontal and vertical parity checks is that the underlying idea generalizes immediately to arbitrary parity check codes. The underlying idea is to start with a bit string (the array of data bits in Fig. 2.14) and to generate parity checks on various subsets of the bits (the rows and columns in Fig. 2.14). The transformation from the string of data bits to the string of data bits and parity checks is called a *parity check code* or *linear code*. An example of a parity check code (other than the horizontal and vertical case) is given in Fig. 2.15. A parity check code is defined by the particular collection of subsets used to generate parity checks. Note that the word *code* refers to the transformation itself; we refer to an encoded bit string (data plus parity checks) as a *code word*.

Let K be the length of the data string for a given parity check code and let L be the number of parity checks. For the frame structure in Fig. 2.2, one can view the data

s_1	s_2	s_3	c_1	c_2	c_3	c_4	
1	0	0	1	1	1	0	
0	1	0	0	1	1	1	$c_1 = s_1 + s_3$
0	0	1	1	1	0	1	$c_2 = s_1 + s_2 + s_3$
1	1	0	1	0	0	1	$c_3 = s_1 + s_2$
1	0	1	0	0	1	1	$c_4 = s_2 + s_3$
1	1	1	0	1	0	0	
0	0	0	0	0	0	0	
0	1	1	1	0	1	0	

Figure 2.15 Example of a parity check code. Code words are listed on the left, and the rule for generating the parity checks is given on the right.

string as the header and packet, and view the set of parity checks as the trailer. Note that it is important to detect errors in the control bits of the header as well as to detect errors in the packets themselves. Thus, $K + L$ is the frame length, which for the present is regarded as fixed. For a given code, each of the possible 2^K data strings of length K is mapped into a frame (*i.e.*, code word) of length $K + L$. In an error-detection system, the frame is transmitted and the receiving DLC module determines if each of the parity checks is still the modulo 2 sum of the corresponding subset of data bits. If so, the frame is regarded by the receiver as error-free, and if not, the presence of errors is detected. If errors on the link convert one code word into another, the frame is regarded by the receiver as error-free, and undetectable errors are said to have occurred in the frame.

Given any particular code, one would like to be able to predict the probability of undetectable errors in a frame for a particular link. Unfortunately, this is very difficult. First, errors on links tend to be dependent and to occur in bursts; there are no good models for the length or intensity of these bursts, which vary widely among links of the same type. Second, for any reasonable code, the frequency of undetectable errors is very small and is thus both difficult to measure experimentally and dependent on rare, difficult-to-model events. The literature contains many calculations of the probability of undetectable errors, but these calculations are usually based on the assumption of independent errors; the results are usually orders of magnitude away from reality.

As a result of these difficulties, the effectiveness of a code for error detection is usually measured by three parameters: (1) the minimum distance of the code, (2) the burst-detecting capability, and (3) the probability that a completely random string will be accepted as error-free. The minimum distance of a code is defined as the smallest number of errors that can convert one code word into another. As we have seen, the minimum distance of a code using a single parity check is 2, and the minimum distance of a code with horizontal and vertical parity checks is 4.

The length of a burst of errors in a frame is the number of bits from the first error to the last, inclusive. The burst-detecting capability of a code is defined as the largest integer B such that a code can detect all bursts of length B or less. The burst-detecting capability of the single parity check code is 1, whereas the burst-detecting capability of a code with horizontal and vertical parity checks is 1 plus the length of a row (assuming that rows are sent one after the other).

By a completely random string of length $K + L$ is meant that each such string is received with probability 2^{-K-L}. Since there are 2^K code words, the probability of an undetected error is the probability that the random string is one of the code words; this occurs with probability 2^{-L} (the possibility that the received random string happens to be the same as the transmitted frame is ignored). This is usually a good estimate of the probability of undetectable errors given that both the minimum distance and the burst-detecting capability of the code are greatly exceeded by the set of errors on a received frame.

Parity check codes can be used for error correction rather than just for error detection. For example, with horizontal and vertical parity checks, any single error can be corrected simply by finding the row and column with odd parity. It is shown in Problem 2.10 that a code with minimum distance d can be used to correct any combination of fewer than $d/2$ errors. Parity check codes (and convolutional codes, which are closely related but lack the frame structure) are widely used for error correction at the physical layer. The modern approach to error correction is to view it as part of the modulation and demodulation process, with the objective of creating a virtual bit pipe with relatively low error rate.

Error correction is generally not used at the DLC layer, since the performance of an error correction code is heavily dependent on the physical characteristics of the channel. One needs error detection at the DLC layer, however, to detect rare residual errors from long noisy periods.

2.3.4 Cyclic Redundancy Checks

The parity check codes used for error detection in most DLCs today are cyclic redundancy check (CRC) codes. The parity check bits are called the CRC. Again, let L be the length of the CRC (*i.e.*, the number of check bits) and let K be the length of the string of data bits (*i.e.*, the header and packet of a frame). It is convenient to denote the data bits as $s_{K-1}, s_{K-2}, \dots, s_1, s_0$, and to represent the string as a polynomial $s(D)$ with coefficients $s_{K-1}, \dots, s_0$,

$$s(D) = s_{K-1}D^{K-1} + s_{K-2}D^{K-2} + \cdots + s_0 \tag{2.14}$$

The powers of the indeterminate D can be thought of as keeping track of which bit is which; high-order terms are viewed as being transmitted first. The CRC is represented as another polynomial,

$$c(D) = c_{L-1}D^{L-1} + \cdots + c_1 D + c_0 \tag{2.15}$$

The entire frame of transmitted information and CRC can then be represented as $x(D) = s(D)D^L + c(D)$, that is, as

$$x(D) = s_{K-1}D^{L+K-1} + \cdots + s_0 D^L + c_{L-1}D^{L-1} + \cdots + c_0 \tag{2.16}$$

The CRC polynomial $c(D)$ is a function of the information polynomial $s(D)$, defined in terms of a *generator polynomial* $g(D)$; this is a polynomial of degree L with binary coefficients that specifies the particular CRC code to be used.

$$g(D) = D^L + g_{L-1}D^{L-1} + \cdots + g_1 D + 1 \tag{2.17}$$

For a given $g(D)$, the mapping from the information polynomial to the CRC polynomial $c(D)$ is given by

$$c(D) = \text{Remainder}\left[\frac{s(D)D^L}{g(D)}\right] \tag{2.18}$$

The polynomial division above is just ordinary long division of one polynomial by another, except that the coefficients are restricted to be binary and the arithmetic on coefficients is performed modulo 2. Thus, for example, $(1+1)$ modulo $2 = 0$ and $(0-1)$ modulo $2 = 1$. Note that subtraction using modulo 2 arithmetic is the same as addition. As an example of the operation in Eq. (2.18),

$$\begin{array}{r l}
 & D^2 + D \\
D^3 + D^2 + 1 & \overline{\big)\, D^5 + \qquad D^3} \\
 & \underline{D^5 + D^4 + \qquad D^2} \\
 & \quad D^4 + D^3 + D^2 \\
 & \quad \underline{D^4 + D^3 + \qquad D} \\
 & \qquad\qquad D^2 + D = \text{Remainder}
\end{array}$$

Since $g(D)$ is a polynomial of degree at most L, the remainder is of degree at most $L-1$. If the degree of $c(D)$ is less than $L-1$, the corresponding leading coefficients $c_{L-1}, \ldots,$ in Eq. (2.18) are taken as zero.

This long division can be implemented easily in hardware by the feedback shift register circuit shown in Fig. 2.16. By comparing the circuit with the long division above, it can be seen that the successive contents of the shift register cells are just the coefficients of the partial remainders in the long division. In practice, CRCs are usually calculated by VLSI chips, which often perform the other functions of DLC as well.

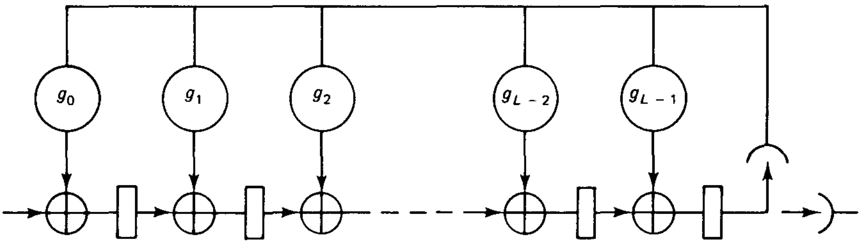


Figure 2.16 Shift register circuit for dividing polynomials and finding the remainder. Each rectangle indicates a storage cell for a single bit and the preceding circles denote modulo 2 adders. The large circles at the top indicate multiplication by the value of g_i. Initially, the register is loaded with the first L bits of $s(D)$ with s_{K-1} at the right. On each clock pulse, a new bit of $s(D)$ comes in at the left and the register reads in the corresponding modulo 2 sum of feedback plus the contents of the previous stage. After K shifts, the switch at the right moves to the horizontal position and the CRC is read out.

Let $z(D)$ be the quotient resulting from dividing $s(D)D^L$ by $g(D)$. Then, $c(D)$ can be represented as

$$s(D)D^L = g(D)z(D) + c(D) \tag{2.19}$$

Subtracting $c(D)$ (modulo 2) from both sides of this equation and recognizing that modulo 2 subtraction and addition are the same, we obtain

$$x(D) = s(D)D^L + c(D) = g(D)z(D) \tag{2.20}$$

Thus, all code words are divisible by $g(D)$, and all polynomials divisible by $g(D)$ are code words. It has not yet been shown that the mapping from $s(D)$ to $x(D)$ corresponds to a parity check code. This is demonstrated in Problem 2.15 but is not necessary for the subsequent development.

Now suppose that $x(D)$ is transmitted and that the received sequence is represented by a polynomial $y(D)$, where $x(D)$ and $y(D)$ differ because of the errors on the communication link. If the error sequence is represented as a polynomial $e(D)$, then $y(D) = x(D) + e(D)$, where, as throughout this section, $+$ means modulo 2 addition; each error in the frame corresponds to a nonzero coefficient in $e(D)$ [*i.e.*, a coefficient in which $y(D)$ and $x(D)$ differ]. At the receiver, Remainder$[y(D)/g(D)]$ can be calculated by essentially the same circuit as that above. Since it has been shown that $x(D)$ is divisible by $g(D)$,

$$\text{Remainder}\left[\frac{y(D)}{g(D)}\right] = \text{Remainder}\left[\frac{e(D)}{g(D)}\right] \tag{2.21}$$

If no errors occur, then $e(D) = 0$ and the remainder above will be 0. The rule followed by the receiver is to decide that the frame is error- free if this remainder is 0 and to decide that there are errors otherwise. When errors actually occur [*i.e.*, $e(D) \neq 0$], the receiver fails to detect the errors only if this remainder is 0; this occurs only if $e(D)$ is itself some code word. In other words, $e(D) \neq 0$ is undetectable if and only if

$$e(D) = g(D)z(D) \tag{2.22}$$

for some nonzero polynomial $z(D)$. We now explore the conditions under which undetected errors can occur.

First, suppose that a single error occurs, say $e_i = 1$, so that $e(D) = D^i$. Since $g(D)$ has at least two nonzero terms (*i.e.*, D^L and 1), $g(D)z(D)$ must also have at least two nonzero terms for any nonzero $z(D)$ (see Problem 2.13). Thus $g(D)z(D)$ cannot equal D^i; since this is true for all i, all single errors are detectable. By the same type of argument, since the highest-order and lowest-order terms in $g(D)$ (*i.e.*, D^L and 1, respectively) differ by L, the highest-order and lowest-order terms in $g(D)z(D)$ differ by at least L for all nonzero $z(D)$. Thus, if $e(D)$ is a code word, the burst length of the errors is at least $L + 1$ (the +1 arises from the definition of burst length as the number of positions from the first error to the last error *inclusive*).

Next, suppose that a double error occurs, say in positions i and j, so that

$$e(D) = D^i + D^j = D^j(D^{i-j} + 1), \quad i > j \tag{2.23}$$

From the argument above, D^j is not divisible by $g(D)$ or by any factor of $g(D)$; thus, $e(D)$ fails to be detected only if $D^{i-j}+1$ is divisible by $g(D)$. For any binary polynomial $g(D)$ of degree L, there is some smallest n for which $D^n + 1$ is divisible by $g(D)$. It is known from the theory of finite fields that this smallest n can be no larger than $2^L - 1$; moreover, for all $L > 0$, there are special L-degree polynomials, called *primitive polynomials*, such that this smallest n is equal to $2^L - 1$. Thus, if $g(D)$ is chosen to be such a primitive polynomial of degree L, and if the frame length is restricted to be at most $2^L - 1$, then $D^{i-j} + 1$ cannot be divisible by $g(D)$; thus, all double errors are detected.

In practice, the generator polynomial $g(D)$ is usually chosen to be the product of a primitive polynomial of degree $L-1$ times the polynomial $D+1$. It is shown in Problem 2.14 that a polynomial $e(D)$ is divisible by $D + 1$ if and only if $e(D)$ contains an even number of nonzero coefficients. This ensures that all odd numbers of errors are detected, and the primitive polynomial ensures that all double errors are detected (as long as the block length is less than 2^{L-1}). Thus, any code of this form has a minimum distance of at least 4, a burst-detecting capability of at least L, and a probability of failing to detect errors in completely random strings of 2^{-L}. There are two standard CRCs with length $L = 16$ (denoted CRC-16 and CRC-CCITT). Each of these CRCs is the product of $D+1$ times a primitive $(L-1)$- degree polynomial, and thus both have the foregoing properties. There is also a standard CRC with $L = 32$. It is a 32-degree primitive polynomial, and has been shown to have a minimum distance of 5 for block lengths less than 3007 and 4 for block lengths less than 12,145 [FKL86]. These polynomials are as follows:

$$g(D) = D^{16} + D^{15} + D^2 + 1 \qquad \text{for CRC-16}$$

$$g(D = D^{16} + D^{12} + D^5 + 1 \qquad \text{for CRC-CCITT}$$

$$g(D) = D^{32} + D^{26} + D^{23} + D^{22} + D^{16} + D^{12} + D^{11} + \\ D^{10} + D^8 + D^7 + D^5 + D^4 + D^2 + D^1 + 1$$

2.4 ARQ: RETRANSMISSION STRATEGIES

The general concept of *a*utomatic *r*epeat re*q*uest (ARQ) is to detect frames with errors at the receiving DLC module and then to request the transmitting DLC module to repeat the information in those erroneous frames. Error detection was discussed in the preceding section, and the problem of requesting retransmissions is treated in this section. There are two quite different aspects of retransmission algorithms or protocols. The first is that of correctness: Does the protocol succeed in releasing each packet, once and only once, without errors, from the receiving DLC? The second is that of efficiency: How much of the bit-transmitting capability of the bit pipe is wasted by unnecessary waiting and by sending unnecessary retransmissions? First, several classes of protocols are developed and shown to be correct (in a sense to be defined more precisely later). Later, the effect that the various parameters in these classes have on efficiency is considered.

Recall from Fig. 2.2 that packets enter the DLC layer from the network layer. The DLC module appends a header and trailer to each packet to form a frame, and the frames are transmitted on the virtual bit pipe (*i.e.*, are sent to the physical layer for transmission). When errors are detected in a frame, a new frame containing the old packet is transmitted. Thus, the first transmitted frame might contain the first packet, the next frame the second packet, the third frame a repetition of the first packet, and so forth. When a packet is repeated, the frame header and trailer might or might not be the same as in the earlier version.

Since framing will not be discussed until the next section, we continue to assume that the receiving DLC knows when frames start and end; thus a CRC (or any other technique) may be used for detecting errors. We also assume, somewhat unrealistically, that *all* frames containing transmission errors are detected. The reason for this is that we want to prove that ARQ works correctly except when errors are undetected. This is the best that can be hoped for, since error detection cannot work with perfect reliability and bounded delay; in particular, any code word can be changed into another code word by some string of transmission errors. This can cause erroneous data to leave the DLC or, perhaps worse, can cause some control bits to be changed. In what follows, we refer to frames without transmission errors as *error-free frames* and those with transmission errors as *error frames*. We are assuming, then, that the receiver can always disinguish error frames from error-free frames.

Finally, we need some assumptions about the bit pipes over which these frames are traveling. The reason for these assumptions will be clearer when framing is studied; in effect, these assumptions will allow us to relax the assumption that the receiving DLC has framing information. Assume first that each transmitted frame is delayed by an arbitrary and variable time before arriving at the receiver, and assume that some frames might be "lost" and never arrive. Those frames that arrive, however, are assumed to arrive in the same order as transmitted, with or without errors. Figure 2.17 illustrates this behavior.

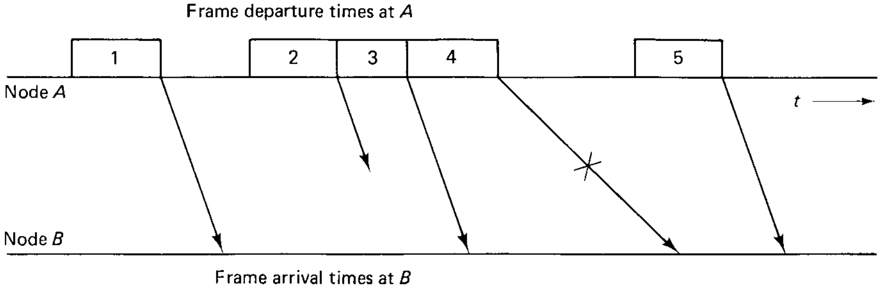


Figure 2.17 Model of frame transmissions: frame 2 is lost and never arrives; frame 4 contains errors; the frames have variable transmission delay, but those that arrive do so in the order transmitted. The rectangles at the top of the figure indicate the duration over which the frame is being transmitted. Each arrow indicates the propagation of a frame from A to B, and the arrowhead indicates the time at which the frame is completely received; at this time the CRC can be recomputed and the frame accepted or not.

2.4.1 Stop-and-Wait ARQ

The simplest type of retransmission protocol is called *stop-and-wait*. The basic idea is to ensure that each packet has been received correctly before initiating transmission of the next packet. Thus, in transmitting packets from point A to B, the first packet is transmitted in the first frame, and then the sending DLC waits. If the frame is error-free, B sends an acknowledgment (called an ack) back to A; if the frame is an error frame, B sends a negative acknowledgment (called a nak) back to A. Since errors can occur from B to A as well as from A to B, the ack or nak is protected with a CRC.

If an error-free frame is received at B, and the corresponding ack frame to A is error-free, then A can start to send the next packet in a new frame. Alternatively, detected errors can occur either in the transmission of the frame or the return ack or nak, and in either case A resends the old packet in a new frame. Finally, if either the frame or the return ack or nak is lost, A must eventually time-out and resend the old packet.

Figure 2.18 illustrates a potential malfunction in such a strategy. Since delays are arbitrary, it is possible for node A to time-out and resend a packet when the first transmission and/or the corresponding ack is abnormally delayed. If B receives both transmissions of the given packet correctly, B has no way of knowing whether the second transmission is a new packet or a repetition of the old packet. One might think that B could simply compare the packets to resolve this issue, but as far as the DLC layer is concerned, packets are arbitrary bit strings and the first and second packets could be identical; it would be a violation of the principle of layering for the DLC layer to rely on higher layers to ensure that successive packets are different.

The simplest solution to this problem is for the sending DLC module (at A) to use a sequence number in the frame header to identify successive packets. Unfortunately, even the use of sequence numbers from A to B is not quite enough to ensure correct operation. The problem is that acks can get lost on the return channel, and thus when B gets the same packet correctly twice in a row, it has to send a new ack for the second reception (see Fig. 2.19). After transmitting the packet twice but receiving only one ack,

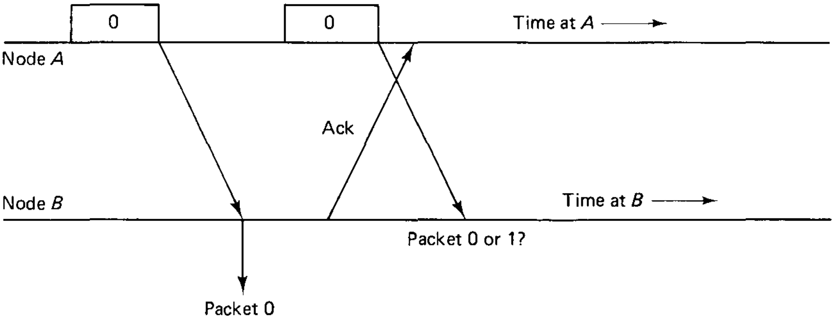


Figure 2.18 The trouble with unnumbered packets. If the transmitter at A times-out and sends packet 0 twice, the receiver at B cannot tell whether the second frame is a retransmission of packet 0 or the first transmission of packet 1.

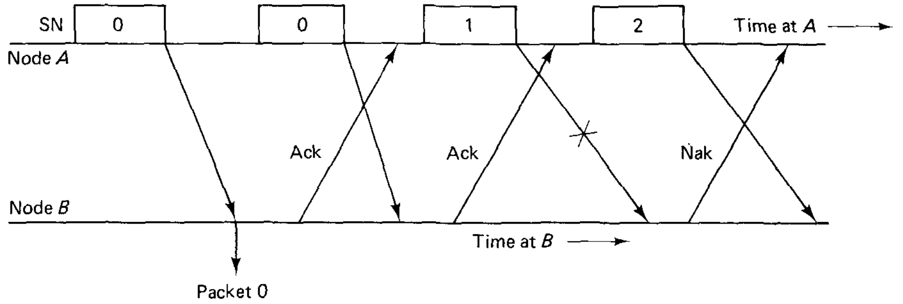


Figure 2.19 The trouble with unnumbered acks. If the transmitter at A times-out and sends packet 0 twice, node B can use the sequence numbers to recognize that packet 0 is being repeated. It must send an ack for both copies, however, and (since acks can be lost) the transmitter cannot tell whether the second ack is for packet 0 or 1.

node A could transmit the next packet in sequence, and then on receiving the second ack, could interpret that as an ack for the new packet, leading to a potential failure of the system.

To avoid this type of problem, the receiving DLC (at B), instead of returning ack or nak on the reverse link, returns the number of the next packet awaited. This provides all the information of the ack/nak, but avoids ambiguities about which frame is being acked. An equivalent convention would be to return the number of the packet just accepted, but this is not customary. Node B can request this next awaited packet upon the receipt of each packet, at periodic intervals, or at an arbitrary selection of times. In many applications, there is another stream of data from B to A, and in this case, the frames from B to A carrying requests for new A to B packets must be interspersed with data frames carrying data from B to A. It is also possible to "piggyback" these requests for new packets into the headers of the data frames from B to A (see Fig. 2.20), but as shown in Problem 2.38, this is often counterproductive for stop-and-wait ARQ. Aside from its effect on the timing of the requests from node B to A, the traffic from B to A does not affect the stop-and-wait strategy from A to B; thus, in what follows, we ignore this reverse traffic (except for the recognition that requests might be delayed). Figure 2.21 illustrates the flow of data from A to B and the flow of requests in the opposite direction.

We next specify the stop-and-wait strategy more precisely and then show that it works correctly. The correctness might appear to be self-evident, but the methodology of the demonstration will be helpful in understanding subsequent distributed algorithms. It

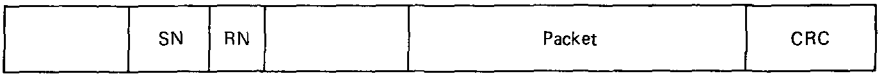


Figure 2.20 The header of a frame contains a field carrying the sequence number, SN, of the packet being transmitted. If piggybacking is being used, it also contains a field carrying the request number, RN, of the next packet awaited in the opposite direction.

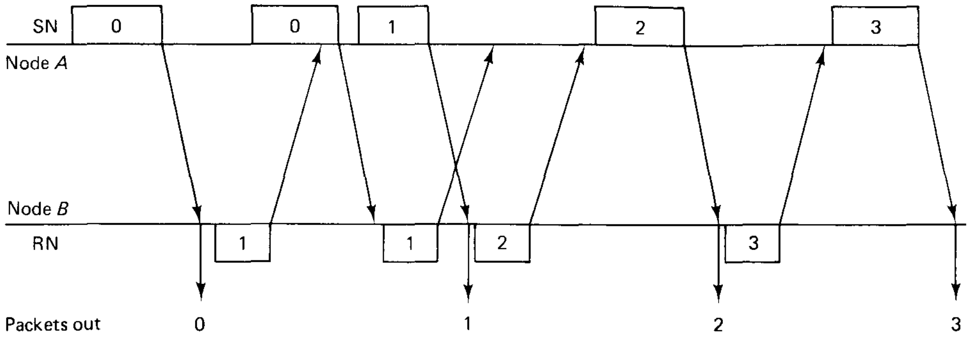


Figure 2.21 Example of use of sequence and request numbers for stop-and-wait transmission from A to B. Note that packet 0 gets repeated, presumably because node A times-out too soon. Note also that node A delays repeating packet 1 on the second request for it. This has no effect on the correctness of the protocol, but avoids unnecessary retransmissions.

will be seen that what is specified is not a single algorithm, but rather a class of algorithms in which the timing of frame transmissions is left unspecified. Showing that all algorithms in such a class are correct then allows one to specify the timing as a compromise between simplicity and efficiency without worrying further about correctness.

Assume that when the strategy is first started on the link, nodes A and B are correctly initialized in the sense that no frames are in transit on the link and that the receiver at B is looking for a frame with the same sequence number as the first frame to be transmitted from A. It makes no difference what this initial sequence number SN is as long as A and B agree on it, so we assume that $SN = 0$, since this is the conventional initial value.

The algorithm at node A for A-to-B transmission:

1. Set the integer variable SN to 0.
2. Accept a packet from the next higher layer at A; if no packet is available, wait until it is; assign number SN to the new packet.
3. Transmit the SNth packet in a frame containing SN in the sequence number field.
4. If an error-free frame is received from B containing a request number RN greater than SN, increase SN to RN and go to step 2. If no such frame is received within some finite delay, go to step 3.

The algorithm at node B for A-to-B transmission:

1. Set the integer variable RN to 0 and then repeat steps 2 and 3 forever.
2. Whenever an error-free frame is received from A containing a sequence number SN equal to RN, release the received packet to the higher layer and increment RN.
3. At arbitrary times, but within bounded delay after receiving any error-free data frame from A, transmit a frame to A containing RN in the request number field.

There are a number of conventional ways to handle the arbitrary delays between subsequent transmissions in the algorithm above. The usual procedure for node A (recall that we are discussing only the stop-and-wait strategy for A-to-B traffic) is to set a timer when a frame transmission begins. If the timer expires before receipt of a request for the next packet from B, the timer is reset and the packet is resent in a new frame. If a request for a new packet is received from B before the timer expires, the timer is disabled until A transmits the next packet. The same type of timer control can be used at node B. Alternatively, node B could send a frame containing a request for the awaited packet each time that it receives a frame from A. Also, if B is piggybacking its request numbers on data frames, it could simply send the current value of *RN* in each such frame. Note that this is not sufficient in itself, since communication from A to B would then halt in the absence of traffic from B to A; thus, node B must send nondata frames containing the *RN* values when there is no B-to-A data traffic. The important point is that whatever timing strategy is being used, it must guarantee that the intervening interval between repetitions in each direction is bounded.

Correctness of stop and wait We now go through an informal proof that this class of algorithms is correct in the sense that a never-ending stream of packets can be accepted from the higher layer at A and delivered to the higher layer at B in order and without repetitions or deletions. We continue to assume that all error frames are detected by the CRC. We also assume that there is some $q > 0$ such that each frame is received error-free with probability at least q. Finally, we recall the assumption that the link is initially empty, that the first packet from A has $SN = 0$, and that node B is initially awaiting the packet with $SN = 0$.

Proofs of this type usually break into two parts, characterized as *safety* and *liveness*. An algorithm is safe if it never produces an incorrect result, which in this case means never releasing a packet out of the correct order to the higher layer at B. An algorithm is live if it can continue forever to produce results (*i.e.*, if it can never enter a deadlock condition from which no further progress is possible). In this case liveness means the capability to continue forever to accept new packets at A and release them at B.

The safety property is self-evident for this algorithm; that is, node B is initially awaiting packet 0, and the only packet that can be released is packet 0. Subsequently (using induction if one wants to be formal), node B has released all the packets in order, up to, but not including, the current value of RN; packet RN is the only packet that it can next accept and release. When an error-free frame containing packet RN is received and released, the value of RN is incremented and the situation above is repeated with the new value of RN.

To see that the liveness property is satisfied, assume that node A first starts to transmit a given packet i at time t_1 (see Fig. 2.22). Let t_2 be the time at which this packet is received error-free and released to the higher layer at node B; let $t_2 = \infty$ if this event never occurs. Similarly, let t_3 be the time at which the sequence number at A is increased to $i + 1$, and let $t_3 = \infty$ if this never occurs. We will show that $t_1 < t_2 < t_3$ and that t_3 is finite. This is sufficient to demonstrate liveness, since using the argument repeatedly for $i = 0$, then $i = 1$, and so on, shows that each packet is transmitted with

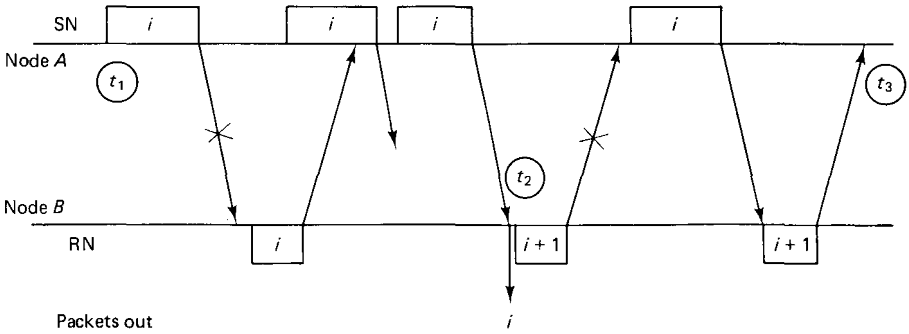


Figure 2.22 Times t_1, t_2, and t_3 when a packet is first placed in a frame for transmission at A, first received and released to the higher layer at B, and first acknowledged at A.

finite delay. Note that we cannot guarantee that the higher layer at A will always supply packets within finite delay, so that the notion of liveness here can only require finite delay given available packets to send.

Let $RN(t)$ be the value of the variable RN at node B as a function of time t and let $SN(t)$ be the corresponding value of SN at node A. It is seen directly from the algorithm statement that $SN(t)$ and $RN(t)$ are nondecreasing in t. Also, since $SN(t)$ is the largest request number received from B up to time t, $SN(t) \leq RN(t)$. By assumption, packet i has never been transmitted before time t_1, so (using the safety property) $RN(t_1) \leq i$. Since $SN(t_1) = i$, it follows that $SN(t_1) = RN(t_1) = i$. By definition of t_2 and t_3, $RN(t)$ is incremented to $i + 1$ at t_2 and $SN(t)$ is incremented to $i + 1$ at t_3. Using the fact that $SN(t) \leq RN(t)$, it follows that $t_2 < t_3$.

We have seen that node A transmits packet i repeatedly, with finite delay between successive transmissions, from t_1 until it is first received error-free at t_2. Since there is a probability $q > 0$ that each retransmission is received correctly, and retransmissions occur within finite intervals, an error-free reception eventually occurs and t_2 is finite. Node B then transmits frames carrying $RN = i+1$ from time t_2 until received error-free at time t_3. Since node A is also transmitting frames in this interval, the delay between subsequent transmissions from B is finite, and, since $q > 0$, t_3 eventually occurs; thus the interval from t_1 to t_3 is finite and the algorithm is live.

One trouble with the stop-and-wait strategy developed above is that the sequence and request numbers become arbitrarily large with increasing time. Although one could simply use a very large field for sending these numbers, it is nicer to send these numbers modulo some integer. Given our assumption that frames travel in order on the link, it turns out that a modulus of 2 is sufficient.

To understand why it is sufficient to send sequence numbers modulo 2, we first look more carefully at what happens in Fig. 2.22 when ordinary integers are used. Note that after node B receives packet i at time t_2, the subsequent frames received at B must all have sequence numbers i or greater (since frames sent before t_1 cannot arrive after t_2). Similarly, while B is waiting for packet $i + 1$, no packet greater than $i + 1$ can be sent [since $SN(t) \leq RN(t)$]. Thus, in the interval while $RN(t) = i + 1$, the received frames all carry sequence numbers equal to i or $i+1$. Sending the sequence number modulo 2 is sufficient to resolve this ambiguity. The same argument applies for all i. By the same argument, in the interval t_1 to t_3, while $SN(t)$ is equal to i, the request numbers received at A must be either i or $i+1$, so again sending *RN* modulo 2 is sufficient to resolve the ambiguity. Finally, since *SN* and *RN* need be transmitted only modulo 2, it is sufficient to keep track of them at the nodes only modulo 2.

Using modulo 2 values for *SN* and *RN*, we can view nodes A and B as each having two states (for purposes of A to B traffic), corresponding to the binary value of SN at node A and RN at node B. Thus, A starts in state 0; a transition to state 1 occurs upon receipt of an error-free request for packet 1 modulo 2. Note that A has to keep track of more information than just this state, such as the contents of the current packet and the time until time-out, but the binary state above is all that is of concern here.

Node B similarly is regarded as having two possible states, 0 and 1, corresponding to the number modulo 2 of the packet currently awaited. When a packet of the desired number modulo 2 is received, the DLC at B releases that packet to the higher layer and changes state, awaiting the next packet (see Fig. 2.23). The combined state of A and B is then initially (0,0); when the first packet is received error-free, the state of B changes to 1, yielding a combined state (0, 1). When A receives the new *RN* value (*i.e.*, 1), the state of A changes to 1 and the combined state becomes (1,1). Note that there is a fixed sequence for these combined states, (0,0), (0,1), (1,1), (1,0), (0,0), and so on, and that A and B alternate in changing states. It is interesting that at the instant of the transition from (0,0) to (0,1), B knows the combined state, but subsequently, up until the transition later from (1,1) to (1,0), it does not know the combined state (*i.e.*, B never knows that A has received the ack information until the next packet is received). Similarly, A knows the combined state at the instant of the transition from (0,1) to (1,1) and of the transition from (1,0) to (0,0). The combined state is always unknown to either A or B, and is frequently unknown to both. The situation here is very similar to that in the three-army problem discussed in Section 1.4. Here, however, information is transmitted even though the combined state information is never jointly known, whereas in the three-army problem, it is impossible to coordinate an attack because of the impossibility of obtaining this joint knowledge.

The stop-and-wait strategy is not very useful for modern data networks because of its highly inefficient use of communication links. In particular, it should be possible to do something else while waiting for an ack. There are three common strategies for extending the basic idea of stop-and-wait ARQ so as to achieve higher efficiency: go back n ARQ, selective repeat ARQ, and finally, the ARPANET ARQ.

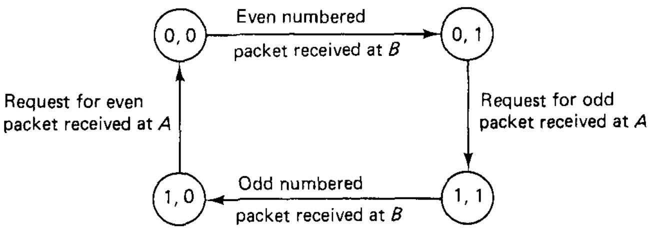


Figure 2.23 State transition diagram for stop-and-wait ARQ. The state is (*SN* mod 2 at A, *RN* mod 2 at B).

2.4.2 Go Back *n* ARQ

Go back n ARQ is the most widely used type of ARQ protocol; it appears in the various standard DLC protocols, such as HDLC, SDLC, ADCCP, and LAPB. It is not elucidating to know the meaning of these acronyms, but in fact, these standards are almost the same. They are discussed in Section 2.6, and some of their differences are mentioned there. Go back n is also the basis of most error recovery procedures at the transport layer.

The basic idea of go back n is very simple. Incoming packets to a transmitting DLC module for a link from A to B are numbered sequentially, and this sequence number SN is sent in the header of the frame containing the packet. In contrast to stop-and-wait ARQ, several successive packets can be sent without waiting for the next packet to be requested. The receiving DLC at B operates in essentially the same way as stop-and-wait ARQ. It accepts packets only in the correct order and sends request numbers RN back to A; the effect of a given request RN is to acknowledge all packets prior to RN and to request transmission of packet RN.

The go back number $n \geq 1$ in a go back n protocol is a parameter that determines how many successive packets can be sent in the absence of a request for a new packet. Specifically, node A is not allowed to send packet $i+n$ before i has been acknowledged (*i.e.*, before $i+1$ has been requested). Thus, if i is the most recently received request from node B, there is a "window" of n packets, from i to $i+n-1$, that the transmitter is allowed to send. As successively higher-numbered requests are received from B, this window slides upward; thus go back n protocols are often called *sliding window* ARQ protocols.

Figure 2.24 illustrates the operation of go back 7 ARQ when piggybacking of request numbers is being used and when there are no errors and a constant supply of traffic. Although the figure portrays data traffic in both directions, the flow of sequence numbers is shown in one direction and request numbers in the other. Note that when the first frame from A (containing packet 0) is received at B, node B is already in the middle of transmitting its second frame. The piggybacked request number at node B is traditionally in the frame header, and the frame is traditionally completely assembled before transmission starts. Thus when packet 0 is received from A, it is too late for node B to request packet 1 in the second frame, so that $RN = 1$ does not appear until the third frame from B. When this frame is completely received at A, the window at A "slides up" from $[0, 6]$ to $[1, 7]$.

Note that even in the absence of transmission errors, there are several sources of delay between the time that a packet is first assembled into a frame at A and the time when A receives an acknowledgment of the packet. First there is the transmission time of the frame, then the propagation delay, then the wait until the frame in transmission at B is completed, then the transmission time of the frame carrying the acknowledgment, and finally, the B-to-A propagation delay; the effect of these delays is discussed later.

Figure 2.25 shows the effect of error frames on go back 4 ARQ. The second frame from A, carrying packet 1, is received in error at node B. Node B continues to look for packet 1 and to transmit $RN = 1$ in frames from B to A. Packets 2, 3, and 4 from A

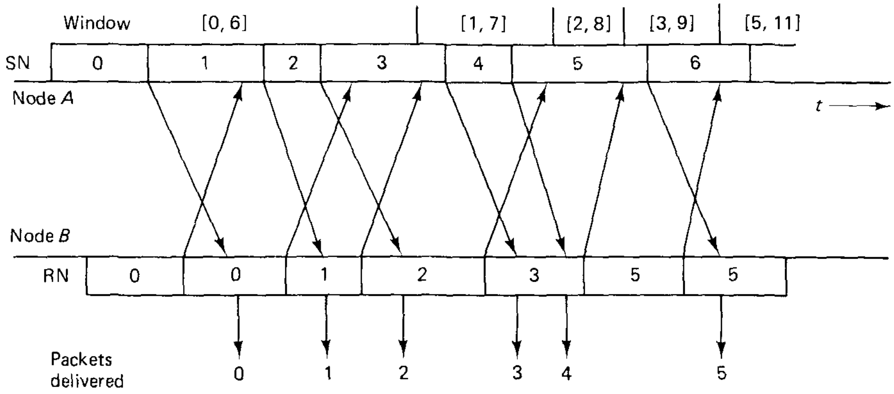


Figure 2.24 Example of go back 7 protocol for A-to-B traffic. Both nodes are sending data, but only the sequence numbers are shown for the frames transmitted from A and only the request numbers are shown for the frames from B. When packet 0 is completely received at B, it is delivered to the higher layer, as indicated at the lower left of the figure. At this point, node B wants to request packet 1, so it sends $RN = 1$ in the next outgoing frame. When that outgoing frame is completely received at A, node A updates its window from [0,6] to [1,7]. Note that when packets 3 and 4 are both received at B during the same frame transmission at B, node B awaits packet 5 and uses $RN = 5$ in the next frame from B to acknowledge both packets 3 and 4.

all arrive at B in error-free frames but are not accepted since node B is looking only for packet 1. One might think that it would be more efficient for node B to buffer packets 2, 3, and 4, thus avoiding the necessity for A to retransmit them after packet 1 is finally retransmitted. Such a buffering strategy is indeed possible and is called *selective repeat* ARQ; this is discussed in Section 2.4.3, but, by definition, go back n does not include this possibility.

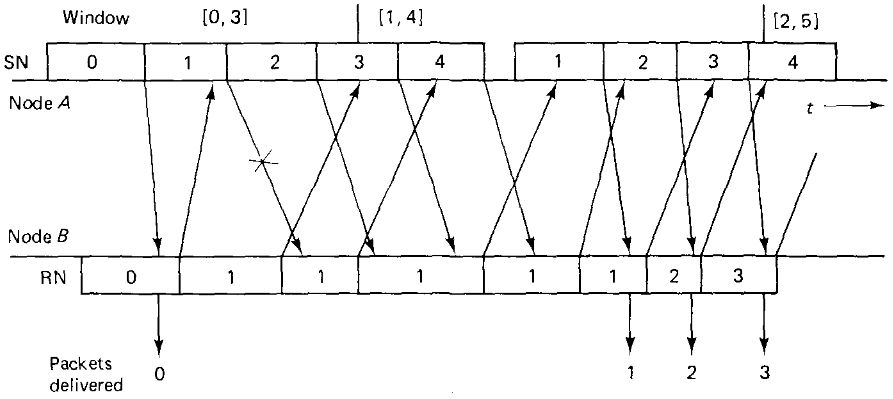


Figure 2.25 Effect of a transmission error on go back 4. Packet 1 is received in error at B, and node B continues to request packet 1 in each reverse frame until node A transmits its entire window, times-out, and goes back to packet 1.

After node A has completed transmission of the frame containing packet 4 (see Fig. 2.25), it has exhausted its window, which is still [1, 4]; node A then goes back and retransmits packet 1. The figure shows the retransmission of packet 1 after a time-out. There are many possible strategies for the timing of retransmissions within the go back n protocol, and for this reason, go back n is really an entire class of algorithms (just like stop and wait). In go back n, however, not only is the timing of transmissions unspecified, but also the selection of a packet within the window. The class of algorithms will be specified precisely after going through several more examples.

Figure 2.26 illustrates the effect of error frames in the reverse direction (from node B to A). Such an error frame need not slow down transmission in the A-to-B direction, since a subsequent error-free frame from B to A can acknowledge the packet and perhaps some subsequent packets (*i.e.*, the third frame from B to A in the figure acknowledges both packets 1 and 2). On the other hand, with a small window and long frames from B to A, an error frame from B to A can delay acknowledgments until after all the packets in the window are transmitted, thus causing A to either wait or to go back. Note that when the delayed acknowledgments get through, node A can jump forward again (from packet 3 to 5 in Fig. 2.26).

Finally, Fig. 2.27 shows that retransmissions can occur even in the absence of any transmission errors. This happens particularly in the case of short frames in one direction and long frames in the other. We discuss the impact of this phenomenon on the choice of window size and frame length later.

Rules followed by transmitter and receiver in go back *n* We now specify the precise rules followed by the class of go back n protocols and then, in the next

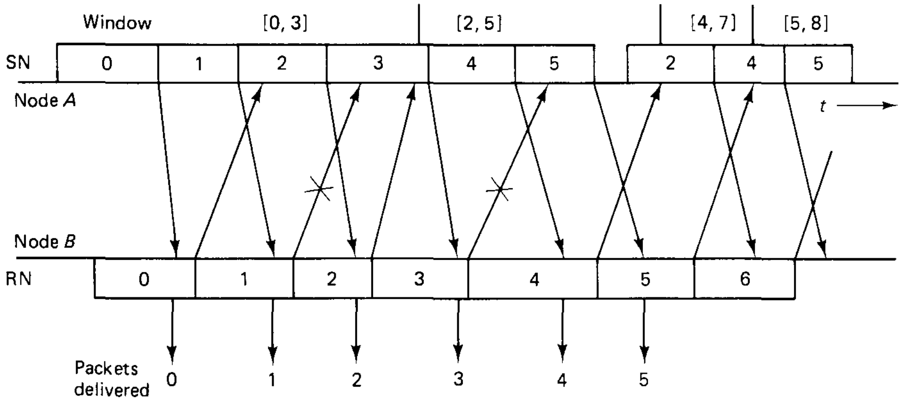


Figure 2.26 Effect of transmission errors in the reverse direction for go back 4. The first error frame, carrying $RN = 1$, causes no problem since it is followed by an error-free frame carrying $RN = 2$ and this frame reaches A before packet number 3, (*i.e.*, the last packet in the current window at A) has completed transmission and before a time-out occurs. The second error frame, carrying $RN = 3$, causes retransmissions since the following reverse frame is delayed until after node A sends its entire window and times-out. This causes A to go back and retransmit packet 2.

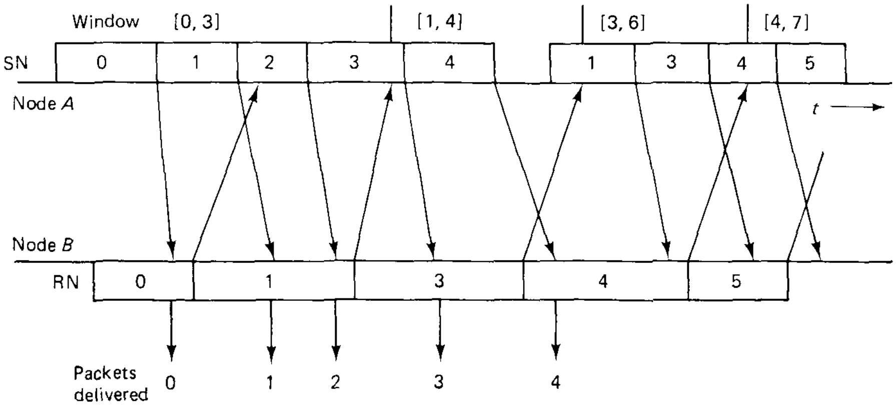


Figure 2.27 Effect of delayed feedback for go back 4. The frames in the B-to-A direction are much longer than those in the A-to-B direction, thus delaying the request numbers from getting back to A. The request for packet 1 arrives in time to allow packet 4 to be sent, but after sending packet 4, node A times-out and goes back to packet 1.

subsection, demonstrate that the algorithms in the class work correctly. We assume here that the sequence numbers and request numbers are integers that can increase without bound. The more practical case in which SN and RN are taken modulo some integer m is considered later.

The rules given here do not treat the initialization of the protocol. We simply assume that initially there are no frames in transit on the link, that node A starts with the transmission of packet number 0, and that node B is initially awaiting packet number 0. How to achieve such an initialization is discussed in Section 2.7.

The transmitter uses two integer variables, SN_{min} and SN_{max}, to keep track of its operations. SN_{min} denotes the smallest-numbered packet that has not yet been acknowledged (*i.e.*, the lower end of the window). SN_{max} denotes the number of the next packet to be accepted from the higher layer. Thus the DLC layer is attempting to transmit packets SN_{min} to $SN_{max} - 1$. Conceptually we can visualize the DLC layer as storing these packets, but it makes no difference where they are stored physically as long as they are maintained somewhere at the node for potential retransmission.

The go back n algorithm at node A for A-to-B transmission:

1. Set the integer variables SN_{min} and SN_{max} to 0.
2. Do steps 3, 4, and 5 repeatedly in any order. There can be an arbitrary but bounded delay between the time when the conditions for a step are satisfied and when the step is executed.
3. If $SN_{max} < SN_{min} + n$, and if a packet is available from the higher layer, accept a new packet into the DLC, assign number SN_{max} to it, and increment SN_{max}.
4. If an error-free frame is received from B containing a request number RN greater than SN_{min}, increase SN_{min} to RN.

5. If $SN_{min} < SN_{max}$ and no frame is currently in transmission, choose some number SN, $SN_{min} \leq SN < SN_{max}$; transmit the SNth packet in a frame containing SN in the sequence number field. At most a bounded delay is allowed between successive transmissions of packet SN_{min} over intervals when SN_{min} does not change.

The go back n algorithm at node B for A-to-B transmission:

1. Set the integer variable RN to 0 and repeat steps 2 and 3 forever.
2. Whenever an error-free frame is received from A containing a sequence number SN equal to RN, release the received packet to the higher layer and increment RN.
3. At arbitrary times, but within bounded delay after receiving any error-free data frame from A, transmit a frame to A containing RN in the request number field.

There are many conventional ways of handling the timing and ordering of the various operations in the algorithm above. Perhaps the simplest is for node A to set a timer whenever a packet is transmitted. If the timer expires before that packet is acknowledged (*i.e.*, before SN_{min} increases beyond that packet number), the packet is retransmitted. Sometimes when this approach is used, the transmitter, after going back and retransmitting SN_{min}, simply retransmits subsequent packets in order up to $SN_{max} - 1$, whether or not subsequent request numbers are received. For example, at the right-hand edge of Fig. 2.26, the transmitter might have followed packet 3 with 4 rather than 5. In terms of the algorithm as stated, this corresponds to the transmitter delaying the execution of step 4 while in the process of retransmitting a window of packets. Another possibility is for node A to cycle back whenever all the available packets in the window have been transmitted. Also, A might respond to a specific request from B for retransmission; such extra communication between A and B can be considered as part of this class of protocols in the sense that it simply guides the available choices within the algorithm.

Perhaps the simplest approach to timing at node B is to piggyback the current value of RN in each data frame going from B to A. When there are no data currently going from B to A, a nondata frame containing RN should be sent from B to A whenever a data frame is received from A.

Correctness of go back *n* We first demonstrate the correctness of this class of algorithms under our current assumptions that SN and RN are integers; we then show that correctness is maintained if SN and RN are integers modulo m, for m strictly greater than the go back number n. The correctness demonstration when SN and RN are integers is almost the same as the demonstration in Section 2.4.1 for stop and wait. In particular, we start by assuming that all frames with transmission errors are detected by the CRC, that there is some $q > 0$ such that each frame is received error-free with probability at least q, and that the system is correctly initialized in the sense that there are no frames on the link and that nodes A and B both start at step 1 of their respective algorithms.

The safety property of the go back n algorithm is exactly the same as for stop and wait. In particular, node B releases packets to the higher layer in order, using the variable RN to track the next packet awaited. To verify the liveness property, assume that i is the value of SN_{min} at node A at a given time t_1 (see Fig. 2.28). Let t_2 be the time at which packet i is received error-free and released to the higher layer at node B; let $t_2 = \infty$ if this event never occurs. Similarly, let t_3 be the time at which SN_{min} is increased beyond i and let $t_3 = \infty$ if this never occurs. We will show that t_3 is finite and that $t_1 < t_3$ and $t_2 < t_3$. This is sufficient to demonstrate liveness, since using the argument for each successive value of SN_{min} shows that each packet is transmitted with finite delay.

Let $RN(t)$ be the value of the variable RN at node B as a function of time t and let $SN_{min}(t)$ be the corresponding value of SN_{min} at node A. It is seen directly from the algorithm statement that $SN_{min}(t)$ and $RN(t)$ are nondecreasing in t. Also, since $SN_{min}(t)$ is the largest request number (if any) received from B up to time t, $SN_{min}(t) \leq RN(t)$. By definition of t_2 and t_3, $RN(t)$ is incremented to $i + 1$ at t_2 and $SN_{min}(t)$ is increased beyond i at t_3. Using the fact that $SN_{min}(t) \leq RN(t)$, it follows that $t_2 < t_3$. Note that it is possible that $t_2 < t_1$, since packet i might have been received error-free and released at B before time t_1 and even before SN_{min} became equal to i.

From the algorithm statement, node A transmits packet i repeatedly, with finite delay between successive transmissions, from t_1 until t_3. If $t_1 < t_2$, then $RN(t) = i$ for $t_1 \leq t \leq t_2$, so the first error-free reception of packet i after t_1 will be accepted and released to the higher layer at B. Since $t_2 < t_3$, node A will retransmit packet i until this happens. Since there is a probability $q > 0$ that each retransmission is received correctly, and retransmissions occur within finite intervals, the time from t_1 to t_2 is finite. Node

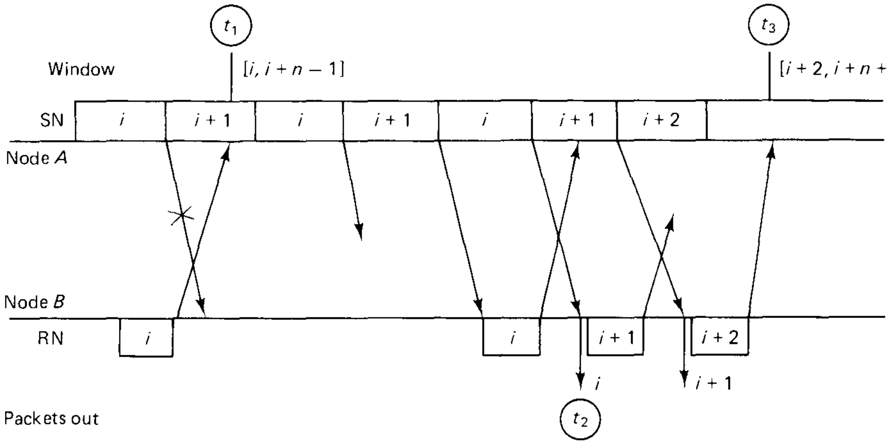


Figure 2.28 SN_{min} is i at time t_1. Packet i is then released to the higher layer at B at time t_2 and the window is increased at A at time t_3.

B (whether $t_1 < t_2$, or vice versa) transmits frames carrying $RN \geq i+1$ from time t_2 until some such frame is received error-free at A at time t_3. Since node A is also transmitting frames in this interval, the delay between subsequent transmissions from B is finite, and, since $q > 0$, the interval from t_2 to t_3 is finite. Thus the interval from t_1 to t_3 is finite and the algorithm is live.

It will be observed that no assumption was made in the demonstration above about the frames traveling in order on the links, and the algorithm above operates correctly even when frames get out of order. When we look at error recovery at the transport layer, the role of a link will be played by a subnetwork and the role of a frame will be played by a packet. For datagram networks, packets can get out of order, so this generality will be useful.

Go back *n* with modulus *m* > *n*. It will now be shown that if the sequence number SN and the request number RN are sent modulo m, for some m strictly greater than the go back number n, the correctness of go back n is maintained as long as we reimpose the condition that frames do not get out of order on the links. To demonstrate this correctness, we first look more carefully at the ordering of events when ordinary integers are used for SN and RN.

Consider the transmission of an arbitrary frame from node A to B. Suppose that the frame is generated at time t_1 and received at t_2 (see Fig. 2.29). The sequence number SN of the frame must lie in node A's window at time t_1, so

$$SN_{min}(t_1) \leq SN \leq SN_{min}(t_1) + n - 1 \qquad (2.24)$$

Also, as shown in the figure,

$$SN_{min}(t_1) \leq RN(t_2) \leq SN_{min}(t_1) + n \qquad (2.25)$$

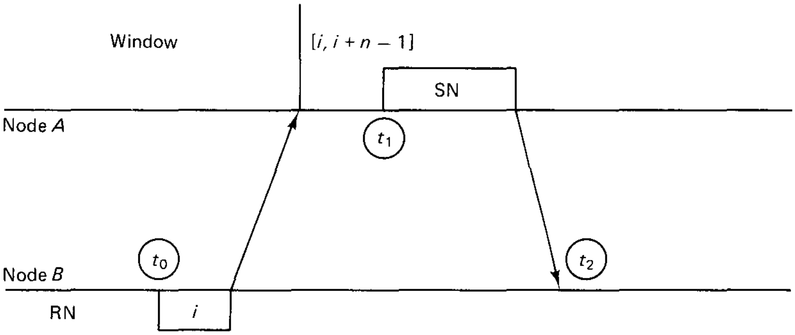


Figure 2.29 Let t_1 and t_2 be the times at which a given frame is generated at A and received at B respectively. The sequence number in the frame satisfies $SN_{min}(t_1) \leq SN \leq SN_{min}(t_1) + n - 1$. The value i of $SN_{min}(t_1)$ is equal to the last received value of RN, which is $RN(t_0) \leq RN(t_2)$. Thus $SN_{min}(t_1) \leq RN(t_2)$, which is the left side of Eq. (2.25). Conversely, no frame with sequence number $SN_{min}(t_1)+n$ can have been sent before t_1 since this value is beyond the upper limit of the window. Since frames travel in order on the link, no frame with this number has arrived at B before t_2, and $RN(t_2) \leq SN_{min}(t_1) + n$, which is the right side of Eq. (2.25).

We see from Eqs. (2.24) and (2.25) that SN and $RN(t_2)$ are both contained in the interval from $SN_{min}(t_1)$ to $SN_{min}(t_1) + n$, and thus must satisfy

$$|RN(t_2) - SN| \le n \tag{2.26}$$

Now suppose that when packet number SN is sent, the accompanying sequence number is sent modulo m, and let sn denote SN mod m. Step 3 of the algorithm at node B must then be modified to: If an error-free frame is received from A containing a sequence number sn equal to RN mod m, release the received packet to the higher layer and increment RN. Since $m > n$ by assumption, we see from Eq. (2.26) that $sn = RN$ mod m will be satisfied if and only if the packet number SN is equal to RN; thus, the algorithm still works correctly.

Next consider the ordering of arriving request numbers (using ordinary integers) relative to the window at node A. From Fig. 2.30, we see that

$$SN_{min}(t_2) \le RN \le SN_{min}(t_2) + n \tag{2.27}$$

Now suppose that RN is sent modulo m, and let $rn = RN$ mod m. Step 4 of the algorithm at node A must then be modified to: If an error-free frame is received from B containing $rn \neq SN_{min}$ mod m, then increment SN_{min} until $rn = SN_{min}$ mod m. Because of the range of RN in Eq. (2.27), we see that this new rule is equivalent to the old rule, and it is sufficient to send request numbers modulo m. At this point, however, we see that it is unnecessary for SN_{min}, SN_{max}, and RN to be saved at nodes A and B as ordinary integers; everything can be numbered modulo m, and the algorithm has been demonstrated to work correctly for $m > n$.

For completeness, we restate the algorithm for operation modulo m; since all numbers are taken modulo m, we use capital letters for these numbers.

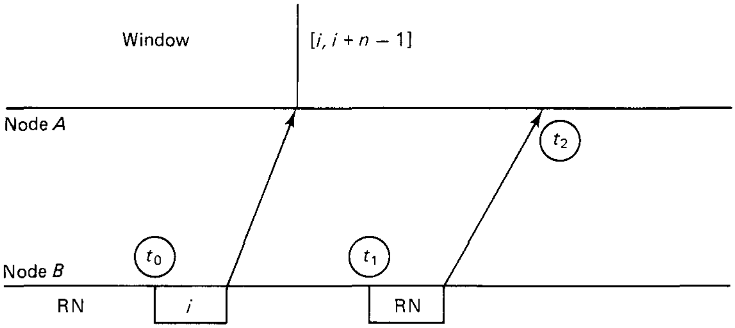


Figure 2.30 Let t_1 and t_2 be the times at which a given frame with request number RN is generated at B and received at A, respectively. Let $SN_{min}(t_2) = i$ be the lower edge of the window at t_2 and let t_0 be the generation time of the frame from B that caused the window at A to move to $[i, i+n-1]$. Since the frames travel in order on the link, $t_0 < t_1$, so $i \le RN$ and thus $SN_{min}(t_2) \le RN$. Similarly, node A cannot have sent packet $i+n$ before t_2, so it certainly cannot have been received before t_1. Thus $RN \le SN_{min}(t_2) + n$.

The go back n algorithm at node A for modulo m operation, $m > n$:

1. Set the modulo m variables SN_{min} and SN_{max} to 0.
2. Do steps 3, 4, and 5 repeatedly in any order. There can be an arbitrary but bounded delay between the time when the conditions for a step are satisfied and when the step is executed.
3. If $(SN_{max} - SN_{min}) \bmod m < n$, and if a packet is available from the higher layer, accept a new packet into the DLC, assign number SN_{max} to it, and increment SN_{max} to $(SN_{max} + 1) \bmod m$.
4. If an error-free frame is received from B containing a request number RN, and $(RN - SN_{min}) \bmod m \leq (SN_{max} - SN_{min}) \bmod m$, set SN_{min} to equal RN.
5. If $SN_{min} \neq SN_{max}$ and no frame is currently in transmission, choose some number SN such that $(SN - SN_{min}) \bmod m < (SN_{max} - SN_{min}) \bmod m$; transmit packet SN in a frame containing SN in the sequence number field.

The go back n algorithm at node B for modulo m operation, $m > n$:

1. Set the modulo m variable RN to 0.
2. Whenever an error-free frame is received from A containing a sequence number SN equal to RN, release the received packet to the higher layer and increment RN to $(RN + 1) \bmod m$.
3. At arbitrary times, but within bounded delay after receiving any error-free data frame from A, transmit a frame to A containing RN in the request number field.

Efficiency of go back *n* implementations Retransmissions, or delays waiting for time-outs, occur in go back n ARQ for the following three reasons: first, errors in the forward direction, second, errors in the feedback direction, and third, longer frames in the feedback than in the forward direction. These will be discussed in reverse order.

The likelihood of retransmissions caused by long reverse frames can be reduced by increasing the go back number n. Unfortunately, the normal value of modulus in the standard protocols is $m = 8$, which constrains n to be at most 7. Figure 2.31 illustrates that even in the absence of propagation delay, reverse frames more than three times the length of forward frames can cause retransmissions for $n = 7$. Problem 2.23 also shows that if frames have lengths that are exponentially distributed, with the same distribution

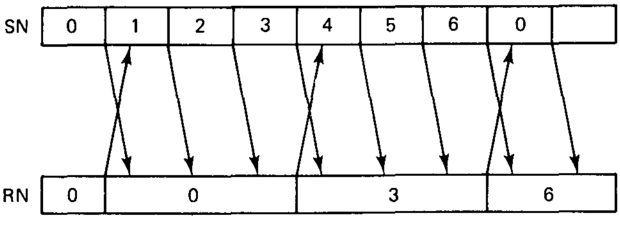


Figure 2.31 Go back 7 ARQ with long frames in the reverse direction. Note that the ack for packet 1 has not arrived at the sending side by the time packet 6 finishes transmission, thereby causing a retransmission of packet 0.

in each direction, the probability p that a frame is not acked by the time the window is exhausted is given by

$$p = (1 + n)2^{-n} \tag{2.28}$$

For $n = 7$, p is equal to 1/16. Frames normally have a maximum permissible length (and a minimum length because of the control overhead), so in practice p is somewhat smaller. However, links sometimes carry longer frames in one direction than in the other, and in this case, there might be considerable waste in link utilization for $n = 7$. When propagation delay is large relative to frame length (as, for example, on high-speed links and satellite links), this loss of utilization can be quite serious. Fortunately, the standard protocols have an alternative choice of modulus as $m = 128$.

When errors occur in a reverse frame, the acknowledgment information is postponed for an additional reverse frame. This loss of line utilization in one direction due to errors in the other can also be avoided by choosing n sufficiently large.

Finally, consider the effect of errors in the forward direction. If n is large enough to avoid retransmissions or delays due to large propagation delay and long frames or errors in the reverse direction, and if the sending DLC waits to exhaust its window of n packets before retransmitting, a large number of packets are retransmitted for each forward error. The customary solution to this problem is the use of time-outs. In its simplest version, if a packet is not acknowledged within a fixed time-out period after transmission, it is retransmitted. This time-out should be chosen long enough to include round-trip propagation and processing delay plus transmission time for two maximum-length packets in the reverse direction (one for the frame in transit when a packet is received, and one to carry the new RN). In a more sophisticated version, the sending DLC, with knowledge of propagation and processing delays, can determine which reverse frame should carry the ack for a given packet; it can go back if that frame is error free and fails to deliver the ack.

In a more fundamental sense, increasing link utilization and decreasing delay is achieved by going back quickly when a forward error occurs, but avoiding retransmissions caused by long frames and errors in the reverse direction. One possibility here is for the receiving DLC to send back a short supervisory frame upon receiving a frame in error. This allows the sending side to go back much sooner than if RN were simply piggybacked on a longer reverse data frame. Another approach is to insert RN in the trailer of the reverse frame, inserting it before the CRC and inserting it at the last moment, after the packet part of the frame has been sent. This cannot be done in the standard DLC protocols, but would have the effect of reducing the feedback delay by almost one frame length.

It is not particularly difficult to invent new ways of reducing both feedback delay and control overhead in ARQ strategies. One should be aware, however, that it is not trivial to ensure the correctness of such strategies. Also, except in special applications, improvements must be substantial to outweigh the advantages of standardization.

2.4.3 Selective Repeat ARQ

Even if unnecessary retransmissions are avoided, go back n protocols must retransmit at least one round-trip-delay worth of frames when a single error occurs in an awaited packet. In many situations, the probability of one or more errors in a frame is 10^{-4} or less, and in this case, retransmitting many packets for each frame in error has little effect on efficiency. There are some communication links, however, for which small error probabilities per frame are very difficult to achieve, even with error correction in the modems. For other links (*e.g.*, high-speed links and satellite links), the number of frames transmitted in a round- trip delay time is very large. In both these cases, selective repeat ARQ can be used to increase efficiency.

The basic idea of selective repeat ARQ for data on a link from A to B is to accept out-of-order packets and to request retransmissions from A only for those packets that are not correctly received. There is still a go back number, or window size, n, specifying how far A can get ahead of RN, the lowest-numbered packet not yet correctly received at B.

Note that whatever ARQ protocol is used, only error-free frames can deliver packets to B, and thus, if p is the probability of frame error, the expected number η of packets delivered to B per frame from A to B is bounded by

$$\eta \leq 1 - p \tag{2.29}$$

As discussed later, this bound can be approached (in principle) by selective repeat ARQ; thus $1 - p$ is sometimes called the throughput of ideal selective repeat. Ideal go back n ARQ can similarly be defined as a protocol that retransmits the packets in one round-trip delay each time that a frame carrying the packet awaited by the receiving DLC is corrupted by errors. The throughput of this ideal is shown in Problem 2.26 to be

$$\eta \leq \frac{1-p}{1+p\beta} \tag{2.30}$$

where β is the expected number of frames in a round-trip delay interval. This indicates that the increase in throughput available with selective repeat is significant only when $p\beta$ is appreciable relative to 1.

The selective repeat algorithm at node A, for traffic from A to B, is the same as the go back n algorithm, except that (for reasons soon to be made clear) the modulus m must satisfy $m \geq 2n$. At node B, the variable RN has the same significance as in go back n; namely, it is the lowest-numbered packet (or the lowest-numbered packet modulo m) not yet correctly received. In the selective repeat algorithm, node B accepts packets anywhere in the range RN to $RN + n - 1$. The value of RN must be sent back to A as in go back n, either piggybacked on data frames or sent in separate frames. Usually, as discussed later, the feedback from node B to A includes not only the value of RN but also additional information about which packets beyond RN have been correctly received. In principle, the DLC layer still releases packets to the higher layer in order, so that the accepted out-of-order packets are saved until the earlier packets are accepted and released.

We now assume again that frames do not get out of order on the links and proceed to see why a larger modulus is required for selective repeat than for go back n. Assume that a frame is received at node B at some given time t_2 and that the frame was generated at node A at time t_1. If SN and RN are considered as integers, Eqs. (2.24) and (2.25) are still valid, and we can conclude from them that the sequence number SN in the received frame must satisfy

$$RN(t_2) - n \leq SN \leq RN(t_2) + n - 1 \tag{2.31}$$

If sequence numbers are sent mod m, and if packets are accepted in the range $RN(t_2)$ to $RN(t_2) + n - 1$, it is necessary for node B to distinguish values of SN in the entire range of Eq. (2.31). This means that the modulus m must satisfy

$$m \geq 2n, \qquad \text{for selective repeat} \tag{2.32}$$

With this change, the correctness of this class of protocols follows as before. The real issue with selective repeat, however, is using it efficiently to achieve throughputs relatively close to the ideal $1 - p$. Note first that using RN alone to provide acknowledgment information is not very efficient, since if several frame errors occur in one round-trip delay period, node A does not find out about the second frame error until one round-trip delay after the first error is retransmitted. There are several ways of providing the additional acknowledgment information required by A. One is for B to send back the lowest j packet numbers that it has not yet received; j should be larger than $p\beta$ (the expected number of frame errors in a round-trip delay time), but is limited by the overhead required by the feedback. Another possibility is to send RN plus an additional k bits (for some constant k), one bit giving the ack/nak status of each of the k packets after RN.

Assume now that the return frames carry sufficient information for A to determine, after an expected delay of β frames, whether or not a packet was successfully received. The typical algorithm for A is then to repeat packets as soon as it is clear that the previous transmission contained errors; if A discovers multiple errors simultaneously, it retransmits them in the order of their packet numbers. When there are no requested retransmissions, A continues to send new packets, up to $SN_{max} - 1$. At this limit, node A can wait for some time-out or immediately go back to SN_{min} to transmit successive unacknowledged packets.

Node A acts like an ideal selective repeat system until it is forced to wait or go back from $SN_{max} - 1$. When this happens, however, the packet numbered SN_{min} must have been transmitted unsuccessfully about n/β times. Thus, by making n large enough, the probability of such a go back can be reduced to negligible value. There are two difficulties with very large values of n (assuming that packets must be reordered at the receiving DLC). The first is that storage must be provided at B for all of the accepted packets beyond RN. The second is that the large number of stored packets are all delayed waiting for RN.

The amount of storage provided can be reduced to $n - \beta$ without much harm, since whenever a go back from $SN_{max} - 1$ occurs, node A can send no new packets beyond $SN_{max} - 1$ for a round-trip delay; thus, it might as well resend the yet-unacknowledged packets from to $SN_{max} - \beta$ to $SN_{max} - 1$; this means that B need not save these packets. The value of n can also be reduced, without increasing the probability of go back, by retransmitting a packet several times in succession if the previous transmission contained errors. For example, Fig. 2.32 compares double retransmissions with single retransmissions for $n = 2\beta + 2$. Single retransmissions fail with probability p and cause β extra retrans-

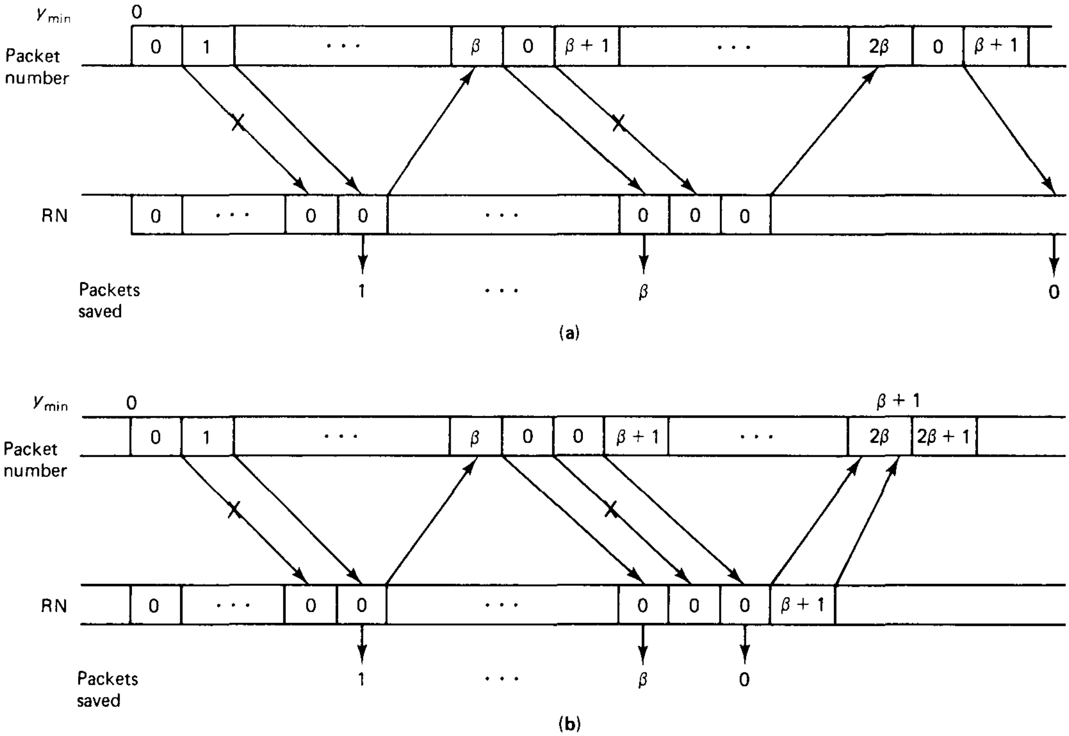


Figure 2.32 Selective repeat ARQ with $n = 2\beta + 2$ and receiver storage for $\beta + 1$ packets. (a) Note the wasted transmissions if a given packet (0) is transmitted twice with errors. (b) Note that this problem is cured, at the cost of one extra frame, if the second transmission of packet 0 is doubled. Feedback contains not only RN but additional information on accepted packets.

missions; double retransmissions rarely fail, but always require one extra retransmission. Thus, double retransmissions can increase throughput if $pn > 1$, but might be desirable in other cases to reduce the variability of delay. (See [Wel82] for further analysis.)

2.4.4 ARPANET ARQ

The ARPANET achieves efficiency by using eight stop-and-wait strategies in parallel, multiplexing the bit pipe between the eight. That is, each incoming packet is assigned to one of eight virtual channels, assuming that one of the eight is idle; if all the virtual channels are busy, the incoming packet waits outside the DLC (see Fig. 2.33). The busy virtual channels are multiplexed on the bit pipe in the sense that frames for the different virtual channels are sent one after the other on the link. The particular order in which frames are sent is not very important, but a simple approach is to send them in round-robin order. If a virtual channel's turn for transmission comes up before an ack has been received for that virtual channel, the packet is sent again, so that the multiplexing removes the need for any time-outs. (The actual ARPANET protocol, however, does use

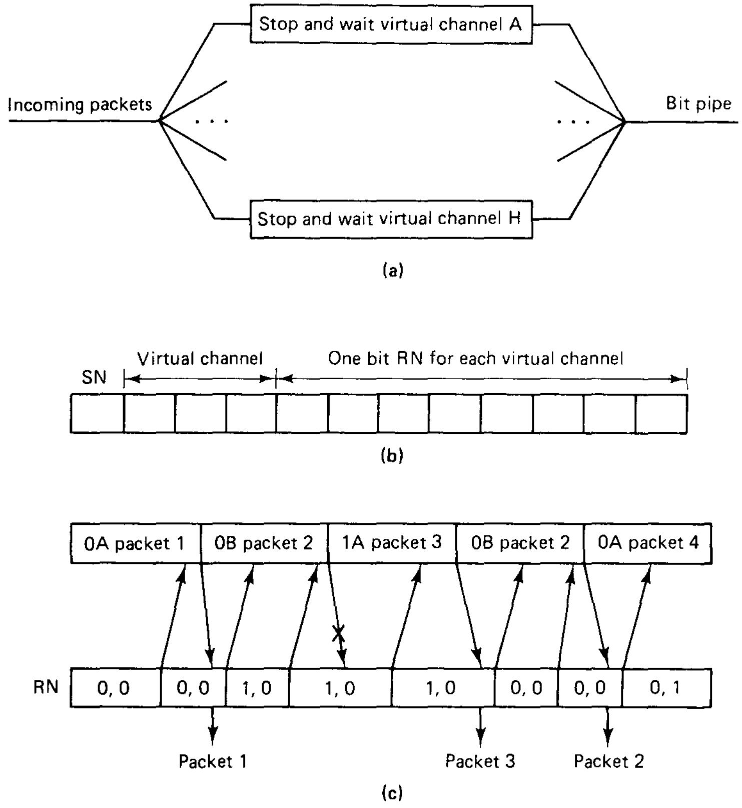


Figure 2.33 ARPANET ARQ. (a) Eight multiplexed, stop-and-wait virtual channels. (b) Bits in the header for ARQ control. (c) Operation of multiplexed stop and wait for two virtual channels. Top-to-bottom frames show SN and the channel number, and bottom-to-top frames show RN for both channels. The third frame from bottom to top acks packet 1 on the A channel.

time-outs.) When an ack is received for a frame on a given virtual channel, that virtual channel becomes idle and can accept a new packet from the higher layer.

Somewhat more overhead is required here than in the basic stop-and-wait protocol. In particular, each frame carries both the virtual channel number (requiring three bits) and the sequence number modulo 2 (*i.e.*, one bit) of the packet on that virtual channel. The acknowledgment information is piggybacked onto the frames going in the opposite direction. Each such frame, in fact, carries information for all eight virtual channels. In particular, an eight-bit field in the header of each return frame gives the number modulo 2 of the awaited packet for each virtual channel.

One of the desirable features of this strategy is that the ack information is repeated so often (*i.e.*, for all virtual channels in each return frame) that relatively few retransmissions are required because of transmission errors in the reverse direction. Typically, only one retransmission is required for each frame in error in the forward direction. The undesirable feature of the ARPANET protocol is that packets are released to the higher layer at the receiving DLC in a different order from that of arrival at the sending DLC. The DLC layer could, in principle, reorder the packets, but since a packet on one virtual channel could be arbitrarily delayed, an arbitrarily large number of later packets might have to be stored. The ARPANET makes no effort to reorder packets on individual links, so this protocol is not a problem for ARPANET. We shall see later that the lack of ordering on links generates a number of problems at higher layers. Most modern networks maintain packet ordering for this reason, and consequently do not use this protocol despite its high efficiency and low overhead. For very poor communication links, where efficiency and overhead are very important, it is a reasonable choice.

2.5 FRAMING

The problem of framing is that of deciding, at the receiving DLC, where successive frames start and stop. In the case of a synchronous bit pipe, there is sometimes a period of idle fill between successive frames, so that it is also necessary to separate the idle fill from the frames. For an intermittent synchronous bit pipe, the idle fill is replaced by dead periods when no bits at all arrive. This does not simplify the problem since, first, successive frames are often transmitted with no dead periods in between, and second, after a dead period, the modems at the physical layer usually require some idle fill to reacquire synchronization.

There are three types of framing used in practice. The first, *character-based framing*, uses special communication control characters for idle fill and to indicate the beginning and ending of frames. The second, *bit-oriented framing with flags*, uses a special string of bits called a flag both for idle fill and to indicate the beginning and ending of frames. The third, *length counts*, gives the frame length in a field of the header. The following three subsections explain these three techniques, and the third also gives a more fundamental view of the problem. These subsections, except for a few comments, ignore the possibility of errors on the bit pipe. Section 2.5.4 then treats the joint problems of ARQ and framing in the presence of errors. Finally, Section 2.5.5 explains the trade-offs involved in the choice of frame length.

2.5.1 Character-Based Framing

Character codes such as ASCII generally provide binary representations not only for keyboard characters and terminal control characters, but also for various communication control characters. In ASCII, all these binary representations are seven bits long, usually with an extra parity bit which might or might not be stripped off in communication [since a cyclic redundancy check (CRC) can be used more effectively to detect errors in frames].

SYN (synchronous idle) is one of these communication control characters; a string of SYN characters provides idle fill between frames when a sending DLC has no data to send but a synchronous modem requires bits. SYN can also be used within frames, sometimes for synchronization of older modems, and sometimes to bridge delays in supplying data characters. STX (start of text) and ETX (end of text) are two other communication control characters used to indicate the beginning and end of a frame, as shown in Fig. 2.34.

The character-oriented communication protocols used in practice, such as the IBM binary synchronous communication system (known as Bisynch or BSC), are far more complex than this, but our purpose here is simply to illustrate that framing presents no insurmountable problems. There is a slight problem in the example above in that either the header or the CRC might, through chance, contain a communication control character. Since these always appear in known positions after STX or ETX, this causes no problem for the receiver. If the packet to be transmitted is an arbitrary binary string, however, rather than a string of ASCII keyboard characters, serious problems arise; the packet might contain the ETX character, for example, which could be interpreted as ending the frame. Character-oriented protocols use a special mode of transmission, called *transparent mode*, to send such data.

The transparent mode uses a special control character called DLE (data link escape). A DLE character is inserted before the STX character to indicate the start of a frame in transparent mode. It is also inserted before intentional uses of communication control characters within such a frame. The DLE is not inserted before the possible appearances of these characters as part of the binary data. There is still a problem if the DLE character itself appears in the data, and this is solved by inserting an extra DLE before each appearance of DLE in the data proper. The receiving DLC then strips off one DLE from each arriving pair of DLEs, and interprets each STX or ETX preceded by an unpaired DLE as an actual start or stop of a frame. Thus, for example, DLE ETX (preceded by something other than DLE) would be interpreted as an end of frame, whereas DLE DLE ETX (preceded by something other than DLE) would be interpreted as an appearance of the bits corresponding to DLE ETX within the binary data.

With this type of protocol, the frame structure would appear as shown in Fig. 2.35. This frame structure is used in the ARPANET. It has two disadvantages, the first of which is the somewhat excessive use of framing overhead (*i.e.*, DLE STX precedes each frame, DLE ETX follows each frame, and two SYN characters separate each frame for a total of six framing characters per frame). The second disadvantage is that each frame must consist of an integral number of characters.

Let us briefly consider what happens in this protocol in the presence of errors. The CRC checks the header and packet of a frame, and thus will normally detect errors

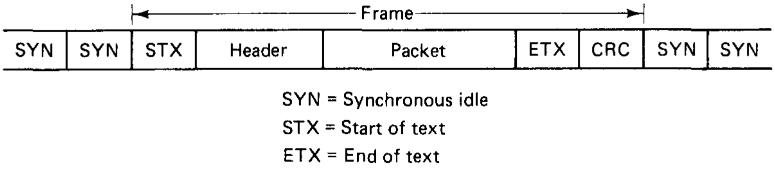


Figure 2.34 Simplified frame structure with character-based framing.

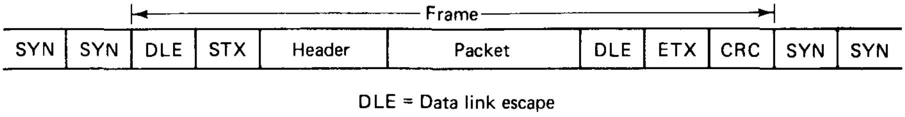


Figure 2.35 Character-based framing in a transparent mode as used in the ARPANET.

there. If an error occurs in the DLE ETX ending a frame, however, the receiver will not detect the end of the frame and thus will not check the CRC; this is why in the preceding section we assumed that frames could get lost. A similar problem is that errors can cause the appearance of DLE ETX in the data itself; the receiver would interpret this as the end of the frame and interpret the following bits as a CRC. Thus, we have an essentially random set of bits interpreted as a CRC, and the preceding data will be accepted as a packet with probability 2^{-L}, where L is the length of the CRC. The same problems occur in the bit-oriented framing protocols to be studied next and are discussed in greater detail in Section 2.5.4.

2.5.2 Bit-Oriented Framing: Flags

In the transparent mode for character-based framing, the special character pair DLE ETX indicated the end of a frame and was avoided within the frame by doubling each DLE character. Here we look at another approach, using a *flag* at the end of the frame. A flag is simply a known bit string, such as DLE ETX, that indicates the end of a frame. Similar to the technique of doubling DLEs, a technique called *bit stuffing* is used to avoid confusion between possible appearances of the flag as a bit string within the frame and the actual flag indicating the end of the frame. One important difference between bit-oriented and character based framing is that a bit-oriented frame can have any length (subject to some minimum and maximum) rather than being restricted to an integral number of characters. Thus, we must guard against appearances of the flag bit pattern starting in any bit position rather than just on character boundaries.

In practice, the flag is the bit string 01^60, where the notation 1^j means a string of j 1's. The rule used for bit stuffing is to insert (stuff) a 0 into the data string of the frame proper after each successive appearance of five 1's (see Fig. 2.36). Thus, the frame, after stuffing, never contains more than five consecutive 1's, and the flag at the end of the frame is uniquely recognizable. At the receiving DLC, the first 0 after each string of five consecutive 1's is deleted; if, instead, a string of five 1's is followed by a 1, the frame is declared to be finished.

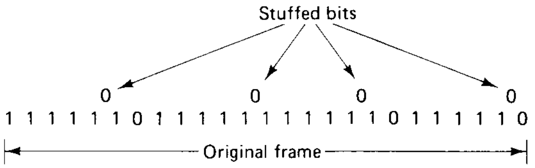


Figure 2.36 Bit stuffing. A 0 is stuffed after each consecutive five 1's in the original frame. A flag, 01111110, without stuffing, is sent at the end of the frame.

Bit stuffing has a number of purposes beyond eliminating flags within the frame. Standard DLCs have an abort capability in which a frame can be aborted by sending seven or more 1's in a row; in addition, a link is regarded as idle if 15 or more 1's in a row are received. What this means is that 01^6 is really the string denoting the end of a frame. If 01^6 is followed by a 0, it is the flag, indicating normal frame termination; if followed by a 1, it indicates abnormal termination. Bit stuffing is best viewed as preventing the appearance of 01^6 within the frame. One other minor purpose of bit stuffing is to break up long strings of 1's, which cause some older modems to lose synchronization.

It is easy to see that the bit stuffing rule above avoids the appearance of 01^6 within the frame, but it is less clear that so much bit stuffing is necessary. For example, consider the first stuffed bit in Fig. 2.36. Since the frame starts with six 1's (following a distinguishable flag), this could not be logically mistaken for 01^6. Thus, stuffing is not logically necessary after five 1's at the beginning of the frame (provided that the receiver's rule for deleting stuffed bits is changed accordingly).

The second stuffed bit in the figure is clearly necessary to avoid the appearance of 01^6. From a strictly logical standpoint, the third stuffed bit could be eliminated (except for the synchronization problem in older modems). Problem 2.31 shows how the receiver rule could be appropriately modified. Note that the reduction in overhead, by modifying the stuffing rules as above, is almost negligible; the purpose here is to understand the rules rather than to suggest changes.

The fourth stuffed bit in the figure is definitely required, although the reason is somewhat subtle. The original string 01^50 surrounding this stuffed bit could not be misinterpreted as 01^6, but the receiving DLC needs a rule to eliminate stuffed bits; it cannot distinguish a stuffed 0 following 01^5 from a data 0 following 01^5. Problem 2.32 develops this argument in detail.

There is nothing particularly magical about the string 01^6 as the bit string used to signal the termination of a frame (except for its use in preventing long strings of 1's on the link), and in fact, any bit string could be used with bit stuffing after the next-to-last bit of the string (see Problem 2.33). Such strings, with bit stuffing, are often useful in data communication to signal the occurrence of some rare event.

Consider the overhead incurred by using a flag to indicate the end of a frame. Assume that a frame (before bit stuffing and flag addition) consists of independent, identically distributed, random binary variables, with equal probability of 0 or 1. Assume for increased generality that the terminating signal for a frame is 01^j for some j (with 01^j0 being the flag and 01^{j+1} indicating abnormal termination); thus, $j = 6$ for the standard flag. An insertion will occur at (*i.e.*, immediately following) the i^{th} bit of the original frame (for $i \geq j$) if the string from $i - j + 1$ to i is 01^{j-1}; the probability of this is 2^{-j}. An insertion will also occur (for $i \geq 2j - 1$) if the string from $i - 2j + 2$ to i is 01^{2j-2}; the probability of this is 2^{-2j+1}. We ignore this term and the probability of insertions due to yet longer strings of 1's: first, because these probabilities are practically negligible, and second, because these insertions are used primarily to avoid long strings of 1's rather than to provide framing. Bit $j - 1$ in the frame is somewhat different than the other bits, since an insertion here occurs with probability 2^{-j+1} (*i.e.*, if the first $j - 1$ bits of the frame are all 1's).

Recall that the expected value of a sum of random variables is equal to the sum of the expected values (whether or not the variables are independent). Thus, the expected number of insertions in a frame of original length K is the sum, over i, of the expected number of insertions at each bit i of the frame. The expected number of insertions at a given bit, however, is just the probability of insertion there. Thus, the expected number of insertions in a string of length $K \geq j - 1$ is

$$(K - j + 3)2^{-j}$$

Taking the expected value of this over frame lengths K (with the assumption that all frames are longer than $j - 1$) and adding the $j + 1$ bits in the termination string, the expected overhead for framing becomes

$$E\{OV\} = (E\{K\} - j + 3)2^{-j} + j + 1 \tag{2.33}$$

Since $E\{K\}$ is typically very much larger than j, we have the approximation and upper bound (for $j \geq 3$)

$$E\{OV\} \leq E\{K\}2^{-j} + j + 1 \tag{2.34}$$

One additional bit is needed to distinguish a normal from an abnormal end of frame.

It will be interesting to find the integer value of j that minimizes this expression for a given value of expected frame length. As j increases from 1, the quantity on the right-hand side of Eq. (2.34) first decreases and then increases. Thus, the minimizing j is the smallest integer j for which the right-hand side is less than the same quantity with j increased by 1, that is, the smallest j for which

$$E\{K\}2^{-j} + j + 1 < E\{K\}2^{-j-1} + j + 2 \tag{2.35}$$

This inequality simplifies to $E\{K\}2^{-j-1} < 1$, and the smallest j that satisfies this is

$$j = \lfloor \log_2 E\{K\} \rfloor \tag{2.36}$$

where $\lfloor x \rfloor$ means the integer part of x. It is shown in Problem 2.34 that for this optimal value of j,

$$E\{OV\} \leq \log_2 E\{K\} + 2 \tag{2.37}$$

For example, with an expected frame length of 1000 bits, the optimal j is 9 and the expected framing overhead is less than 12 bits. For the standard flag, with $j = 6$, the expected overhead is about 23 bits (hardly cause for wanting to change the standard).

2.5.3 Length Fields

The basic problem in framing is to inform the receiving DLC where each idle fill string ends and where each frame ends. In principle, the problem of determining the end of an idle fill string is trivial; idle fill is represented by some fixed string (*e.g.*, repeated SYN characters or repeated flags) and idle fill stops whenever this fixed pattern is broken. In principle, one bit inverted from the pattern is sufficient, although in practice, idle fill is usually stopped at a boundary between flags or SYN characters.

Since a frame consists of an arbitrary and unknown bit string, it is somewhat harder to indicate where it ends. A simple alternative to flags or special characters is to include a length field in the frame header. DECNET, for example, uses this framing technique. Assuming no transmission errors, the receiving DLC simply reads this length in the header and then knows where the frame ends. If the length is represented by ordinary binary numbers, the number of bits in the length field has to be at least $\lfloor \log_2 K_{\max} \rfloor + 1$, where $K_{\max}$ is the maximum frame size. This is the overhead required for framing in this technique; comparing it with Eq. (2.37) for flags, we see that the two techniques require similar overhead.

Could any other method of encoding frame lengths require a smaller expected number of bits? This question is answered by information theory. Given any probability assignment $P(K)$ on frame lengths, the source coding theorem of information theory states that the minimum expected number of bits that can encode such a length is at least the entropy of that distribution, given by

$$H = \sum_K P(K) \log_2 \frac{1}{P(K)} \tag{2.38}$$

According to the theorem, at least this many bits of framing overhead, on the average, must be sent over the link per frame for the receiver to know where each frame ends. If $P(K) = 1/K_{\max}$, for $1 \leq K \leq K_{\max}$, then H is easily calculated to be $\log_2 K_{\max}$. Similarly, for a geometric distribution on lengths, with given $E\{K\}$, the entropy of the length distribution is approximately $\log_2 E\{K\} + \log_2 e$, for large $E\{K\}$. This is about 1/2 bit below the expression in Eq. (2.37). Thus, for the geometric distribution on lengths, the overhead using flags for framing is essentially minimum. The geometric distribution has an interesting extremal property; it can be shown to have the largest entropy of any probability distribution over the positive integers with given $E\{K\}$ (*i.e.*, it requires more bits than any other distribution).

The general idea of source coding is to map the more likely values of K into short bit strings and less likely values into long bit strings; more precisely, one would like to map a given K into about $\log_2[1/P(K)]$ bits. If one does this for a geometric distribution, one gets an interesting encoding known as the unary–binary encoding. In particular, for a given j, the frame length K is represented as

$$K = i2^j + r; \quad 0 \leq r < 2^j \tag{2.39}$$

The encoding for K is then i 0's followed by a 1 (this is known as a unary encoding of i) followed by the ordinary binary encoding of r (using j bits). For example, if $j = 2$ and $K = 7$, K is represented by $i = 1$, $r = 3$, which encodes into 0111 (where 01 is the unary encoding of $i = 1$ and 11 is the binary encoding of $r = 3$). Note that different values of K are encoded into different numbers of bits, but the end of the encoding can always be recognized as occurring j bits after the first 1.

In general, with this encoding, a given K maps into a bit string of length $\lfloor K/2^j \rfloor + 1 + j$. If the integer value above is neglected and the expected value over K is taken, then

$$E\{OV\} = E\{K\}2^{-j} + 1 + j \tag{2.40}$$

Note that this is the same as the flag overhead in Eq. (2.34). This is again minimized by choosing $j = \lfloor \log_2 E\{K\} \rfloor$. Thus, this unary–binary length encoding and flag framing both require essentially the minimum possible framing overhead for the geometric distribution, and no more overhead for any other distribution of given $E\{K\}$.

2.5.4 Framing with Errors

Several peculiar problems arise when errors corrupt the framing information on the communication link. First, consider the flag technique. If an error occurs in the flag at the end of a frame, the receiver will not detect the end of frame and will not check the cyclic redundancy check (CRC). In this case, when the next flag is detected, the receiver assumes the CRC to be in the position preceding the flag. This perceived CRC might be the actual CRC for the following frame, but the receiver interprets it as checking what was transmitted as two frames. Alternatively, if some idle fill follows the frame in which the flag was lost, the perceived CRC could include the error-corrupted flag. In any case, the perceived CRC is essentially a random bit string in relation to the perceived preceding frame, and the receiver fails to detect the errors with a probability essentially 2^{-L}, where L is the length of the CRC.

An alternative scenario is for an error within the frame to change a bit string into the flag, as shown for the flag 01^60:

$$0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ \ldots \qquad \text{(sent)}$$

$$0\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ \ldots \qquad \text{(received)}$$

It is shown in Problem 2.35 that the probability of this happening somewhere in a frame of K independent equiprobable binary digits is approximately $(1/32)Kp$, where p is the probability of a bit error. In this scenario, as before, the bits before the perceived flag are interpreted by the receiver as a CRC, and the probability of accepting a false frame, given this occurrence, is 2^{-L}. This problem is often called the data sensitivity problem of DLC, since even though the CRC is capable of detecting any combination of three or fewer errors, a single error that creates or destroys a flag, plus a special combination of data bits to satisfy the perceived preceding CRC, causes an undetectable error.

If a length field in the header provides framing, an error in this length field again causes the receiver to look for the CRC in the wrong place, and again an incorrect frame is accepted with probability about 2^{-L}. The probability of such an error is smaller using a length count than using a flag (since errors can create false flags within the frame); however, after an error occurs in a length field, the receiver does not know where to look for any subsequent frames. Thus, if a length field is used for framing, some synchronizing string must be used at the beginning of a frame whenever the sending DLC goes back to retransmit. (Alternatively, synchronization could be used at the start of every frame, but this would make the length field almost redundant.)

There are several partial solutions to these problems, but none are without disadvantages. DECNET uses a fixed-length header for each frame and places the length of the frame in that header; in addition, the header has its own CRC. Thus, if an error occurs in the length field of the header, the receiver can detect it by the header CRC, which is in a known position. One difficulty with this strategy is that the transmitter must still resynchronize after such an error, since even though the error is detected, the receiver will not know when the next frame starts. The other difficulty is that two CRCs must be used in place of one, which is somewhat inefficient.

A similar approach is to put the length field of one frame into the trailer of the preceding frame. This avoids the inefficiency of the DECNET approach, but still requires a special synchronizing sequence after each detected error. This also requires a special header frame to be sent whenever the length of the next frame is unknown when a given frame is transmitted.

Another approach, for any framing technique, is to use a longer CRC. This at least reduces the probability of falsely accepting a frame if framing errors occur. It appears that this is the most likely alternative to be adopted in practice; a standard 32 bit CRC exists as an option in standard DLCs.

A final approach is to regard framing as being at a higher layer than ARQ. In such a system, packets would be separated by flags, and the resulting sequence of packets and flags would be divided into fixed-length frames. Thus, frame boundaries and packet boundaries would bear no relation. If a packet ended in the middle of a frame and no further packets were available, the frame would be completed with idle fill. These frames would then enter the ARQ system, and because of the fixed-length frames, the CRC would always be in a known place. One disadvantage of this strategy is delay; a packet could not be accepted until the entire frame containing the end of the packet was accepted. This extra delay would occur on each link of the packet's path.

2.5.5 Maximum Frame Size

The choice of a maximum frame length, or maximum packet length, in a data network depends on many factors. We first discuss these factors for networks with variable packet lengths and then discuss some additional factors for networks using fixed packet lengths. Most existing packet networks use variable packet lengths, but the planners of broadband ISDN are attempting to standardize on a form of packet switching called asynchronous transfer mode (ATM) which uses very short frames (called cells in ATM) with a fixed length of 53 bytes. The essential reason for the fixed-length is to simplify the hardware for high-speed switching. There are also some advantages of fixed-length frames for multiaccess systems, as discussed in Chapter 4.

Variable frame length Assume that each frame contains a fixed number V of overhead bits, including frame header and trailer, and let K_{max} denote the maximum length of a packet. Assume, for the time being, that each message is broken up into as many maximum-length packets as possible, with the last packet containing what is left over. That is, a message of length M would be broken into $\lceil M/K_{max} \rceil$ packets, where $\lceil x \rceil$ is the smallest integer greater than or equal to x. The first $\lceil M/K_{max} \rceil - 1$ of these packets each contain K_{max} bits and the final packet contains between 1 and K_{max} bits.

The total number of bits in the resulting frames is then

$$\text{total bits} = M + \left\lceil \frac{M}{K_{max}} \right\rceil V \tag{2.41}$$

We see from this that as K_{max} decreases, the number of frames increases and thus the total overhead in the message, $\lceil M/K_{max} \rceil V$, increases. In the limit of very long messages, a fraction $V/(V + K_{max})$ of the transmitted bits are overhead bits. For shorter messages, the fraction of overhead bits is typically somewhat larger because of the reduced length of the final packet.

A closely related factor is that the nodes and external sites must do a certain amount of processing on a frame basis; as the maximum frame length decreases, the number of frames, and thus this processing load, increase. With the enormous increase in data rates available from optical fiber, it will become increasingly difficult to carry out this processing for small frame lengths. In summary, transmission and processing overhead both argue for a large maximum frame size.

We next discuss the many factors that argue for small frame size. The first of these other factors is the pipelining effect illustrated in Fig. 2.37. Assume that a packet must be completely received over one link before starting transmission over the next. If an entire message is sent as one packet, the delay in the network is the sum of the message transmission times over each link. If the message is broken into several packets, however, the earlier packets may proceed along the path while the later packets are still being transmitted on the first link, thus reducing overall message delay.

Since delay could be reduced considerably by starting the transmission of a packet on one link before receiving it completely on the preceding link, we should understand why this is not customarily done. First, if the DLC is using some form of ARQ, the CRC must be checked before releasing the packet to the next link. This same argument holds even if a CRC is used to discard packets in error rather than to retransmit them. Finally, if the links on a path have different data rates, the timing required to supply incoming bits to the outgoing link becomes very awkward; the interface between the DLC layer and the network layer also becomes timing dependent.

Let us investigate the combined effect of overhead and pipelining on message delay, assuming that each packet is completely received over one link before starting transmission on the next. Suppose that a message of length M is broken into maximum-length packets, with a final packet of typically shorter length. Suppose that the message must be transmitted over j equal-capacity links and that the network is lightly loaded, so that waiting at the nodes for other traffic can be ignored. Also, ignore errors on the links (which will be discussed later), and ignore propagation delays (which are independent of maximum packet length). The total time T required to transmit the message to the destination is the time it takes the first packet to travel over the first $j - 1$ links, plus the time it takes the entire message to travel over the final link (*i.e.*, when a nonfinal frame finishes traversing a link, the next frame is always ready to start traversing the link). Let C be the capacity of each link in bits per second, so that TC is the number of bit transmission times required for message delivery. Then, assuming that $M \geq K_{max}$,

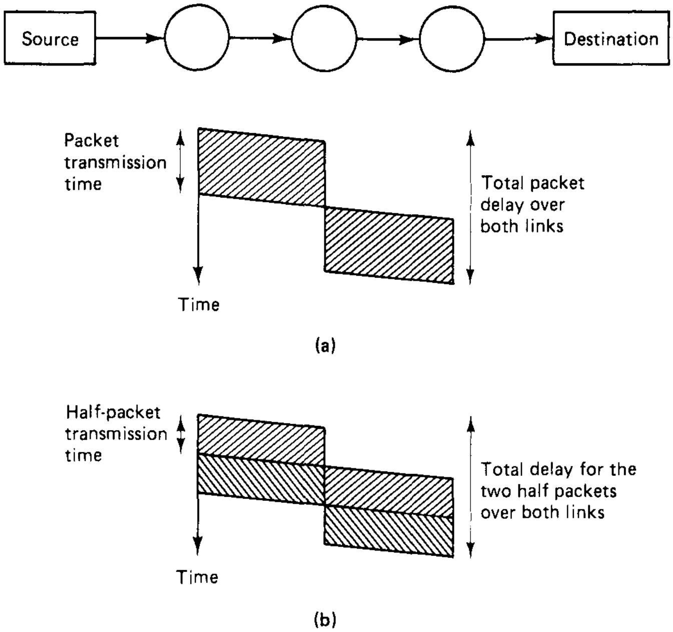

Source
Destination
Packet transmission time
Total packet delay over both links
Time
(a)
Half-packet transmission time
Total delay for the two half packets over both links
Time
(b)

Figure 2.37 Decreasing delay by shortening packets to take advantage of pipelining. (a) The total packet delay over two empty links equals twice the packet transmission time on a link plus the overall propagation delay. (b) When each packet is split in two, a pipelining effect occurs. The total delay for the two half packets equals 1.5 times the original packet transmission time on a link plus the overall propagation delay.

$$TC = (K_{max} + V)(j - 1) + M + \left\lceil \frac{M}{K_{max}} \right\rceil V \tag{2.42}$$

In order to find the expected value of this over message lengths M, we make the approximation that $E\{\lceil M/K_{max}\rceil\} = E\{M/K_{max}\} + \frac{1}{2}$ (this is reasonable if the distribution of M is reasonably uniform over spans of K_{max} bits). Then

$$E\{TC\} \approx (K_{max} + V)(j - 1) + E\{M\} + \frac{E\{M\}V}{K_{max}} + \frac{V}{2} \tag{2.43}$$

We can differentiate this with respect to K_{max} (ignoring the integer constraint) to find the value of K_{max} that minimizes $E\{TC\}$. The result is

$$K_{max} \approx \sqrt{\frac{E\{M\}V}{j-1}} \tag{2.44}$$

This shows the trade-off between overhead and pipelining. As the overhead V increases, K_{max} should be increased, and as the path length j increases, K_{max} should be reduced. As a practical detail, recall that delay is often less important for file transfers than for other messages, so file transfers should probably be weighted less than other messages in the estimation of $E\{M\}$, thus arguing for a somewhat smaller K_{max} than otherwise.

As the loading in a network increases, the pipelining effect remains, although packets typically will have to queue up at the nodes before being forwarded. The effect of overhead becomes more important at heavier loads, however, because of the increased number of bits that must be transmitted with small packet sizes. On the other hand, there are several other effects at heavier loads that argue for small packet sizes. One is the "slow truck" effect. If many packets of widely varying lengths are traveling over the same path, the short packets will pile up behind the long packets because of the high transmission delay of the long packets on each link; this is analogous to the pileup of cars behind a slow truck on a single-lane road. This effect is analyzed for a single link in Section 3.5. The effect is more pronounced over a path, but is mathematically intractable.

Delay for stream-type traffic (such as voice) is quite different from delay for data messages. For stream-type traffic, one is interested in the delay from when a given bit enters the network until that bit leaves, whereas for message traffic, one is interested in the delay from arrival of the message to delivery of the complete message. Consider the case of light loading again and assume an arrival rate of R and a packet length K. The first bit in a packet is then held up for a time K/R waiting for the packet to be assembled. Assuming that the links along the path have capacities $C_1, C_2, \ldots$, each exceeding R, and assuming V bits of framing overhead, a given packet is delayed by $(K+V)/C_i$ on the ith link. When a given packet is completely received at the last node of the network, the first bit of the packet can be released immediately, yielding a total delay

$$T = \frac{K}{R} + (K+V)\sum_i \frac{1}{C_i} \tag{2.45}$$

Assuming that the received data stream is played out at rate R, all received bits have the same delay, which is thus given by Eq. (2.45). We have tacitly assumed in deriving this equation that $(K+V)/C_i \leq K/R_i$ for each link i (*i.e.*, that each link can transmit frames as fast as they are generated). If this is violated, the queueing delay becomes infinite, even with no other traffic in the network. We see then from Eq. (2.45) that T decreases as K decreases until $(K+V)/C_i = K/R$ for some link, and this yields the minimum possible delay. Packet lengths for stream traffic are usually chosen much larger than this minimum because of the other traffic that is expected on the links. As link speeds increase, however, the dominant delay term in Eq. (2.45) is K/R, which is unaffected by other traffic. For 64 kbps voice traffic, for example, packets usually contain on the order of 500 bits or less, since the delay from the K/R term starts to become objectionable for longer lengths.

Note that under light loading, the delay of a session (either message or stream) is controlled by the packet length of that session. Under heavy-loading conditions, however, the use of long packets by some users generally increases delay for all users. Thus a maximum packet length should be set by the network rather than left to the users.

Several effects of high variability in frame lengths on go back n ARQ systems were discussed in Section 2.4.2. High variability either increases the number of packets that must be retransmitted or increases waiting time. This again argues for small maximum packet size. Finally, there is the effect of transmission errors. Large frames have a somewhat higher probability of error than small frames (although since errors are usually correlated, this effect is not as pronounced as one would think). For most links in use (with the notable exception of radio links), the probability of error on reasonable-sized frames is on the order of 10^{-4} or less, so that this effect is typically less important than the other effects discussed. Unfortunately, there are many analyses of optimal maximum frame length in the literature that focus only on this effect. Thus, these analyses are relevant only in those special cases where error probabilities are very high.

In practice, typical maximum frame lengths for wide area networks are on the order of 1 to a few thousand bits. Local area networks usually have much longer maximum frame lengths, since the path usually consists of a single multiaccess link. Also, delay and congestion are typically less important there and long frames allow most messages to be sent as a single packet.

Fixed frame length When all frames (and thus all packets) are required to be the same length, message lengths are not necessarily an integer multiple of the packet length, and the last packet of a message will have to contain some extra bits, called fill, to bring it up to the required length. Distinguishing fill from data at the end of a packet is conceptually the same problem as determining the end of a frame for variable-length frames. One effect of the fill in the last packet is an additional loss of efficiency, especially if the fixed packet length is much longer than many messages. Problem 2.39 uses this effect to repeat the optimization of Eq. (2.44), and as expected, the resulting optimal packet length is somewhat smaller with fixed frame lengths than with variable frame lengths. A considerably more important practical effect comes from the need to achieve a small delay for stream-type traffic. As was pointed out earlier, 64 kbps voice traffic must use packets on the order or 500 bits or less, and this requirement, for fixed frame length, forces *all* packets to have such short lengths. This is the primary reason why ATM (see Section 2.10) uses 53 byte frames even though very much longer frames would be desirable for most of the other traffic.

2.6 STANDARD DLCs

There are a number of similar standards for data link control, namely HDLC, ADCCP, LAPB, and SDLC. These standards (like most standards in the network field) are universally known by their acronyms, and the words for which the acronyms stand are virtually as cryptic as the acronyms themselves. HDLC was developed by the International Standards Organization (ISO), ADCCP by the American National Standards Institute (ANSI), LAPB by the International Consultative Committee on Telegraphy and Telephony (CCITT), and SDLC by IBM. HDLC and ADCCP are virtually identical and are described here. They have a wide variety of different options and modes of operation. LAPB is the DLC layer for X.25, which is the primary standard for connecting an external site to a subnet; LAPB is, in essence, a restricted subset of the HDLC options and modes, as will be described later. Similarly, SDLC, which was the precursor of HDLC and ADCCP, is essentially another subset of options and modes.

HDLC and ADCCP are designed for a variety of link types, including both multiaccess or point-to-point links and both full-duplex links (*i.e.*, both stations can transmit at once) or half-duplex links (*i.e.*, only one station can transmit at a time). There are three possible *modes* of operation, one of which is selected when the link is initialized for use.

The first mode, the *normal response mode* (NRM), is for a master–slave type of link use such as is typical between a computer and one or more peripherals (the word "normal" is of historical significance only). There is a primary or master station (the computer) and one or more secondary or slave stations. The secondary stations can send frames only in response to polling commands from the primary station. This mode is reasonable in multiaccess situations where the polling commands ensure that only one station transmits at a time (see Chapter 4 for a general discussion of multiaccess communication). NRM is also reasonable for half-duplex links or for communication with peripherals incapable of acting in other modes.

The second mode, the *asynchronous response mode* (ARM), is also a master–slave mode in which the secondary nodes are not so tightly restricted. It is not widely used and will not be discussed further.

The third mode, the *asynchronous balanced mode* (ABM), is for full-duplex point-to-point links between stations that take equal responsibility for link control. This is the mode of greatest interest for point-to-point links in data networks. It is also used, in slightly modified form, for multiaccess communication in local area networks. LAPB uses only this mode, whereas SDLC uses only the first two modes.

We first describe the frame structure common to all three modes and all four standards. Then the operation of the normal response mode (NRM) and the asynchronous balanced mode (ABM) are described in more detail. Although ABM is our major interest, the peculiarities of ABM will be understood more easily once NRM is understood.

Figure 2.38 shows the frame structure. The flag, $01^6 0$, is as described in Section 2.5.2, with bit stuffing used within the frame (the flag is not considered as part of the frame proper). One or more flags must separate each pair of frames. The CRC uses the CRC-CCITT generator polynomial $g(D) = D^{16} + D^{12} + D^5 + 1$ as explained in Section 2.3.4; it checks the entire frame (not including the flag). One modification is that the first 16 bits of the frame are inverted in the calculation of the CRC (although not inverted in transmission). Also, the remainder is inverted to form the CRC. At the receiving DLC, the first 16 bits of the frame are similarly inverted for recalculation of the remainder, which is then inverted for comparison with the received CRC. One reason for this is to avoid having an all-zero string (which is common if the link or modem is malfunctioning) satisfy the CRC. Another reason is to avoid satisfying the CRC if a few

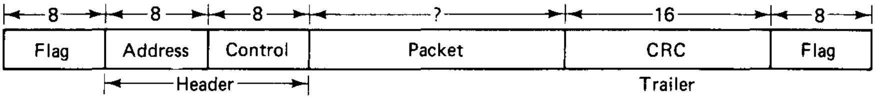


Figure 2.38 Frame structure of standard DLCs. The address, control, and CRC fields can each be optionally extended in length.

zeros are added or deleted from the beginning or end of the frame. As an option, the 32-bit CRC discussed in Section 2.3.4 can be used. This optional 32-bit CRC is used in the IEEE 802 standard for local area networks.

The address field normally consists of one byte (eight bits). For the normal response mode (NRM), the address is always that of the secondary station; that is, when the primary sends a frame to a secondary, it uses the secondary's address, and when the secondary responds, it uses its own address. Note that this address is not intended as a destination for a session, but simply to distinguish stations for multiaccess use. For point-to-point links, the address field has no natural function, although we shall see later that it has some rather peculiar uses in ABM. There is an option in which the address field is extendable to an arbitrary number of 1 or more bytes; the first bit in a byte is 1 if that byte is the last address byte, and otherwise it is 0.

The layout of the control field is shown in Fig. 2.39. There are three different frame formats: *information*, *supervisory*, and *unnumbered*. The information frames are the actual packet-carrying frames; they normally use go back n ARQ with a modulus of 8. As shown in the figure, the sequence and request numbers SN and RN are part of the control field. The supervisory frames are used to return ARQ information (*i.e.*, RN) either when there are no data packets to send or when a speedy ack or nak is needed. Finally, the unnumbered frames are used for the initiation and termination of a link and for sending various kinds of supplementary control information.

There is an option under which the control field is two bytes long and in which SN and RN are each seven bits long and each use a modulus of 128. Aside from this lengthening of SN and RN, this option is essentially the same as the normal single-byte control field, which is assumed in the following description.

The first bit of the control field distinguishes the information format from the supervisory and unnumbered formats. Similarly, for noninformation formats, the second bit distinguishes supervisory from unnumbered formats. The fifth bit of the control field is the *poll final* (P/F) bit. For the normal response mode, the primary sends a 1 in this bit position to poll the secondary. The secondary must then respond with one or more frames, and sends a 1 in this bit position to denote the final frame of the response. This polling and response function of the P/F bit in NRM is used for supervisory and unnumbered frames as well as information frames. The P/F bit is used in a rather strange way for the asynchronous balanced mode (ABM), as described later.

There are four types of supervisory frames: *receive-ready* (RR), *receive-not-ready* (RNR), *reject* (REJ), and *selective-reject* (SREJ). The type is encoded in the third and

<table>
<tr><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th></th></tr>
<tr><td>0</td><td colspan="3">SN</td><td>P/F</td><td colspan="3">RN</td><td>Information</td></tr>
<tr><td>1</td><td>0</td><td colspan="2">type</td><td>P/F</td><td colspan="3">RN</td><td>Supervisory</td></tr>
<tr><td>1</td><td>1</td><td colspan="2">type</td><td>P/F</td><td colspan="3">type</td><td>Unnumbered</td></tr>
</table>

Figure 2.39 Structure of the control field for the information, supervisory, and unnumbered frame formats.

fourth bits of the control field. All of these have the basic function of returning ARQ information (*i.e.*, all packets with numbers less than RN are acknowledged) and consequently they contain neither a sequence number nor an information field. RR is the normal response if there is no data packet on which to piggyback RN. RNR means in addition that the station is temporarily unable to accept further packets; this allows a rudimentary form of link flow control—that is, if the buffers at one end of the link are filling up with received packets, the other end of the link can be inhibited from further transmission.

The REJ supervisory frame is sent to indicate either a received frame in error or a received frame with other than the awaited sequence number. Its purpose is to improve the efficiency of go back n ARQ, as explained in Section 2.4.2. Similarly, SREJ provides a primitive form of selective repeat. It acks all packets before RN and requests retransmission of RN itself followed by new packets; one cannot provide supplementary information about multiple required retransmissions. This facility can be useful, however, on satellite links, where there is a very small probability of more than one frame error in a round-trip delay. The implementation of REJ and SREJ is optional, whereas RR and RNR are required.

The unnumbered frames carry no sequence numbers and are used for initializing the link for communication, disconnecting the link, and special control purposes. There are five bits in the control field available for distinguishing different types of unnumbered frames, but not all are used. Six of these types will be referred to generically as *set mode* commands. There is one such command for each mode (NRM, ARM, and ABM), and one for each mode using the extended control field. This command is sent by a primary (for NRM or ARM) and by either node (for ABM) to initialize communication on a link. Upon sending or receiving this command, a station sets its sequence and request numbers to zero. The recipient of the command must acknowledge it with an unnumbered frame, either an *unnumbered ack* which agrees to start using the link, or a *disconnected mode* which refuses to start using the link. Similarly, there is a disconnect command which is used to stop using the link; this also requires an unnumbered ack in response.

Figure 2.40 shows the operation of the NRM mode for a primary and two secondaries. Note that the primary initiates communication with each separately and disconnects from each separately. Note also that the P/F bit is set to 1 on each unnumbered frame that requires a response. After a given unnumbered frame with $P = 1$ is sent to a particular secondary, no new unnumbered frame with $P = 1$ may be sent to that secondary until the given frame is acknowledged by a frame with $F = 1$; naturally the given frame may be repeated after a time-out waiting for the ack. This alternation of unnumbered frames with acknowledgments is similar to the stop-and-wait protocol. Section 2.7 explains this similarity carefully and shows that the acknowledgment strategy here contains the same defect as stop-and-wait strategies without numbered acknowledgments.

In the asynchronous balanced mode, the decision was made, in designing these protocols, to have each station act as both a primary and a secondary (thus replacing a simple problem with a more familiar but more difficult problem). When a station sends a primary frame (in its role as a primary station), it uses the other station's address in the address field, and the P/F bit is interpreted as a poll bit. When a station sends

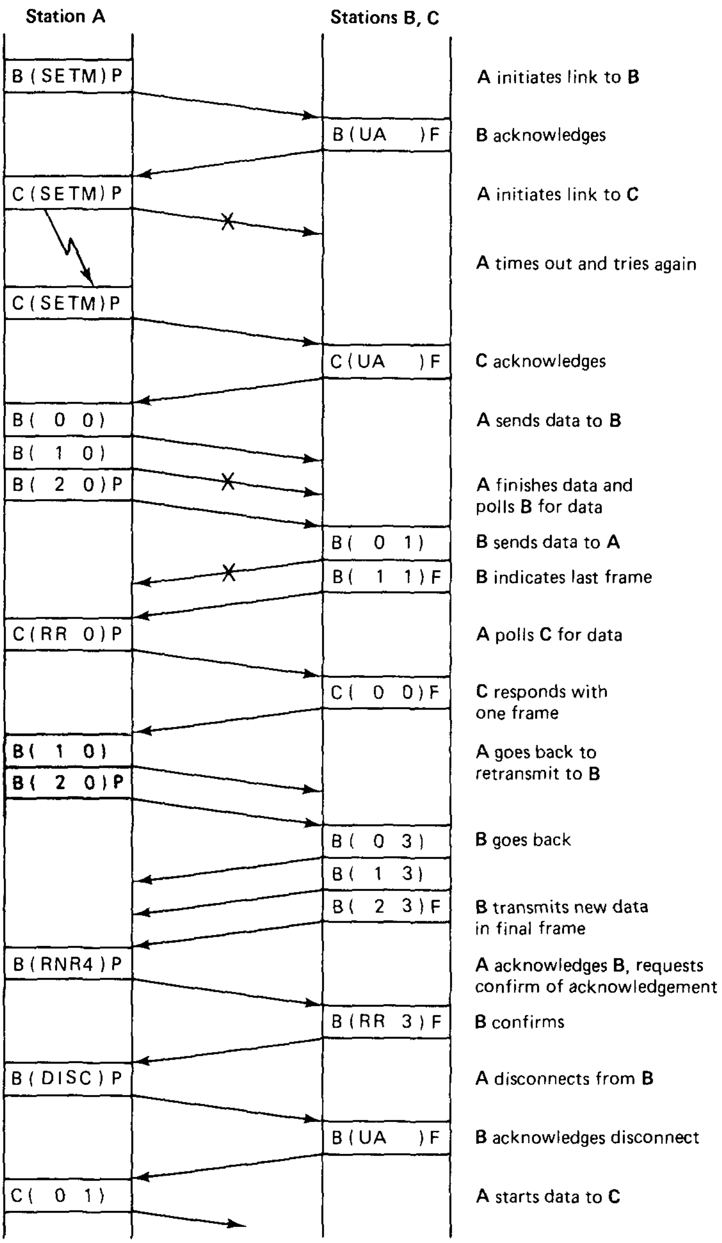


Figure 2.40 Normal response mode (NRM) operation of primary A with two secondaries, B and C. SETM refers to the command setting, the NRM mode with $SN = 0$ and $RN = 0$ at both stations. The address of each frame is given first, followed by (SN, RN) for information frames, (type,RN) for supervisory frames, and (type) for unnumbered frames; this is followed by a P/F bit if set to 1. Note that A uses different SN and RN values for B and C.

a secondary frame (in its role as a secondary), it uses its own address, and the P/F bit is interpreted as a final bit. In customary operation, all information frames and all unnumbered frames requiring acks are sent as primary frames; all unnumbered frames carrying acks or responses, and most supervisory frames (which ack primary data frames), are sent as secondary frames.

What this means in practice is that the P/F bit is either P or F, depending on the address field. This bit, in conjunction with the address field, has three possible values: $P = 1$, requiring a response from the other station acting in a secondary role; $F = 1$, providing the secondary response, and neither. Given the somewhat inadequate performance of this poll/final alternation, this modified use of the P/F bit was an unfortunate choice.

Figure 2.41 shows a typical operation of ABM. The most interesting part of this is the next-to-last frame sent by station A; this is a supervisory frame sent with $P = 1$, thus requiring an acknowledgment. Since A has signified unwillingness to accept any more packets, a response from B signifies that B knows that packets up to three inclusive have been accepted and that no more will be accepted. Thus, upon receiving the acknowledgment of this, A can disconnect, knowing that B knows exactly which packets from B have been accepted; A also knows which of its packets have been accepted, so the disconnect leaves no uncertainty about the final condition of any packets at either A

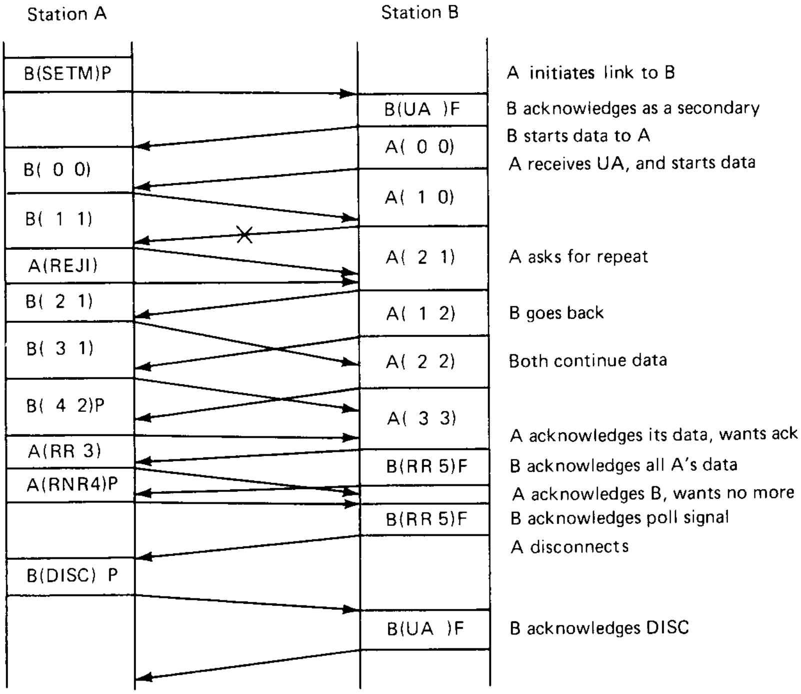


Figure 2.41 Asynchronous balanced mode (ABM) of operation. SETM refers to the set ABM command, which initiates the link with SN and RN at both sides equal to 0. Note the use of addresses in different types of frames.

or B. Note that this "ideal" disconnect requires some previous agreement (not specified by the standard) between A and B; in particular, A must send RNR, with $P = 1$ and get an ack before disconnecting—this lets B know that A did not go back to accepting packets again temporarily before the disconnect. This "ideal" disconnect does not work, of course, if the line is being disconnected because of a malfunction; in this case, the idea of an orderly sequence of RNR, ack, disconnect, and ack is unrealistic.

Another type of unnumbered frame is "frame reject." This is sent in response to a received frame with a valid CRC but otherwise impossible value (such as exceeding the maximum frame length, being shorter than the minimum frame length, having an RN value not between SN_{min} and $SN_{\text{max}} - 1$, or having an impossible control field). Thus, either an undetectable error has occurred or one of the stations has malfunctioned. Such an error is outside the realm of what the ARQ is designed for, and it should be reported at each station. The usual response to a "frame reject" is for the opposite station either to reinitialize the link with a set mode command or send an unnumbered "reset" command. The first resets RN and SN at each station to 0, and the second sets SN to 0 at the station that sent the faulty frame and RN to 0 at the receiving station. Both stations will again start to send packets (starting from the first packet not acked), but because of the resetting, some packets might arrive more than once and out of order. This is not a fault of the protocol, but a recognition that node malfunctions and undetectable errors cannot be dealt with.

2.7 INITIALIZATION AND DISCONNECT FOR ARQ PROTOCOLS

The discussion of ARQ in Section 2.4 assumed throughout that the protocol was correctly initialized with no frames traveling on the link and with sequence and request numbers equal to 0 at both ends of the link. Initialization is a general problem for protocols involving communication between two nodes, and arises, for example, in transport protocols and many special-purpose protocols. We shall soon see that initialization and disconnection are intimately related.

One might think that the problem of initializing an ARQ protocol is trivial—one might imagine simply initializing the DLC at each node when the link is first physically connected and then using the ARQ protocol forever. Unfortunately, if the link fails or if one or both nodes fail, initialization is more complicated. We first discuss the problem in the presence of link failures alone and then consider the case in which nodes can fail also.

2.7.1 Initialization in the Presence of Link Failures

It was shown in Section 2.4 that standard ARQ protocols operate correctly (assuming correct initialization) in the presence of arbitrary link delays and errors. Thus, even if the link fails for some period of time, the ARQ protocol could continue attempting packet transmission, and those packets not received correctly before the failure would be received after the failure. The difficulty with this point of view is that when a link fails for a protracted period of time, it is necessary for the transport and/or network layers to take over the problem of reliable end-to-end delivery, using alternative paths for those packets that were not delivered over the failed link. When the failed link eventually returns to operation, the higher layers set up new paths through the link, thus presenting the link with a completely new stream of packets.

We see then that the operation of a link protocol over time is properly viewed as being segmented into an alternation of up and down periods. The DLC must be properly initialized at the beginning of each up period, whereas at the end of an up period, there may be packets still in the process of communication that have been accepted into the DLC for transmission at one node but not yet received and released at the opposite node. Stated more mathematically, correct operation of a DLC means that in each up period, the string of packets released to the higher layer at one node must be a prefix of the string of packets accepted for transmission from the higher layer at the opposite node.

2.7.2 Master–Slave Protocol for Link Initialization

The careful reader will have observed that there is a problem in defining an up period carefully; in particular, the nodes at opposite ends of a link might not agree at any given instant about whether the link is up or down. Rather than resolving this issue in the abstract, we present a simple protocol for initializing and disconnecting the link. The protocol is a master–slave protocol where one node, say A, is in charge of determining when the link should be up or down. Node A then informs node B at the other end of the link whenever it determines that the link has changed from down to up or up to down.

The sequence of messages (for this protocol) that go from A to B is then simply an alternating sequence of initialize and disconnect messages (see Fig. 2.42). In this discussion we ignore the data messages that are sent during up periods and simply focus on the messages of the initialization protocol. We view these messages as using a stop-and-wait protocol, using modulo 2 sequence numbering. Thus, the initialize messages (denoted INIT) correspond to frames with $SN = 1$, and the disconnect messages (denoted DISC) correspond to frames with $SN = 0$. The acknowledgments sent by node B will be denoted ACKI and ACKD. In terms of stop-and-wait ARQ, ACKI corresponds to $RN = 0$ [*i.e.*, the INIT message ($SN = 1$) has been correctly received and DISC ($SN = 0$) can be sent when desired by A].

When node A sends INIT, it waits for the acknowledgment ACKI and continues to send INIT on successive time-outs until receiving ACKI. Node A is then initialized (as described below) and can send and receive data packets until it decides to disconnect, at which time it sends DISC (repeatedly if need be) until receiving ACKD. The decision to disconnect (after A is initialized and up) corresponds to the acceptance of a new packet from the higher layer in ordinary stop and wait. The decision to initialize (after A has received ACKD and is down) has the same interpretation. Note that these decisions to try to initialize or disconnect are not part of the initialization protocol, but rather should be viewed either as coming from a higher layer or as coming from measurements of the link. The decisions to actually send the messages INIT or DISC are part of the protocol, however, and rely on having already received an acknowledgment for the previous message.

From our analysis of the stop and wait protocol, we know that each message from A is accepted once and only once in order. There is also a strict alternation of events between A and B (*i.e.*, B accepts INIT, A accepts ACKI, B accepts DISC, A accepts ACKD, etc.). Now assume that A regards the link as up from the time that it receives ACKI from B until it sends DISC to B (see Fig. 2.42). Similarly, B regards the link as up from the time that it receives INIT from A until it receives DISC from A. Each node sends and accepts data packets only when it regards the link as up. Assuming that frames stay in order on the links, we see that the link from A to B must be free of data when a DISC message reaches B, and it must stay free until the next up period starts at A. Similarly, the link from B to A must be free of data when ACKD reaches A and must stay free until the next up period starts at B (see Fig. 2.43). It follows that the link is free of data in both directions from the time when an ACKD message reaches A until the next INIT message reaches B. Assuming that A initializes its ARQ protocol on sending the INIT message and that B initializes on receiving the INIT message, we see that the ARQ is correctly initialized.

A peculiar feature of this protocol is that once node A starts to initialize, there is no way to abort the initialization before receiving ACKI from node B (see Problem 2.40). Naturally, node A can start to disconnect as soon as ACKI is received, thus sending no data, but A must continue to send INIT until receiving ACKI. The protocol could be complicated to include a message to abort an attempted initialization, but the example described in Fig. 2.44 should make the reader wary of ad hoc additions to protocols. A similar issue is that node B might wish to refuse to initialize (which would be handled by the "disconnected mode" message in HDLC). The cleanest way to handle this is for node B to acknowledge A's INIT message, but to use an extra control field to request A to disconnect immediately without sending data.

The initialization protocol used in the normal response mode of HDLC (see Fig. 2.40) behaves in almost the same way as the initialization protocol above. The SETM unnumbered frame corresponds to INIT, the DISC unnumbered frame corresponds to

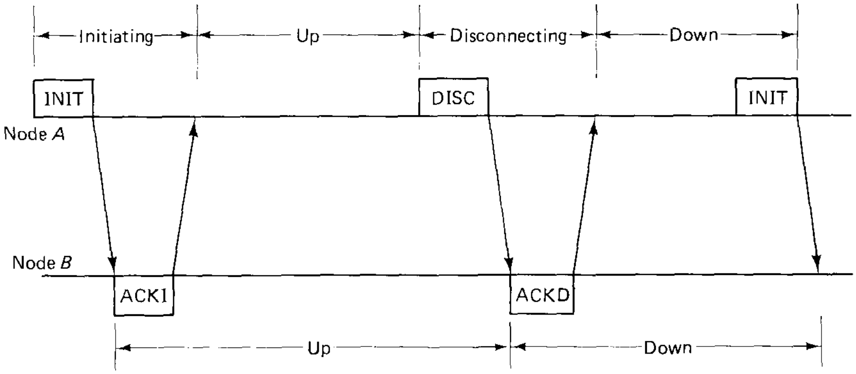


Figure 2.42 Initialization and disconnection protocol. Data are transmitted only in up periods and the ARQ is initialized at the beginning of an up period. Note that node A cycles between periods of initiating, being up, disconnecting, and being down.

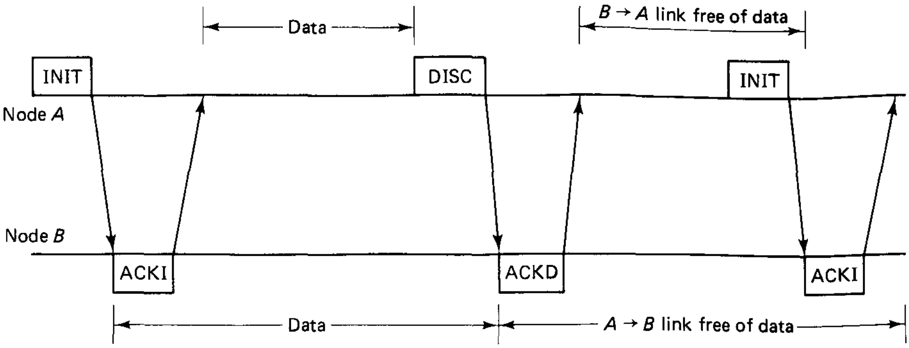


Figure 2.43 Illustration of periods when link is free of data in each direction. Note that the link is free in both directions both when A sends INIT and when B receives it.

DISC, and the ACK unnumbered frame corresponds to both ACKI and ACKD; that is, the unnumbered acks in HDLC fail to indicate whether they are acking initialize or disconnect frames. Figure 2.44 illustrates how this can cause incorrect operation. Node A times-out twice trying to initialize a link between A and B. Finally, on receiving an ack, it initializes the ARQ and sends data packet D with $SN = 0$. After timing out waiting for an acknowledgment, node A decides that the link is not operational, sends a disconnect, and the higher layers send D over some other path. An ack from one of the earlier SETM messages then arrives at A and is viewed as an ack to DISC, so A tries to initialize the link again. The third old ack then arrives, A sends a new data packet D', again initialized with $SN = 0$; D' is lost on the link, but an ack for the old packet D arrives; packet D' then never reaches node B and the ARQ system has failed.

This conceptual error in HDLC does not seem to have caused many problems over the years. The reason for this appears to be that in most networks, link delays are not arbitrary and are easily determined. Thus it is easy to set time-outs (for retransmission of unnumbered frames) large enough to avoid this type of failure. Even if the time-outs were set very poorly, it would be highly unlikely for a node to decide to disconnect and

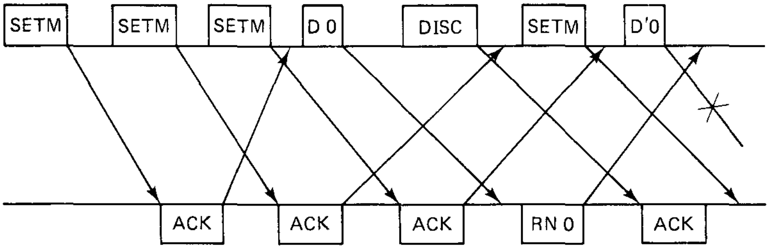


Figure 2.44 Example of HDLC in which node A attempts to initialize the link three times, due to delayed acknowledgments from B. The delayed acks are falsely interpreted as being acks for a subsequent disconnect and initialization. This in turn causes a failure in the data packets.

then to try to reinitialize over a time span comparable to the link delay. Despite the lack of practical problems caused by this error, however, this is a good example of the danger of designing protocols in an ad hoc fashion.

In the initialization and disconnect protocol just discussed, the process of disconnecting is a critical part of the protocol, and can be viewed as the first part of a new initialization of the link. In fact, if the link fails, the message DISC cannot be sent over the link until the link has recovered; it is this message, however, that guarantees that the next INIT message is not an earlier copy of an earlier attempt to initialize. As seen in Section 2.6, initialization and disconnection are often used on a multidrop line to set up and terminate communication with a given secondary node in the absence of any link failures, and in these applications, the DISC message is really a signal to disconnect rather than part of the initialization process.

Some readers may have noticed that the stop-and-wait protocol that we are using for initialization itself requires initialization. This is true, but this protocol does not require reinitialization on each link failure; it requires initialization only once when the link is physically set up between the two nodes. Recall that the ARQ protocol requires reinitialization on each link failure because of the need to break the packet stream between successive up periods on the link.

2.7.3 A Balanced Protocol for Link Initialization

It is often preferable to have an initialization protocol that is the same at both ends of the link. At the very least, there is a conceptual simplification in not having to decide which node is master and ensuring that each node uses the appropriate protocol. The essential idea of a balanced initialization protocol is to use two master–slave protocols, with node A playing the master for one and node B playing the master for the other. The only new requirement is to synchronize the two protocols so that they have corresponding up and down periods.

To simplify this synchronization somewhat, we assume that each INIT or DISC message from a node (acting as master) also contains a piggybacked ACKI or ACKD for the master–slave protocol in the opposite direction. Thus a single received message can affect both master–slave protocols, and we regard the ACK as being acted on first. ACK messages can also be sent as stand alone messages when there is no need to send an INIT or DISC but an INIT or DISC is received.

We recall that in the master–slave initialization protocol from A to B, the link was considered up at A from the receipt of ACKI until an external decision to disconnect (see Fig. 2.42). The link was regarded as up at B from the receipt of INIT until the receipt of the next DISC. For the balanced protocol, a node regards the link as *up* if it is up according to both the $A \rightarrow B$ protocol and the $B \rightarrow A$ protocol. A node initializes its ARQ at the beginning of an up period and sends and acknowledges data during this period. Similarly, a node regards the link as down if it is down according to both the $A \rightarrow B$ and $B \rightarrow A$ protocols.

If the link is regarded as up at node A (according to both master–slave protocols), either an external decision at A or a DISC message from B causes A to leave the up period and start to disconnect by sending a DISC message. As a special case of this, if a DISC message from B carries a piggybacked ACKI, the ACKI can cause node A to regard the link as up, in which case the DISC message immediately ends the up period (see Problem 2.40). Similarly, if the link is regarded as down at node A, either an external decision or an INIT message from B causes A to leave the down period and start to initialize by sending an INIT message. Again, if an INIT message from B is accompanied by a piggybacked ACKD, the ACKD can cause A to regard the link as down, and the INIT message immediately ends the down period.

Figure 2.45 illustrates the operation of this balanced protocol. Each of the master–slave protocols operate as described in the preceding section, and in particular, each node, in its role as slave, acks each INIT and DISC message from the other node; also each node, in its role as master, continues to retransmit INIT and DISC messages until acknowledged. The synchronizing rules of the preceding paragraph simply limit when external decisions can be made about starting to initialize and disconnect. For this reason, the safety of the balanced protocol follows immediately from the correctness of the master–slave protocols. The safety of the protocol can be verified, using the same assumptions as in Section 2.4, simply by verifying that when one node starts to initialize (or disconnect), the other node will eventually start to do the same (if it did not start earlier), and both nodes will eventually reach the up state (or the down state in the case of disconnect). See Problem 2.40 for details.

This protocol has the same peculiar problems as the master-slave protocol in that a node might be sending INIT messages with no intention of transferring data; this could occur either because of a link failure while in the process of initializing or because of sending an INIT message in response to the opposite node sending an INIT message. These problems are best treated by using extra fields for control information and recognizing that the protocol here is primarily designed for synchronizing the two nodes.

The initialization part of this balanced protocol (*i.e.*, sending INIT, waiting for both ACKI and INIT from the other side, and responding to INIT with ACKI) is often

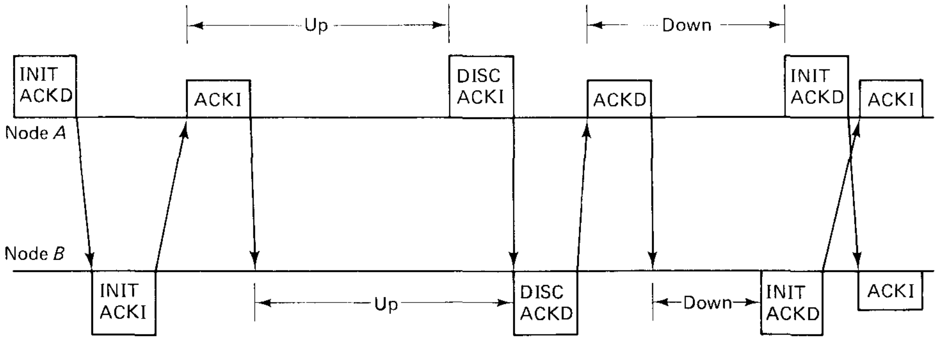


Figure 2.45 Example of balanced initialization and disconnect protocol. Either node can start to initialize from the down state or start to disconnect from the up state and the other node will follow.

called a three-way handshake. The reason for this terminology is that the ACKI from the second node is usually piggybacked on the INIT from the second node, making a three-message interchange. It is always possible that both sides start to initialize independently, however, and then all four messages must be sent separately (as seen on the right-hand side of Fig. 2.45). We shall see a closely related type of three-way handshake in transport protocols, but the problem is more complex there because of the possibility of packets traveling out of order.

2.7.4 Link Initialization in the Presence of Node Failures

From the viewpoint of a network, a node failure is similar to (but, as we shall see, not the same as) the failures of all the adjacent links to that node; in both cases, no communication is possible passing through that node or going to or from that node. For our purposes, a node is modeled as either working correctly or being in a failure state; in the failure state, no input is accepted, no output is produced, and no operations occur (except for a possible loss of state information). We view each node as alternating in time between correct operation and the failure state (typically, with very long periods between failures). Thus the question of interest here is that of reinitialization after a node failure. Another interesting class of problems, which we do not address here, is that of intermittent faulty operation, or even malicious operation, at a node. This is really a network layer problem and is treated in [Per88].

If a node maintains its state information during a failure period, we view the data at the node as being lost or retransmitted over other paths. Thus when the node returns to correct operation, the adjacent links must be reinitialized and the problem is really the same as if all the adjacent links had failed. If a node loses its state information during a failure period, the problem of reinitialization is much more difficult than that of adjacent link failures.

To illustrate the problem of initialization after loss of state information, consider the master–slave initialization protocol of Section 2.7.2 and suppose that node A, the master node, fails and recovers several times in rapid succession. Assume also that when A recovers, the initialization protocol starts in the down state (*i.e.*, in the state for which ACKD was the last received message); Problem 2.41 looks at the opposite case where it starts in the up state. Each time node A recovers, it receives a new sequence of packets to transmit to node B after initializing the (A, B) link.

Figure 2.46 illustrates what can happen if the time between failure and recovery is on the order of the round-trip delay. Node B has no way of knowing that A has failed (other than the messages of the initialization protocol, which are supposed to initialize the link after a failure). In the figure, A sends just the INIT message and then fails. When it recovers, it sends INIT again, and receives the ack from the previous INIT; this completes the initialization and allows A to send data packet D with sequence number 0. Node A then fails again, and on recovery gets the responses from B from the previous period when A was functioning.

The failure to initialize correctly here is not due to the particular protocol and is not due to the particular starting state. In fact, incorrect operation can occur for any

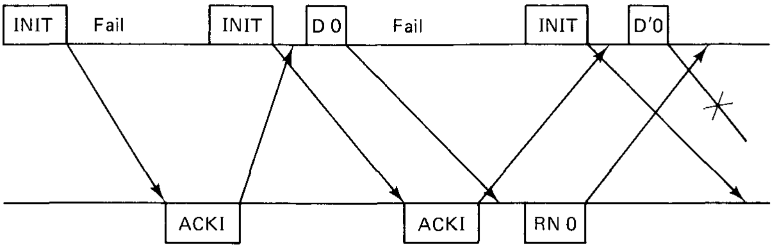


Figure 2.46 Example of a sequence of node failures, with loss of state, that causes a failure in correct initialization. Each time the node recovers from a failure, it assumes that ACKD was the last protocol message received, thus allowing acknowledgments from earlier initialization periods to get confused with later periods.

protocol and any starting state, assuming that there is no upper bound on link delay (see [Spi89] and [LMF88]). Problem 2.41 illustrates this phenomenon.

There are several solutions to the problem of initialization after a node failure. The first is to provide nodes with a small amount of nonvolatile storage. For example, the initialization protocols in this section can operate correctly with only a single bit of nonvolatile storage per adjacent link. If the links have a maximum propagation delay, another solution is simply to use sufficiently long time-outs in the initialization protocol to avoid the problems above (see [Spi89]). Yet another type of solution, which we explore later in terms of transport protocols, is to use a pseudo-random-number generator to avoid confusion between different periods of correct operation. We shall see that such protocols are very robust, although they have the disadvantage that there is a small probability of incorrect operation.

2.8 POINT-TO-POINT PROTOCOLS AT THE NETWORK LAYER

The major conceptual issues at the network layer, namely routing and flow control, involve algorithms that are distributed between all the nodes of the network; these issues are treated in Chapters 5 and 6, respectively. There are a number of smaller issues, however, that simply involve interactions between a pair of nodes. One of these issues is the transfer of packets between adjacent nodes or sites. At the network layer, this involves being able to distinguish packets of one session from another and being able to distinguish different packets within the same session. These issues are of particular concern between a source–destination site and the adjacent subnet node because of the need for standardization between users of the network and the subnet.

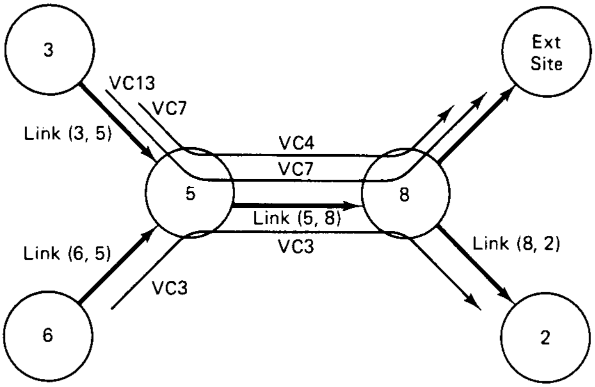
In the following subsections, we first treat the problem of addressing and session identification from a general viewpoint and give several examples to illustrate the range of possibilities. We then discuss packet numbering with a brief description of how this relates to flow control and to error recovery at the network layer. Finally, we discuss the X.25 network layer standard.

2.8.1 Session Identification and Addressing

The issue of session identification is relevant at the network layer. A packet received by a node must contain enough information for the node to determine how to forward it toward its destination. The brute-force approach to this problem is for the header of each packet to contain identification numbers for both the source and destination sites and additional identification numbers to identify the session within each site. This approach has great generality, since it allows different packets of the same session to travel over different paths in the subnet; it also requires considerable overhead.

If the network uses virtual circuits, far less overhead is required in the packet headers. With virtual circuits, each (one way) session using the network at a given time has a given path through the network, and thus there is a certain set of sessions using each link. It is helpful to visualize each link as being shared by a set of "virtual channels," distinguished by numbers. When a new session is set up, a path is established by assigning, on each link of the path, one unused virtual channel to that session. Each node then maintains a table mapping each busy incoming virtual channel on each link onto the corresponding outgoing virtual channel and link for the corresponding session (see Fig. 2.47). For example, if a given session has a path from node 3 to 5 to 8, using virtual channel 13 on link (3,5) and virtual channel 7 on link (5,8), then node 5 would map [link (3,5), virtual channel 13] to [link (5,8), virtual channel 7], and node 8 would map [link (5,8), virtual channel 7] to the access link to the appropriate destination site with a virtual channel number for that link. The nice thing about these virtual channels is that each link has its own set of virtual channels, so that no coordination is required between the links with respect to numbering.

With this virtual channel strategy, the only thing that must be transmitted in the packet header on a link is an encoding of the virtual channel number. The simplest way



Node 5 table
(3, 5) VC13 → (5, 8) VC7
(3, 5) VC7 → (5, 8) VC4
(6, 5) VC3 → (5, 8) VC3

Figure 2.47 Use of virtual channel (VC) numbers to establish virtual circuits in a network. Each packet carries its VC number on each link and the node changes that number (by a table representing the virtual circuit) for the next link on the path.

to do this, of course, is to represent the virtual channel number as a binary number in a fixed field of the header. Because of the limited capacity of the link, there is some natural upper bound on the number of sessions a link can handle, so there is no problem representing the virtual channel number by a fixed number of binary digits.

Later, other ways of representing virtual channel numbers will be discussed, but for now note that any method of distinguishing sessions must do something equivalent to representing virtual channel numbers. In other words, a given link carries data for a number of different sessions, and the transmitted bits themselves must distinguish which data belong to which session. The packet header might supply a great deal of additional information, such as source and destination identities, but the bare minimum required is to distinguish the sessions of a given link. We now look at how this representation is handled in TYMNET.

Session identification in TYMNET TYMNET is a network developed in 1970 primarily to provide access for a time-sharing computer service, but expanded rapidly to allow users to communicate with their own computers. Since the terminals of that period usually transmitted only one character at a time and usually waited for an echo of that character to come back before transmitting the next character, the protocols were designed to work well with very short messages consisting of one or several characters. Although the network never received much academic publicity, many of the ideas first implemented there are now standard network practice. The frames are very short, with a maximum length of 66 bytes; the frames have the format shown in Fig. 2.48.

The frame header of 16 bits is broken into five bits for a synchronization string, five bits for a word count, three bits for SN, and three bits for RN. The word count gives the number of 16 bit words in the frame, not counting the header and trailer. Since

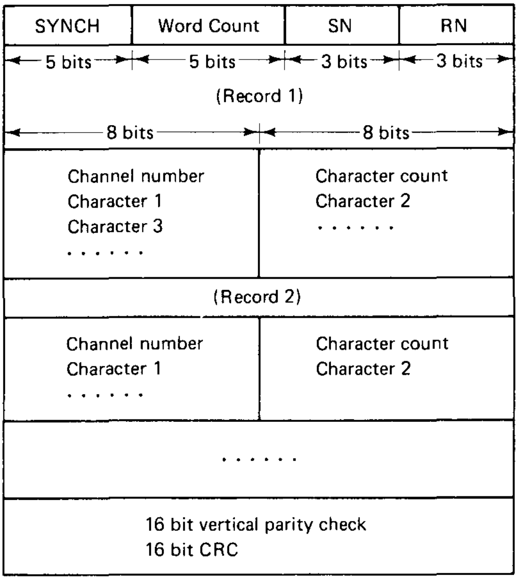


Figure 2.48 Frame structure in TYMNET. The first 16 bits is a frame header, and the last 32 bits is a frame trailer. Note that data from multiple sessions are included in a single frame.

the nodes were implemented with 16 bit minicomputers, all the fields of the frame are organized in 16 bit (two byte) increments. Note that the 5 bit word count can represent up to 31 words of two bytes each, or 62 bytes. Thus, restricting the frames to be multiples of 16 bits serves to reduce (slightly) the overhead required for framing. The SN and RN fields provide conventional go back n ARQ with a modulus of $m = 8$. The last four bytes (the frame trailer) provide error detection, using both a 16 bit CRC and 16 vertical parity checks. Thus, somewhat more error detection is provided than with the standard 16 bit CRC, but slightly less than with a 32 bit CRC.

Each record within a frame corresponds to data from a different session. A record starts with a byte giving the virtual channel number, which identifies the session. The next byte tells how many characters of data are contained in the record. If the number is odd, the last byte of the record is just a fill character, so that the next record can start on a word boundary. A record here can be thought of as a very short packet, so that the channel number and character count correspond to the packet header. Thus, the packets are so short that a number of packets are normally grouped into one frame so as to avoid excessive frame overhead. These packets are somewhat different from the usual concept of a packet in that they do not necessarily contain the same number of characters from one link to the next on their path.

This type of strategy is usually referred to as *statistical multiplexing*. This means that the link is multiplexed between the virtual channels, but multiplexed on a demand basis. Conventional packet switching, in which there is just one packet per frame, is another form of statistical multiplexing, but conventionally, one refers to that as packet switching rather than statistical multiplexing. When a frame is assembled for transmission at a node, the sessions are served in round-robin order, taking some number of characters from each session that has characters waiting.

Session identification in the Codex networks The Codex networks use quite a similar statistical multiplexing strategy, again with multiple sessions transmitting characters within the same frame. Here, however, a more efficient encoding is used to represent the virtual channel number and the amount of data for that session. In assembling a frame, the sessions are again served in sequential order, starting with virtual channel 1. Instead of sending the virtual channel number, however, an encoding is sent of the gap (*i.e.*, the number of idle virtual channels) between the number of the session being served and the previous session served (see Fig. 2.49). Thus, when all sessions have characters to send, these gaps are all zero, and if many but not all of the sessions have characters, most of the gaps are small. Thus, by encoding small gaps into a small number of bits, and large gaps into more bits, very high efficiency is maintained when large numbers of sessions are active. On the other hand, if only a small number of sessions are active, but have a large amount of data each, then only a small number of records will be in a frame, and again high efficiency is maintained.

Because of implementation issues, all of the fields in the Codex frames are multiples of four bit "nibbles." Thus, the smallest number of bits that can be used to represent a gap for identifying a session is one nibble. The method that identifies the number of characters in a record uses a special start record character, 0000, to start

Unary-binary gap encoding:
$0 \rightarrow 10$
$1 \rightarrow 11$
$2 \rightarrow 010$
$3 \rightarrow 011$
$4 \rightarrow 0010$
......

Start record character: 0000

Sample encoding of data:
$(0000)(10)(x_1)(x_2)(x_3)(x_4)(0000)(10)(x_1)(x_2)$
$(0000)(11)(x_1)(0000)(0010)(x_1)(x_2)(x_3)\ldots$

Figure 2.49 Efficient encoding for virtual channels and lengths of records. In the example shown, there are four characters x_1, x_2, x_3, x_4 for VC1, two characters for VC2, one for VC4, and three for VC9. Each record starts with 0000 to indicate the start of new record (where 0000 cannot occur in data characters); this is followed by the encoding of the gap between virtual channels with data. The Codex protocols are similar, but gaps are encoded into multiples of four bits. Parentheses are for visual clarity.

each new record. The characters for each session first go through a data compressor that maps highly probable characters into one nibble, less likely characters into two nibbles, and least likely characters into three nibbles. This is done adaptively, thus automatically using the current character statistics for the session. The 0000 nibble is reserved to indicate the start of a new record. Thus, the "packet header" consists of first a nibble to indicate a new packet (record), and next one or more nibbles representing the new session number. Under heavy loading, the packet header is typically just one byte long.

This strategy of encoding the difference between successive virtual channel numbers can also be used in conventional packet switched networks. There are two advantages in this. First, the packet overhead is reduced under heavy loading, and second, serving the sessions on a link in round-robin order tends to promote fairness in the service that each session receives.

2.8.2 Packet Numbering, Window Flow Control, and Error Recovery

In datagram networks, packets might be dropped and might get out of order relative to other packets in the same session. Thus, to achieve reliable communication, it is necessary to provide some way to identify successive packets in a session. The most natural and usual way of doing this is to number successive packets modulo 2^k for some k (*i.e.*, to provide a k-bit sequence number). This sequence number is placed in the packet header if the network layer modules at the source and destination have the responsibility for reordering the packets and retransmitting dropped packets. In many networks, for reasons we discuss later, the transport layer takes the responsibility for reordering and retransmission, and in that case the sequence number above is placed in the transport header. Sometimes sequence numbers are used at the network layer, at the internet sublayer, and at the transport layer.

For virtual circuit networks, there is also a need to number the packets in a session, but the reason is less obvious than for datagram networks. In our discussion of data link control, we saw a number of ways in which packets might get lost or arrive at the destination containing errors. First, a sequence of channel errors might (with very low probability) lead to an incorrect frame that still satisfies the CRC at the DLC layer. Second, errors made while data packets pass through a node are not checked by the CRC at the DLC layer. Third, if a link on the session path fails, there is uncertainty about how many packets were successfully transmitted before the failure. Finally, if a node fails, the packets stored at that node are typically lost. The error events described above are presumably quite rare, but for some sessions, correct transmission is important enough that error recovery at the network or transport layer is necessary.

Error recovery From a conceptual standpoint, error recovery at either the network or transport layer is quite similar to ARQ at the DLC layer. Each packet within a given session is numbered modulo $m = 2^k$ at the source site, and this sequence number SN is sent in the packet header at the network or transport layer. The destination site receives these packets and at various intervals sends acknowledgment packets back to the source site containing a request number RN equal (mod m) to the lowest-numbered yet-unreceived packet for the session. The receiver can operate either on a go back n basis or on a selective repeat basis.

It is important to understand both the similarity and the difference between the problem of end-to-end error recovery and that of error detection and retransmission at the DLC layer. End-to-end error recovery involves two sites and a subnet in between, whereas at the DLC layer, there are two nodes with a link in between. The sequence numbering for end-to-end error recovery involves only the packets (or bytes, messages, etc.) of a given session, whereas at the DLC layer, all packets using the link are sequentially numbered. At the DLC layer, we assumed that frames could get lost on the link, and similarly here we are allowing for the possibility of packets being thrown away or lost. At the DLC layer, we assumed variable delay for frame transmission, and here there is certainly widely variable delay. (Note that variable delay on a link is somewhat unnatural, but we assumed this so as to avoid worrying about processing delays.) The only real difference (and it is an important difference) is that frames must stay in order on a link, whereas packets might arrive out of order in a network.

Recall that datagram networks typically make no attempt to keep packets of the same session on the same path, so that out-of-order packets pose a generic problem for datagram networks. One would think that virtual circuit networks would be guaranteed to keep packets of the same session in order, but there are some subtle problems even here concerned with setting up new paths. If a link fails and a new path is set up for a session, the last packets on the old path might arrive out of order relative to the first packets on the new path. This problem can be avoided by a well-designed network layer, but for simplicity, we simply assume here that we are dealing with a datagram network and that packets can get out of order arbitrarily on the network.

Since packets can get lost, packets must sometimes be retransmitted end-to-end, and since packets can be delayed arbitrarily, a packet can be retransmitted while the earlier copy of the packet is lurking in the network; that earlier copy can then remain in the network until the packet numbering wraps around, and the packet can then cause an uncorrectable error. This scenario is perhaps highly unlikely, but it serves to show that end-to-end error recovery at the network or transport layer cannot be guaranteed to work correctly if packets are sometimes lost, sometimes get out of order, and sometimes become arbitrarily delayed.

There are three possible ways out of the dilemma described above: first, insist that the network layer use virtual circuits and maintain ordering even if the path for a session is changed; second, make the modulus for numbering packets so large that errors of the foregoing type are acceptably unlikely; and third, create a mechanism whereby packets are destroyed after a maximum lifetime in the network, using a modulus large enough that wraparound cannot occur in that time. It is shown in Problem 2.43 that the required modulus, using selective repeat, must be twice the window size plus the maximum number of packets of a session that can be transmitted while a given packet is alive. In Section 2.9 we discuss how the transport protocol TCP uses a combination of the last two approaches described above.

End-to-end error recovery also requires some sort of end-to-end check sequence such as a supplementary CRC to detect errors either made at the subnet nodes or undetected by the CRC at the link layer. In TCP, for example, this check sequence is 16 bits long and is the one's complement of the one's complement sum of the 16 bit words making up the packet. This type of check sequence is not as good at detecting errors as a CRC, but is easier to implement in software for typical processors. One interesting problem that occurs with end-to-end check sequences is that packets frequently change somewhat when passing from node to node. The primary example of this is the changing virtual channel numbers in the virtual channel addressing scheme discussed in Section 2.8.1. One certainly does not want to recompute the end-to-end check sequence at each node, since this would fail in the objective of detecting node errors.

The solution to this problem is to calculate the end-to-end check sequence before placing values in those fields that can vary from node to node. At the destination site, the check sequence is recomputed, again ignoring the values in those varying fields. This solution leaves the varying fields unprotected except by the link layer. For the example of virtual channel addressing, however, there is a good way to restore this protection. Both end sites for the session know the universal address of the other (from the setup of the session). Thus these addresses can be included in the check sequence, thus protecting against packets going to the wrong destination. Other parameters known to both end sites can also be included in the check sequence. As one example, we discussed the large end-to-end sequence number fields required to number packets in datagram networks. Only a few of the least significant bits are required to reorder the packets over a window, whereas the rest of the bits are required to ensure that very old packets (and very old acknowledgments) get rejected. By transmitting only the least significant bits of the sequence and request numbers, and including the more significant bits in the check sequence, one achieves greater efficiency at no cost in reliability.

Flow control The use of either go back n or selective repeat ARQ for end-to-end error recovery provides a rather effective form of flow control as an added bonus. If we ignore errors and retransmissions, we see that with a window size of n, at most n packets for a given session can be outstanding in the network at a given time. A new packet j can be sent only after an acknowledgment is received for packet $j - n$. The effect of this is that as congestion builds up and delay increases, acknowledgments are delayed and the source is slowed down. This is called end-to-end window flow control and is analyzed in Chapter 6. If the destination wishes to receive packets less rapidly, it can simply delay sending the current request number RN; thus this type of flow control can be effective in reducing congestion both in the subnet and at the destination site.

In the discussion of data link control we did not mention this flow control aspect of ARQ. The reason is that a link simply carries a fixed number of bits per second and there is no reason to limit this flow (other than the ability of the receiving node to handle the incoming data). Recall that standard DLCs provide a receive-not-ready supervisory packet as a way to protect the receiving node from too many incoming packets. This mechanism allows speedy acknowledgments, thus preventing unnecessary retransmissions due to time-outs, but also permits the receiving node to exert some flow control over the sending node.

Combining end-to-end acknowledgments with flow control has some limitations. If an acknowledgment does not arrive because of an error event, the packet should be retransmitted, whereas if the ack is delayed due to network congestion, the packet should not be retransmitted, since it would only add to the congestion. Since delays in the subnet are highly variable, it is difficult for time-outs to distinguish between these cases. To make matters worse, the destination might be overloaded and postpone sending acks as a way of preventing further transmissions from the source; this again might cause unnecessary retransmissions due to time-outs.

One partial solution to these problems is to provide the destination with a way to slow down the source without delaying acknowledgments. One way of doing this is by a technique such as the receive-not-ready supervisory frames in the standard data link controls. Such a frame both provides a speedy ack and prevents further transmission. This same technique can be used at the network layer (and is used in the X.25 network layer standard to be discussed shortly). A *permit* scheme is a more refined technique for the same objective. The destination sends two feedback numbers back to the source rather than one. The first, the usual RN, provides the ack function, and the second, a permit, tells the source how many additional packets the destination is prepared to receive. Thus if the permit number is j, the source is permitted to send packets from number RN to $RN + j - 1$ inclusive. In effect, this allows the destination to change the window size with each acknowledgment to the source at will. It allows the destination to send permits for only as many packets as will fit in the available buffers. This permit scheme could be provided in DLCs as well, but it is less useful there because the buffering requirements at a node are relatively modest because of the fixed number of links at a node.

Error recovery at the transport layer versus the network layer We have discussed end-to-end error recovery and its relationship to end-to-end window flow control. We now consider whether error recovery belongs at the transport layer or the network layer. From a practical standpoint, the answer now and in the near future is that error recovery belongs at the transport layer. The ISO transport layer standard known as

TP4 and the de facto standard TCP both provide for error recovery at the transport layer. Part of the reason for this is the prevalence of internetworking and the large number of networks that do not provide reliable service. The Internet Protocol, which is designed to work with TCP and which connects many universities and research labs, is itself an unreliable protocol, as will be discussed shortly.

The major disadvantage of having unreliable network and internet protocols and error recovery at the transport layer is that the transport layer has no way to distinguish between acknowledgments that are slow because of congestion and acknowledgments that will never arrive because the packet was thrown away. This problem is particularly severe for datagram networks. As a result, when the network becomes congested, packets start to be retransmitted, leading to still greater congestion, more dropped packets, and thus unstable behavior. There are a number of ad hoc approaches to reducing the severity of this problem, but it appears that error recovery at the transport layer will always lead to problems unless such recovery is required only rarely . This in turn will require almost reliable operation in the internet layer and in the various network layers.

2.8.3 The X.25 Network Layer Standard

The X.25 standard was developed by CCITT to provide a standard interface between external sites and subnet nodes. Recall that the physical layer, X.21, of this standard was discussed briefly in Section 2.2, and the DLC layer, LAPB, was discussed in Section 2.6. In X.25 terminology, the external site is referred to as DTE (data terminal equipment) and the subnet node as DCE (data communication equipment), but we shall continue to refer to them as sites and nodes.

The structure of an X.25 data packet is shown in Fig. 2.50. Recall that the DLC layer adds a frame header and trailer to this packet structure before transmission on the link. The third byte of the packet header is very similar to the control byte of the standard DLCs (except that SN and RN here refer to sequence numbers within the session, as discussed above). The control bit, C, is 0 for data packets and 1 for control packets (to be discussed later). This control bit is different from the control bit in the frame header; at the DLC layer, all packets, data and control, are treated indistinguishably, using 0 as the DLC layer control bit. Finally, the "more" bit, M, in the packet header is 1 for nonfinal packets in a message and 0 for the final packet.

The last half of the first byte and the second byte of the packet header is a 12-bit virtual channel number, as discussed in Section 2.7. This number can be different at the two ends of the virtual circuit for a given session. Many subnets use this same packet structure within the subnet, and the virtual channel number can then change from link to link, as described in Section 2.7.

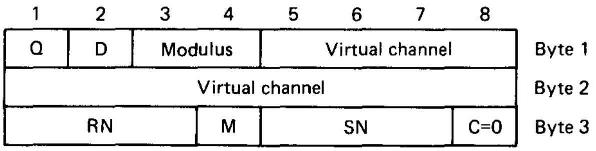


Figure 2.50 Packet header for data in X.25. For a control packet the C bit is 1.

The first bit, Q, in the packet header (if used at all) has the value 1 for control packets at the transport and higher layers and has the value 0 for control packets at the network layer and for data packets. The next bit, D, indicates whether the acknowledgment (RN) has end-to-end significance or just link significance. If $D = 1$, then RN signifies that the destination site has received all packets for the session previous to RN, that RN is awaited, and that the source is free to send a window of packets from RN on. If $D = 0$, RN signifies that the local subnet node or site has received packets previous to RN, that RN is awaited, and the other site or node is free to send a window of packets from RN on. It is possible for some sessions to use end-to-end significance and others, single-link significance. As discussed above, end-to-end acks are very helpful in providing error recovery. Some sessions, however, do not require error recovery, and some provide it at a higher layer, for example, by using a query–response interaction between the sites. For single-link significance, the acknowledgment function of RN is redundant (the DLC layer guarantees that the packet is received). Thus, RN is used strictly for flow control, and the receiving node delays sending RN until it is prepared to accept a window of packets beyond RN.

The third and fourth bits of the header indicate whether the modulus is 8 or 128. If the modulus is 128, the third byte is extended to two bytes to provide seven bits for SN and seven for RN.

The control packets in X.25 are quite similar to the supervisory frames and unnumbered frames in the standard DLCs. The third byte of the packet header indicates the type of control packet, with the final bit (control bit) always 1 to distinguish it from a data packet. The virtual channel number is the same as in a data packet, so that these control packets always refer to a particular session. There are receive-ready, receive-not-ready, and reject control packets, each of which carry RN; these are used for acknowledgments in the same way as in DLCs. The receive-not-ready packet provides some ability to acknowledge while asking for no more packets, but there is no ready-made facility for permits.

One interesting type of control packet is the call-request packet. After the three-byte header, this packet contains an address field, containing the length of the calling address and called address in the first byte, followed by the addresses themselves. This is used to set up a virtual circuit for a new session in the network. Note that the virtual channel number is of no help in finding the destination; it simply specifies the virtual channel number to be used after the virtual circuit is established. The subnet node, on receiving a call-request packet, sets up a path to the destination and forwards the packet to the destination site. If the recipient at the destination site is willing and able to take part in the session, it sends back a call-accepted packet; when this reaches the originating site, the session is established.

Both the call-request and call-accepted packet contain a facilities field; this contains data such as the maximum packet length for the session, the window size, who is going to pay for the session, and so on. The default maximum packet length is 128 bytes, and the default window size is 2. If the call-request packet specifies different values for maximum packet length or window size, the recipient can specify values closer to the default values, which must then be used. This is a sensible procedure for the external sites, but it is somewhat awkward for the subnet. The subnet can specify a maximum packet length and maximum window size to be used by the sessions using the subnet, but it is also desirable to have the ability to reduce window sizes under heavy loading, and this cannot be done naturally with X.25.

If the subnet is unable to set up the virtual circuit (because of heavy loading, for example), or if the recipient is unwilling to accept the call, a clear-request control packet goes back to the initiating site, explaining why the session could not be established.

2.8.4 The Internet Protocol

The Internet is a collection of thousands of networks interconnected by gateways. It was developed under the U.S. Department of Defense Advanced Research Projects Agency (DARPA) support as an effort to connect the myriad of local area networks at U.S. universities to the ARPANET. It now provides almost universal data access within the academic and research communities. The protocol developed to interconnect the various networks was called the Internet Protocol, usually referred to as IP. The backbone functions have since been taken over by the U.S. National Science Foundation (NSF), and the current topology is quite chaotic and still growing rapidly.

The individual networks comprising the Internet are joined together by gateways. A gateway appears as an external site to an adjacent network, but appears as a node to the Internet as a whole. In terms of layering, the internet sublayer is viewed as sitting on top of the various network layers. The internet sublayer packets are called datagrams, and each datagram can pass through a path of gateways and individual networks on the way to its final destination. When a gateway sends a datagram into an individual network, the datagram becomes the body of the packet at the network layer (see Fig. 2.51). Thus the individual network sends the datagram on to the next gateway or to the final destination (if on that network) in the same way that it handles other packets. Assuming that the individual network succeeds in getting the datagram to the next gateway or destination, the individual network looks like a link to the internet layer.

One of the goals of the Internet project was to make it as easy as possible for networks to join the Internet. Arbitrary networks were allowed to join the Internet, even if they allowed datagrams to be delayed arbitrarily, lost, duplicated, corrupted by noise, or reversed in order. This lack of reliability underneath was one of the reasons for making IP a datagram protocol; the problem of error recovery was left for the transport layer. Note that if an individual network happens to be a virtual circuit network, this poses no problem. Most virtual circuit networks can also handle datagrams, and alternatively, permanent virtual circuits could be established between gateways and sites on a given network to carry the internet datagrams between those gateways and sites. It would also have been possible to develop a virtual circuit internet protocol. This would have involved using error recovery at each gateway, thus making the individual networks appear like virtual circuit networks. We discuss this later.

Two transport layer protocols were developed along with IP. The first, the transmission control protocol, known as TCP, uses error recovery and reordering to present

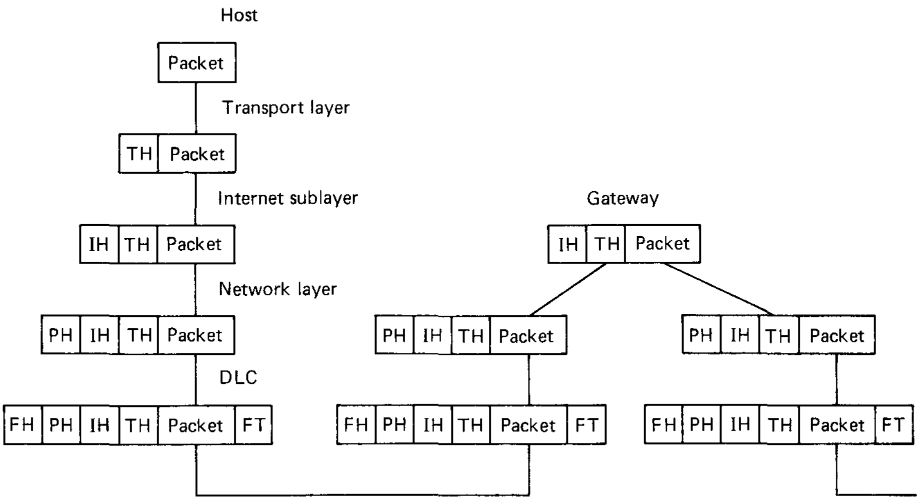


Figure 2.51 Illustration of a packet as it passes through a gateway. Within any given network, only the packet header and frame header are used, whereas at a gateway, the internet header is used to select the next network and to construct a new packet header appropriate to the new network.

the user with reliable virtual circuit service. The second, the user datagram protocol, simply deals with the question of where to send datagrams within a given destination.

The primary functions of IP are, first, to route datagrams through the Internet; second, to provide addressing information to identify the source and destination sites; and third, to fragment datagrams into smaller pieces when necessary because of size restrictions in the various networks. We shall not discuss routing here, since that is a topic for Chapter 5. We first discuss addressing and then fragmentation.

Since datagrams are being used, it is necessary for each datagram to contain identifications of the source and destination sites that are universally recognizable throughout the Internet (*i.e.*, the virtual channel addressing scheme of Section 2.8.1 is not applicable to datagram networks). Assigning and keeping track of these addresses is difficult because of the size and rapidly changing nature of the Internet. The addressing scheme used for IP is hierarchical in the sense that each site is identified first by a network identifier and then by an identifier for the site within the network. One peculiar consequence of this is that a site that directly accesses several networks has several addresses, and the address used determines which of those networks a datagram will arrive on.

The addresses in IP are all 32 bits in length, with a variable number of bits allocated to the network ID and the rest to the site (or host as it is called in IP terminology). In particular, the networks are divided into three classes, A, B, and C, according to size. The largest networks, class A, have a 7 bit network ID and a 24 bit host ID. Class B networks have 14 bits for network ID and 16 for host, and class C have 22 bits for network and 8 for host. The remaining one or two bits out of the 32 bit field are used to encode which class is being used. These addresses indicate only the source and destination sites. The intended process or user at the site is contained in additional information in the transport layer header discussed in Section 2.9.

The original length of a datagram is selected primarily for user convenience, but since individual networks have limitations on packet lengths, the datagrams must sometimes be fragmented into smaller pieces for passage through these networks. These fragments are datagrams in their own right (and can be further fragmented if need be) and the fragments are not combined into the original datagram until reaching the final destination. If one or more fragments are lost in the Internet, then, after some time-out, the ones that arrive at the destination are also thrown away, and (if TCP is being used at the transport layer) the entire datagram is eventually retransmitted by the source site. This appears to be quite inefficient, but the alternative would have been to do error recovery on the fragments in the internet layer.

When a datagram is fragmented, most of the header is simply copied to each fragment. The fragment length field (see Fig. 2.52), however, is modified to give the number of bytes in the given fragment. Fragmentation is always done on eight byte boundaries, and the fragment offset field gives the starting point of the given fragment in eight byte increments. One of the three flag bits in the header is used to indicate whether or not a fragment is the last fragment of the datagram. The final piece of information needed at the destination to reconstruct a datagram from its fragments is the datagram ID. The source site numbers successive datagrams, and this sequence number is used as the datagram ID; this is copied to each fragment and allows the destination to know which fragments are part of the same datagram. It is not difficult to see that the combination of fragment length, offset, last fragment bit, and datagram ID provided for each fragment allows the original datagram to be reconstructed.

We now give a quick summary of what the other fields in the IP datagram header do. The field "Vers" tells which version of the protocol is being used. This is useful when a new version of the protocol is issued, since it is virtually impossible to synchronize a changeover from one version to another. The field "HL" gives the length of the header (including options) in 32 bit increments. "Service" allows users to specify what kind of service they would like, but there is no assurance that the gateways pay any attention to this.

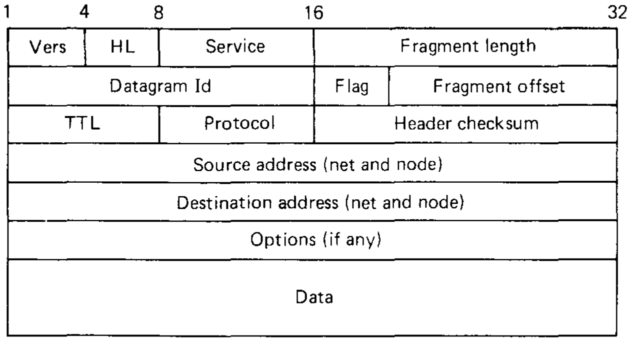


Figure 2.52 IP datagram header format in 32 bit words. Note that the minimum-size header is 20 bytes.

The time to live field, "TTL," is decremented each time the datagram reaches a new gateway, and the datagram is thrown away when the count reaches zero. This is supposed to prevent old datagrams from existing in the Internet too long, but unfortunately, there is no mechanism to prevent a datagram from remaining in one of the individual networks for an arbitrarily long period. The field "protocol" indicates which higher-layer protocol is to operate on the datagram. This is necessary since the information about where the datagram is to go at the destination site is in the transport header, and the format of this header depends on which transport protocol is being used. Finally, the header check sum checks on the header. This seems a little peculiar for a datagram, but as a practical matter, one does not want datagrams in a large internet going to some random destination in the event of an error.

2.9 THE TRANSPORT LAYER

The transport layer is the fourth layer in the OSI layered architecture and has the functions, (1) of breaking messages into packets, (2) of performing error recovery if the lower layers are not adequately error free, (3) of doing flow control if not done adequately at the network layer, and (4) of multiplexing and demultiplexing sessions together. The first function, breaking messages into packets, is relatively straightforward, aside from the question of selecting the maximum packet length for a network. As discussed in the preceding subsection, even the selection of packet size is less important for TCP combined with IP, since IP fragments packets when necessary; thus, the transport layer packets going from TCP to IP can really be viewed as messages with a transport layer header.

We have already discussed the general problem of end-to-end error recovery and the reasons why it is normally done at the transport layer. Rather than discussing the other functions of the transport layer in general, we shall discuss them in terms of TCP and the ISO standards.

2.9.1 Transport Layer Standards

One has a wide choice of standard transport protocols. The International Standards Organization (ISO), in its OSI layered architecture, has five different classes of transport protocol (denoted TP), numbered 0 to 4. TP class 0 is a relatively simple protocol that splits messages into packets and reassembles them at the destination but provides neither for multiplexing of sessions nor for error recovery. Class 1 allows for some error recovery but relies on the network layer and consistency checks for detection of the need for retransmission. Class 2 provides multiplexing but no error recovery, and class 3 provides the error recovery of class 1 and the multiplexing of class 2. Finally, class 4 provides multiplexing plus a more complete form of error recovery using both detection of errors and retransmission. The class 4 protocol is designed for use with datagram networks and other networks that do not adequately deliver each packet once, only once, correctly, and in order.

The U.S. National Institute for Standards and Technology has also developed a U.S. Federal Information Processing Standard (FIPS) for the transport layer; this is essentially the same as TP classes 2 and 4 of the OSI standard except for some additional options.

The U.S. Department of Defense has standardized TCP (transmission control protocol). As discussed in the preceding subsection, this protocol was originally developed for the ARPANET and was designed to operate on top of IP, thus allowing internetworking between the ARPANET and other networks. TCP was developed before the OSI TP protocols, and TP class 4 uses many of the features of TCP. In what follows, we discuss TCP and then contrast it briefly with TP class 4.

2.9.2 Addressing and Multiplexing in TCP

In Section 2.8.4 we discussed addressing in the Internet and pointed out that the IP header must contain sufficient information to route the given packet through the Internet, namely a network ID and a host ID. One of the basic tenets of layering is that the header (and perhaps trailer) of a packet at a given layer contains all the information necessary for the module at that layer to do its job; the body of the packet at that layer (which contains the headers for the higher layers) should not be consulted by the module at the given layer. When a packet is passed from TCP to IP at a given node, there are extra parameters passed along with the packet that provide IP with addressing information. Similarly, when the datagram arrives at its destination, the IP header is stripped off, but the addressing information and other header information can be passed up as parameters.

In TCP, the entire address of a source (or destination) is called a *socket.* It is organized hierarchically as, first, network ID, then host ID, and finally, the user or process ID within that node. The user or process ID within a host is called a *port* in TCP (*i.e.*, the transport layer is viewed as having many ports accessing the next higher layer, one for each session). The port ID, for both source and destination, is included in the transport header, whereas the network and node IDs appear in the IP header. This means that all sessions going from a given source host to a given destination host will normally have the same source and destination address in the IP header and will be distinguished only in the transport header. Thus all sessions between the same pair of hosts could be viewed as being multiplexed together at the transport layer into a common lower-layer session. The fact that IP is a datagram protocol is irrelevant from this standpoint; TCP could also operate with an X.25 form of network layer, and then the network layer would have to set up a session and map the host address into an address in the X.25 packet header.

It is possible to have multiple logical addresses for the same host in TCP. Thus one could avoid multiplexing by using a separate address in the IP header for each transport layer session. The general point to be learned from this, however, is that transport headers, internet headers, and network headers all have address fields. Sometimes there is redundancy between the information in these various fields, but the information in the network header must be sufficient for routing and flow control in an individual network. Similarly, the internet header plus the network header must be sufficient for routing between gateways. Finally, the transport header must be sufficient to distinguish all sessions multiplexed together.

The size of the address fields at different layers in the various standards is somewhat bizarre. TCP uses 16 bit port numbers, and TP class 4 similarly uses 16 bits for a reference number that performs the same role. IP uses 32 bit addresses with 8, 16, or 24 bits representing the node and the rest representing the component network (along with an encoding of whether 8, 16, or 24 bits represent the node). On the other hand, X.25 uses only 12 bits as a virtual circuit number. The use of virtual circuits allows part of this efficiency at the network layer, but one still wonders why the standard network layer is so stingy with address bits and the transport layer and internet sublayer are so profligate. This same concern for efficiency at the network (and DLC) layer and lack of concern at higher layers runs through all the standard protocols. One might have hoped that with the enormous flexibility in these protocols, some flexibility allowing a little efficiency would have been included (such as variable-length address fields), but the designers of transport protocols and internet protocols seem dedicated to the principle that bigger is better. The large size of address fields at the transport and IP levels and the small size at the network layer sometimes inhibit maintaining session identities (and thus providing different types of service, such as delay and error recovery to different sessions) at the network layer.

2.9.3 Error Recovery in TCP

Error recovery in TCP follows the general pattern described in Section 2.8.2; the strategy is selective repeat ARQ. The transport layer packets each contain both a 32 bit sequence number, SN, and a 32 bit request number, RN, in the packet header. The number SN is used in a rather peculiar way in TCP, since it counts successive bytes in the session data rather than successive packets. In particular, SN is the number of the first data byte in the given transport layer packet. Thus if a given transport layer packet has a sequence number $SN = m$ and contains n bytes in the packet body (*i.e.*, the part of the packet labeled "data" in Fig. 2.53), then the next transport layer packet for that session has $SN = m + n$. There appears to be no good reason for this, but it does not do any harm, other than adding complexity and overhead. One might think that this type of numbering would make it easier to fragment packets into smaller pieces for internetworking, but as we have seen, IP has its own mechanisms for handling fragmentation. The request number RN, as one might expect, denotes the next desired

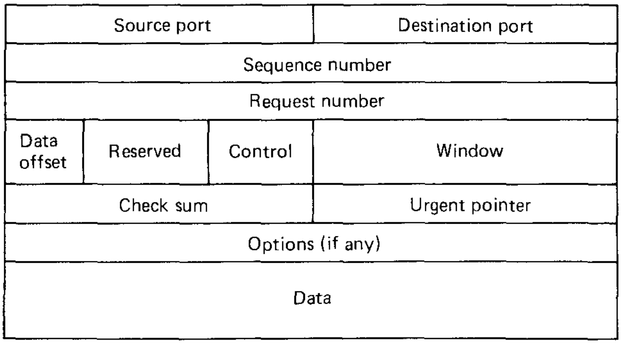


Figure 2.53 TCP header format in 32 bit words; note that the minimum-sized header is 20 bytes.

packet in the opposite direction by giving the number of the first data byte in that packet. As with most ARQ schemes, the RN field allows acknowledgments to be piggybacked on data for the given session in the reverse direction, and in the absence of such data, a packet with just the header can carry the RN information.

As discussed earlier, errors are detected in TCP by a 16 bit check sum calculated from both the transport layer packet and some parameters (such as source and destination address and packet length) that are passed to the next lower layer. These addresses are also checked in IP with a header check sum, but TCP is supposed to work with other lower layers, so this provides worthwhile extra protection.

From the description above, it should be clear that error recovery for packets within an established connection at the transport layer is not a logically complicated matter . The same principles apply as at the DLC layer, and the only new feature is that of enlarging the modulus to allow for out-of-order packet arrivals. There are several complications, however, that remain to be discussed.

The first such complication is that of retransmission time-outs. At the DLC layer, retransmission time-outs provide an interesting issue to think through in understanding how distributed algorithms work, but the issue is not very critical. At the transport layer, the issue is quite serious, since high congestion in the network causes large delays, which trigger retransmissions from the transport layer, which further increase delays. When the congestion becomes bad enough, packets will also be discarded in the network. This triggers more retranmissions, and eventually throughput is reduced to almost zero. One advantage of doing error recovery at the network layer is that it is possible to provide better estimates on delay, and thus avoid retransmission under high-delay conditions.

The final complication is the problem of error recovery when setting up and tearing down connections. What makes this problem tricky is that one does not want to save information about connections after they have been torn down. Thus there is the possibility of a packet from an old connection entering a node after a new connection has been set up between the same processes. There is a similar possibility of confusion between the packets used to initiate or tear down a connection, and finally there is a possibility of confusion when one node crashes and loses track of connections that still exist at other nodes.

In TCP, the major mechanism used to deal with the initiation and disconnection of sessions is the synchronization of the sequence number and request number at the initiation of a session. Each node has a clock counter (not synchronized between nodes) that increments once each 4 μsec. When a new connection is attempted, this clock counter is used as the initial sequence number for the first packet in an initiation handshake. The opposite node responds by acknowledging this sequence number in its request number (as usual) and using its own clock counter for its own initial sequence number. Since the clocks are incrementing faster than the sequence numbers of any established connection can increment (at least for data nets at today's speeds) and since the 32 bit clock counter wraps around less than once in 4 hours, this mechanism solves most of the problems noted above. With this type of protocol, the responding node cannot start to send data (beyond this initial packet and its potential repetitions) until it receives a second packet from the initiating node acking the responding node's first packet. Because of this third required packet, this type of initiation is called a three-way handshake. We discussed this type of handshake in Section 2.7. What is new here is that avoiding confusion between old and new connections is achieved by a pseudorandom choice of sequence numbers rather than by using an alternation of connection and disconnection. The strategy here has the disadvantage of potential malfunction under arbitrary delays, but the use of a 32 bit number makes this possibility remote.

The major type of problem for this kind of handshake is that it is possible for one node to believe that a given connection is open while the other node believes that it is closed. This can happen due to one node crashing, the network disconnecting temporarily, or incomplete handshakes on opening or closing. If the node that believes the connection is open sends a packet, and then the other node starts to initiate a connection, it is possible for the new connection to get set up and then for the old packet to arrive. If the old connection had been in existence for a very long time, it is possible for both the sequence and request numbers to be in the appropriate windows, causing an error. This type of situation appears very unlikely, especially considering the large amount of redundancy in the rest of the header.

For completeness, the format of the header in TCP is shown below as a sequence of 32 bit words. We have discussed the source port, destination port, sequence number, request number, and checksum. The data offset gives the number of 32 bit words in the header, the urgent pointer has to do with high-priority data and need not concern us, and the options also need not concern us. Each of the six control bits has a particular function. For example, the fifth control bit is the SYN bit, used to indicate an initiation packet that is setting the sequence number field. The second control bit is the ACK bit, used to distinguish the first initiation packet (which does not carry an ACK) from the second and subsequent packets, which do carry an ACK.

2.9.4 Flow Control in TCP/IP

TCP uses a permit scheme to allow the destination to control the flow of data from the source host. This is implemented by a 16 bit field called a *window* stating how many bytes beyond the request number can be accepted for the given connection by the given node. This provides some rationale for TCP to count bytes (or some fixed-length unit of data), since there is such high potential variability in datagram lengths. For a given size of window, the level of congestion in the Internet (or network in the case of no internetworking) controls how fast data flow for the given connection. Thus this scheme both protects the destination from being saturated and slows down the sources in the event of congestion.

It is also possible for a source to monitor the round-trip delay from datagram transmission to acknowledgment receipt. This can provide an explicit measure of congestion and there are many strategies that use such measurements to constrain data flow into the Internet or subnet. There are some inherent difficulties with strategies that use such measurements to decrease source flow. The first is that such strategies are subject to instability, since there is a delay between the onset of congestion and its measurement. This is made worse by throwing away packets under congestion. The next problem is that "good citizen" sessions that cut back under congestion are treated unfairly relative to sessions that do not cut back. Since the Internet uses datagrams, the Internet cannot distinguish "good citizen" sessions from bad, and thus datagrams are thrown away without regard to session.

It is important to recognize that these are inherent problems associated with any datagram network. The network resources are shared among many sessions, and if the resources are to be shared fairly, the network has to have some individual control over the sessions. The session that a datagram belongs to, in principle, is not known at the network layer (note from Fig. 2.52 that the IP protocol has no header field to indicate session number), and thus the datagram network or internet cannot monitor individual sessions. A corollary of this is that datagram networks cannot provide guaranteed types of service to given sessions and cannot negotiate data rate. A datagram network might give different classes of service to different kinds of datagrams, but the choice of service class comes from the transport layer and is not controlled by the network. If some sessions place very heavy demands on the network, all sessions suffer.

TCP/IP has been enormously successful in bringing wide area networking to a broad set of users. The goal of bringing in a wide variety of networks precluded the possibility of any effective network or internet-oriented flow control, and thus precluded the possibility of guaranteed classes of service. How to provide such guarantees and still maintain internetworking among a wide variety of networks is a challenge for the future.

2.9.5 TP Class 4

The ISO transport protocol, TP class 4, is very similar to TCP, with the following major differences. First, sequence numbers and request numbers count packets rather than bytes (in the jargon of the standards, packets at the transport level are called transport protocol data units or TPDUs). Next, acknowledgments are not piggybacked on packets going the other way, but rather, separate acknowledgment packets are sent (using the same selective repeat strategy as before). These ack packets are sent periodically after some time-out even with no data coming the other way, and this serves as a way to terminate connections at one node that are already terminated at the opposite node.

Another difference is that sequence numbers are not initiated from a clock counter, but rather start from 0. The mechanism used in TP class 4 to avoid confusion between old and new connections is a destination reference number. This is similar to the destination port number in TCP, but here the reference number maps into the port number and can essentially be chosen randomly or sequentially to avoid confusion between old and new connections.

2.10 BROADBAND ISDN AND THE ASYNCHRONOUS TRANSFER MODE

In Sections 2.8 and 2.9 we have described some of the point-to-point aspects of network layer and transport layer protocols. In this section we describe a new physical layer protocol called asynchronous transfer mode (ATM). From a conceptual viewpoint, ATM performs the functions of a network layer protocol, and thus we must discuss why network layer functions might be useful at the physical layer. Since ATM was developed for use in broadband ISDN systems, however, we start with some background on broadband ISDN.

A broadband integrated services digital network (broadband ISDN) is a network designed to carry data, voice, images, and video. The applications for such networks are expected to expand rapidly after such networks are available. Recall that in our discussion of ISDN in Section 2.2, we pointed out that a high-resolution image represented by 10^9 bits would require over 4 hours for transmission on a 64 kbit/sec circuit and would require 11 minutes on a 1.5 Mbit/sec circuit. Broadband ISDN is an effort to provide, among other things, data rates that are high enough to comfortably handle image data in the future. The planned access rate for broadband ISDN is 150 Mbit/sec, which is adequate for image traffic and also allows for the interconnection of high-speed local area networks (see Section 4.5.5). This access rate also allows video broadcast traffic, video conferencing, and many potential new applications.

The technology is currently available for building such networks, and the potential for new applications is clearly very exciting. However, the evolution from the networks of today to a full-fledged broadband ISDN will be very difficult, and it is debatable whether this is the right time to standardize the architectural details of such a network. Some of the arguments for rapid standardization are as follows. There are current needs for these high data rates, including image transmission, accessing high-speed local area networks, and accessing supercomputers; the rapid development of such a network would meet these needs and also encourage new applications. Many people also argue that the cost of networks is dominated by the "local loop" (*i.e.*, the access link to the user); optical fiber will increasingly be installed here, and economy dictates combining all the telecommunication services to any given user on a single fiber rather than having multiple networks with separate fibers. The difficulty with this argument is that it is not difficult to multiplex the access to several networks on a single fiber. Another popular argument for the rapid deployment of broadband ISDN is that there are economies of scale in having a single network provide all telecommunication services.

One of the arguments against rapid standardization is that there are also diseconomies of scale involved with a single all-purpose network. These diseconomies come from trying to force very different types of communication requirements, such as video and conventional low-speed data, into a common mold. Another argument is that there is no large-scale engineering experience with such networks and little knowledge of how the potential applications will evolve. This is exacerbated by the rapid development of optical fiber technology, which is rapidly changing the range of choices available in the backbone of a broadband network. Finally, the evolution of broadband ISDN is central to the future of the telecommunication industry, and thus political, legal, and regulatory issues, as well as corporate competitive positioning, will undoubtedly play a large and uncertain role in the evolution and structure of future networks. In what follows we look only at technological issues, but the reader should be aware that this is only part of the story.

Broadband ISDN can be contrasted both with data networks and with the voice network. In contrast with data networks, there is an increase of over three orders of magnitude in the link speeds envisioned for broadband ISDN. Because of this, broadband

ISDN designers are more concerned with computational complexity and less concerned with efficient link utilization than are designers of conventional data networks. Because of the speed constraints, the protocols in broadband ISDN should be simple and amenable to both parallel and VLSI implementation.

Another important contrast with data networks is that individual user requirements are critically important in broadband ISDN. Part of the reason for this is that voice and video normally require both a fixed bit rate and fixed delay, whereas packet-switched data networks generally have variable delays and often use flow control to reduce data rates. In the past, data applications that required fixed rate and/or fixed delay usually avoided data networks and instead used dedicated leased lines from the voice network. In other words, traditional data networks provide a limited range of service, which users can take or leave. Broadband ISDN, on the other hand, is intended to supply all necessary telecommunication services, and thus must provide those services, since there will be no additional voice network on which to lease lines.

Next consider the contrast of broadband ISDN with the voice network. All voice sessions on a voice network are fundamentally the same. Except on the local loops, voice is usually sent digitally at 64 kbits/sec, with one eight bit sample each 125 μsec. In Section 2.2 we described briefly how these sessions are time-division multiplexed together. The important point here is that all of these multiplexing systems, including the SONET standard, use a 125 μsec frame broken up into a large number of one byte slots. Switching is accomplished at a node by mapping the slots in the incoming frames into the appropriate slots of the outgoing frames. In contrast, broadband ISDN must cope with potential user rates from less than 1 bit/sec to hundreds of megabits per second. It must also cope with both bursty traffic and constant-rate traffic.

Despite the enormous difference between broadband ISDN and the voice network, broadband ISDN will probably evolve out of the present-day voice network if and when it is built. The reason for this is that the bulk of transmission and switching facilities currently exist within the current voice network, and current wide area data networks typically lease their transmission facilities from the voice network. Because of this, the early thinking about broadband ISDN focused on using the underlying synchronous 125-μsec frame structure of the voice network to provide the physical layer transmission.

Within the frame structure of the voice network, it is possible to group multiple slots per frame into a single circuit whose rate is a multiple of 64 kbits/sec. These higher-rate circuits can be leased either for dedicated data channels or for use as links in data networks. They come in standard multiples of 64 kbits/sec (*e.g.*, DS1 service is 24 times 64 kbits/sec and DS3 service is 28 times the DS1 speed). These higher-rate circuits could also be used in broadband ISDN for high-rate services such as video. In this sense, broadband ISDN was initially visualized as an evolution from (narrowband) ISDN, maintaining the emphasis both on 64 kbit/sec circuit switched sessions and on various standard multiples of the 64 kbit/sec speed. The early objective for broadband ISDN was simply to increase data rates to allow for video, high-resolution images, and interconnection of high-speed local area networks.

As interest in broadband ISDN increased and as standardization efforts started, an increasing number of people questioned the underlying reliance on the synchronous frame structure of the voice network and proposed using an asynchronous packet structure for multiplexing at the physical layer. As this debate quickened, use of the conventional synchronous frame structure became known as STM (synchronous transfer mode), and the use of a packet structure became known as ATM (asynchronous transfer mode). Fundamentally, therefore, the choice between STM and ATM is the choice between circuit switching and packet switching.

As discussed in Section 1.2.3, conventional data networks use packet switching principally because of the burstiness of the data, but burstiness is far less important for broadband ISDN. In the first place, voice traffic totally outweighs conventional data traffic and will probably continue to do so (see Problem 1.2). Next, circuit switching is already used in the voice network, is ideal for voice traffic, and is well suited for video traffic. High-resolution images would not fit well into circuit switching, but could be handled (along with conventional data) on a packet network using the transmission resources of a circuit-switched net as a physical layer.

A more important aspect of the choice between STM and ATM is the question of the flexibility of STM for handling a wide variety of different data rates. STM would require each circuit-switched application to use some standard multiple of 64 kbits/sec. This would lead to great switching complexity if the number of standard rates were large, and to great inefficiency otherwise. In addition, the proponents of ATM argue that the use of a small set of standard rates would inhibit the development of new applications and the development of improved data compression techniques.

This lack of flexibility in STM is particularly troublesome at the local loop. Here, given a small set of sessions, each would have to be assigned one of the standard rates. Any attempt to multiplex several sessions (particularly bursty data sessions) on one standard rate, or to use several standard rates for one session, would lead to considerable complexity. In comparison with these problems of matching sessions to standard rates, the ATM solution of simply packetizing all the data looks very simple.

While the relative merits of STM and ATM were being debated, there was great activity in developing high-speed packet-switching techniques. These techniques were highly parallelizable and amenable to VLSI implementation. As a result of both the flexibility of ATM for local access and the promise of simple implementation, the CCITT study group on broadband ISDN selected ATM for standardization.

Before describing ATM in more detail, we should understand how it fits into the OSI seven-layer architecture. In the original view of ISDN, the synchronous frame structure of STM was quite properly viewed as the physical layer. Thus, when ATM replaced STM, it was reasonable to consider ATM as the physical layer for data transmission. This had the added advantage of relieving ATM from any responsibility to conform to the OSI standard network layer. Another consequence of using ATM in place of STM, however, is that there is no longer a need for a packet-switched network on top of ATM; ATM deals with bursty data directly as a packet-switching technique in its own right. We will see later that ATM, as a packet-switching layer, has an *adaptation layer* on top of it. This adaptation layer is much like the transport layer of OSI; it breaks up the incoming messages or bit streams into fixed-length packets (called cells in ATMese), and it is the responsibility of the adaptation layer, using the ATM layer, to reconstitute the messages or data stream at the other end. What comes in could be OSI layer 2 frames: the adaptation layer will view this as a user message, break it into "cells," and reconstitute the user message (*i.e.*, the frame) at the destination. Thus, as far as the OSI layer 2 is concerned, the broadband ISDN looks like a bit pipe. Naturally, there is no need for a user to go through all the OSI layers, since data can be presented directly to the broadband ISDN.

2.10.1 Asynchronous Transfer Mode (ATM)

ATM packets (or cells) all have the same length, 53 bytes, consisting of 5 bytes of header information followed by 48 bytes of data (see Fig. 2.54). The reason for choosing a fixed length was that implementation of fast packet switches seems to be somewhat simpler if packets arrive synchronously at the switch. One might question being so restrictive in a long-term standard because of a short-term implementation issue, but the standards committee clearly felt some pressure to develop a standard that would allow early implementation.

The major reason for choosing 48 bytes of data for this standard length was because of the packetization delay of voice. Standard 64 kbit/sec voice is generated by an eight bit sample of the voice waveform each 125 μsec. Thus, the time required to collect 48 bytes of voice data in a cell is 6 msec. Even if a cell is transmitted with no delay, the reconstruction of the analog voice signal at the destination must include this 6 msec delay. The problem now is that this reconstructed voice can be partially reflected at the destination and be added to the voice in the return direction. There is another 6 msec packetization delay for this echo on the return path, and thus an overall delay of 12 msec in the returning echo. A delayed echo is quite objectionable in speech, so it should either be kept small or be avoided by echo cancellation. In the absence of echo cancellation everywhere, it is desirable to keep the delay small.

There are several reasons why a larger cell length would have been desirable. One is that there is a fixed amount of computation that must be done on each cell in an ATM switch, and thus, the computational load on an ATM switch is inversely proportional to the amount of data in a cell. The other relates to the communication inefficiency due to the cell header (and as we shall see later, the header in the adaptation layer). The length of 48 bytes was a compromise between voice packetization delay and these reasons for wanting larger cells.

The format of the header for ATM cells has two slightly different forms, one for the access from user to subnet and the other within the subnet (see Fig. 2.55). The only difference between the header at the user–subnet interface and that inside the network

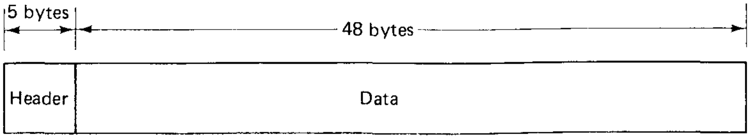
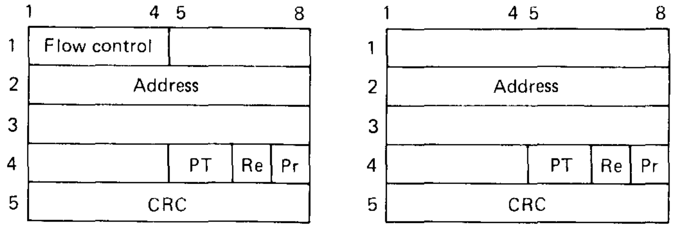


Figure 2.54 Packet (cell) format in ATM. Note that the cell size is fixed.



User–subnet access

Internal subnet

Figure 2.55 ATM header format for the user–subnet interface and for the internal subnet. PT stands for payload type, Re for reserved, and Pr for priority bit.

is a four bit "generic flow control" field provided at the user–subnet interface. This is intended to allow the user to mediate between the demands of different applications at the user site. The use of this field is not standardized and is used only to help the user to statistically multiplex the cells from different applications onto the access link to the network. The question of how to accomplish flow control within the broadband ISDN network is discussed in Section 2.10.3, but there is no field in the ATM header (other than the priority bit) to help accomplish this function.

The contents of the address field in the header can change as the cell travels from link to link, and aside from a detail to be discussed later, this address is used in the same way as the virtual channel numbers described for addressing in Section 2.8. That is, ATM uses virtual circuits, and when a virtual circuit is set up for a session, a virtual channel number is assigned to that session on each link along the session's path. Since the address field contains 24 bits for the user–network interface, over 16 million sessions can access the network from one user site. Since the "user" might be a corporation or a local area network, for example, this is not quite as extravagant as it sounds. Within the network, the address field contains 28 bits, allowing for over 268 million sessions to share a link.

There is an additional quirk about addressing in ATM. The address field is divided into two subfields, called the virtual channel identifier (VCI) and the virtual path identifier (VPI) in ATMese. The reason for this is to allow sessions sharing the same path (or at least sharing the same path over some portion of their entire paths) to be assigned the same virtual path identifier and to be switched together. For example, two office buildings of a given corporation might require a large number of virtual circuits between the two offices, but they could all use the same virtual path identifiers and be switched together within the network. The VCI subfield contains 16 bits, and the VPI subfield contains 8 bits at the user–subnet interface and 12 bits within the subnet.

After the address field comes the "payload type" field, the "reserved field," which is not used, and the "priority bit" field, which is discussed in Section 2.10.3. The payload type field is used to distinguish between cells carrying user data and cells containing network control information.

Finally, at the end of the header is an eight bit CRC checking on the ATM header. The generator polynomial of this CRC is $g(D) = D^8 + D^2 + D + 1$. This is the product of a primitive polynomial of degree 7 times $D + 1$, and, as explained in Section 2.3.4, the header thus has a minimum distance of 4. Note that this CRC checks only on the cell header, not on the data. Thus errors in the data are delivered to the destination and some other means is required to achieve error-free transmission of user information. The reason for checking the header is to reduce the probability of having data for one user get delivered to another user.

One very unusual feature of this CRC is that it is sometimes used to *correct* errors rather than just to detect them. The reason for this is that broadband ISDN is intended for use on optical fiber channels, and at least with current modulation techniques, errors are very improbable (the error rate is less than 10^{-9}) and more or less independent. Thus single error correction might actually remove most header errors. In an attempt to guard against bursts of errors due to equipment problems, the receiving node switches between single error correction and pure error detection as illustrated in Fig. 2.56. After either correcting an error or detecting more than a single error, the decoder switches to an error-detecting mode and stays there until the next header is received without detected errors. This is quite a novel approach to improving the reliability of optical channels in a simple way. The only problem is that optical technology is still rather fluid, and this approach is very dependent both on a very low channel error probability and on an extremely low probability of multiple errors.

We next investigate how ATM interfaces to the physical channel underneath. Since ATM performs the functions of a network layer, we would expect to find some sort of data link control layer beneath it. However, in this case neither ARQ nor framing is required, so there is no need for DLC. There is no need for ARQ for two reasons: first, the error probability is very low on the links, and second, a large part of the traffic is expected to be voice or video, where retransmission is inappropriate. Thus, error recovery is more appropriate on an end-to-end basis as required. In North America and Japan, ATM will use SONET as a lower layer and there are control bytes within the SONET frame structure that can specify the beginning of an ATM cell within the SONET frame. This of course explains why the user access rate for ATM was standardized at the SONET STS-3 rate. Since all cells are of the same length, framing information is not needed on each cell. Some European countries are not planning to use SONET as a lower layer to ATM, so they will require some other means of framing.

Since framing information is not provided on each cell, it is necessary to be careful about what to send when no data are available. One solution is to send "unassigned cells" which carry no data but fill up a cell slot. A special reserved virtual circuit number is used to identify this type of cell so that it can be discarded at each ATM switching point.

The use of SONET underneath ATM has a number of other advantages. One advantage is that very-high-data-rate applications could have the possibility of using SONET directly rather than being broken down into tiny ATM cells that must be individually

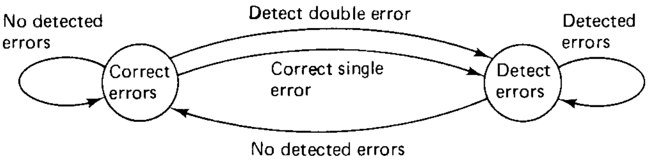


Figure 2.56 Error correction and detection on the headers of ATM cells. A single error is corrected only if the preceding cell header contained no detected errors.

switched at each node. Another is that SONET circuits carrying ATM traffic could be switched directly through intermediate nodes rather than being broken down at each switch into ATM cells. If one takes the ATM user access rate of 150 mbits/sec seriously, one can see that backbone links might be required to carry enormous data rates, and switches that operate on 53 bytes at a time are not attractive. If one looks further into the future and visualizes optical fibers carrying 10^{12} or 10^{13} bits/sec, the thought of ATM switches at every node is somewhat bizarre.

2.10.2 The Adaptation Layer

The adaptation layer of broadband ISDN sits on top of the ATM layer and has the function of breaking incoming source data into small enough segments to fit into the data portions of ATM cells. In the sense that ATM is performing the functions of a network layer, the adaptation layer performs the function of a transport layer. Thus the adaptation layer is normally used only on entry and exit from the network. Since the adaptation layer interfaces with incoming voice, video, and message data, its functions must depend on the type of input. The adaptation layer is not yet well defined within the standards bodies and is in a state of flux. Thus we give only a brief introduction to the way in which it accomplishes its functions.

Different types of input are separated into the following four major classes within the adaptation layer:

Class 1: Constant-bit-rate traffic. Examples of this are 64 kbit/sec voice, fixed-rate video, and leased lines for private data networks.

Class 2: Variable-bit-rate packetized data that must be delivered with fixed delay. Examples of this are packetized voice or video; the packetization allows for data compression, but the fixed delay is necessary to reconstruct the actual voice or video signal at the receiver.

Class 3: Connection-oriented data. Examples of this include all the conventional applications of data networks described in Section 1.1.3 (assuming that a connection is set up before transferring data). This class includes the signaling used within the subnet for network control and establishment of connections. Finally, it includes very-low-rate stream-type data.

Class 4: Connectionless data. Examples include the conventional applications of data networks in cases where no connection is set up (*i.e.*, datagram traffic).

For classes 2, 3, and 4, the source data coming into the adaptation layer are in the form of frames or messages. The adaptation layer must break these frames into segments that fit into the ATM cells and provide enough extra information for the destination to be able to reconstruct the original frames. Along with this, the different classes (and different applications within the classes) have differing requirements for treating the problems of dropped or misdirected ATM cells. We describe the approach taken for class 3 traffic and then briefly describe the differences for the other classes.

Class 3 (connection-oriented) traffic The adaptation layer is split into two sublayers, called the convergence sublayer and the segmentation and reassembly sublayer. The first treats the user frames as units and is concerned with flow control and error recovery. The second treats the segments of a frame as units and is concerned with reassemblying the segments into frames. As usual, peer communication between nodes at each sublayer takes place via headers and trailers as illustrated in Fig. 2.57.

Current plans call for a two byte header and two byte trailer in the segmentation and reassembly sublayer. The header consists of the following fields:

Segment type (2 bits). This distinguishes whether a segment is the beginning segment of a convergence sublayer frame, an intermediate segment, a final segment, or a segment that contains an entire frame.

Sequence number (4 bits). This numbers segments within a convergence sublayer frame; this helps check on dropped or misdirected cells.

Reserved (10 bits). This might be used to multiplex multiple-user sessions on a single virtual circuit.

The trailer consists of the following fields:

Length indicator (6 bits). This tells how many bytes from the convergence sublayer frame are contained in the given segment. This is useful primarily for the last segment of a frame.

CRC (10 bits). The CRC polynomial is $D^{10} + D^9 + D^5 + D^4 + D + 1$. This is a primitive polynomial times $D+1$ and has the properties given in Section 2.3.4. It is used to check on the entire segment, including the header and the length indicator of the trailer.

It should be clear that the header and trailer above provide enough data to reconstruct the convergence sublayer frames from the received segments. The sequence

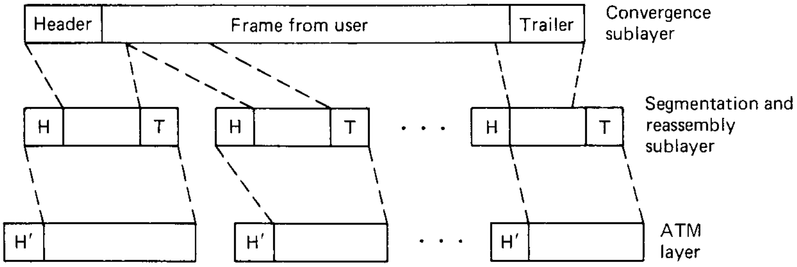


Figure 2.57 Headers and trailers at the convergence sublayer and segmentation and reassembly sublayer of the adaptation layer. Note that the user frame and its convergence sublayer header and trailer need not completely fill the data area at the segmentation and reassembly sublayer. The data plus header and trailer at the segmentation and reassembly sublayer must be exactly 48 bytes in length, however, to fit into the ATM data area.

number and CRC also provide some protection against bit errors and dropped or misdirected cells, but the approach is ad hoc.

The header and trailer for the convergence sublayer have not yet been defined. It is surprising that a CRC is used at the segmentation and reassembly sublayer rather than at the convergence sublayer. The longer block length at the convergence sublayer would have allowed greater error detection (or correction) power and higher efficiency. It would also have made it more convenient to use a CRC only when required by the user.

Class 4 (connectionless) traffic The headers and trailers at the segmentation and reassembly sublayer are expected to be the same for class 4 and class 3 traffic. The major new problem is that of connectionless traffic in a network where the cells are routed by virtual circuits. As discussed in Section 2.10.3, the most promising approach to this traffic is to view the routing as taking place at a higher layer than the adaptation layer; essentially this means viewing the datagram traffic as a higher-layer network using ATM virtual circuits as links (see Fig. 2.58). Thus, whereas connection-oriented traffic enters the adaptation layer only at origin and destination, connectionless traffic passes through the adaptation layer at each node used as a datagram switch. Between any two datagram switches, the traffic would use permanent virtual circuits assigned to carry all the connectionless traffic between the two switches. An interesting question is whether to view the switching of connectionless traffic as taking place in the convergence sublayer or strictly at a higher layer.

Class 1 and 2 traffic Class 1 traffic is constant-bit-rate traffic without any framing structure on the input to the adaptation layer. Thus a user data stream simply has to be broken into segments. The current plan is to use a one byte header and no trailer at the segmentation and reassembly sublayer (thus providing 47 bytes of user data per ATM cell). The header would consist of a four bit segment number plus a four bit CRC on the segment number. The reason for this is that occasional errors in this type

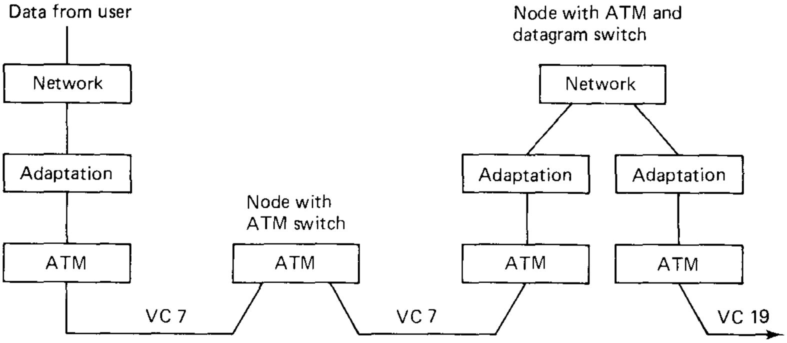


Figure 2.58 Datagram traffic can use an ATM network by being routed from one datagram switch to another. The virtual link between neighboring datagram switches is a virtual circuit in the ATM network dedicated to the aggregate of all the datagram traffic traveling between the two datagram switches.

of data are permissible, but to maintain framing, it is important to know when data have been dropped or misinserted, and the segment numbers provide this protection. Class 2 traffic has the same resilience to occasional errors as class 1 and the same need to know about dropped cells. At the same time, it has an incoming frame structure, requiring some of the features of the class 3 protocol for reassembling the frames.

2.10.3 Congestion

One would think that congestion could not be a problem in a network with the enormous link capacities of a broadband ISDN network. Unfortunately, this first impression is naive. Video and image applications can use large data rates, and marginal increases in quality will always appear to be possible by further increases in rate. Also, given a broadband network, there will be vested interests in ensuring that it is reasonably loaded, and reasonable loading will typically lead in time to heavy loading.

There are three mechanisms in broadband ISDN for limiting congestion. The first is for the network and user to agree to the required rate of the user at connection setup time. This agreement is expected to involve other quantities, such as the burstiness of the source and the quality of service required. The second mechanism is for the network to monitor each connection (within the convergence sublayer of the adaptation layer) to ensure that the user is complying with the agreed-upon rate and burstiness. Finally, the third mechanism is the priority bit in the ATM header (see Fig. 2.55). We explain briefly how these mechanisms work and interact. Chapter 6 develops the background required for a more thorough understanding of these mechanisms.

The ability to negotiate an agreed-upon rate between user and network is one of the primary reasons for a connection-oriented network. Without such negotiation, it would be impossible for the network to guarantee data rates and limited delay to users. It is easy to determine the effect of a new constant-bit-rate user, since such a session simply uses part of the available capacity on each link of the path without otherwise contributing to queueing delays. For class 2 and class 3 traffic, it is more difficult to determine the effect of a new session, since the burstiness of the traffic can cause queueing delays which can delay existing users as well as the user attempting to set up a connection. Relatively little is known about the interactions of delay between users of widely differing characteristics, and even less is known about the impact of different service disciplines at the switching nodes on these delays.

Since there is no negotiation of rates between the network and connectionless users, these users must be carefully controlled to avoid generating excessive delays for the connection-oriented sessions for which guaranteed rates and delays have been negotiated. This is why it is desirable to have a higher-layer network handling the connectionless traffic. This allows rates to be allocated for the aggregate of all connectionless traffic on given links, thus allowing statistical multiplexing between the connectionless users without having those users interfere with the connection-oriented traffic.

Along with negotiating rates when virtual circuits are established, it is necessary for the network to monitor and regulate the incoming traffic to ensure that it satisfies the agreed-upon limits. There are a number of mechanisms, such as the leaky bucket mechanism, for monitoring and regulating incoming traffic, and these are discussed in Chapter 6. The problem of interest here is what to do with traffic that exceeds the agreed-upon limits. The simplest approach is simply to discard such traffic, but this appears to be foolish when the network is lightly loaded. Another approach is to allow the excessive traffic into the network but to mark it so that it can be discarded later at any congested node.

The priority bit in the ATM header could be used to mark cells that exceed the negotiated rate limits of a user. This priority bit could also be set by a user to indicate lower-priority traffic that could be dropped by the network. Such a capability is useful in some video compression and voice compression schemes. It appears at first that using a single bit to carry all priority information in such a large and diverse network severely limits flexibility in applying flow control to the network. For example, suppose that the priority bit were used both by users to send relatively unimportant data and by the network to mark excessive data from a user. Then it would be possible for the network to be flooded by marked but unimportant video traffic; this could cause the network to discard important bursty traffic that was marked by the network because of exceeding the negotiated burstiness.

One possibility for an increased differentiation of priorities comes from the fact that individual users could negotiate priority while setting up a connection, and the subnet nodes could use these user priorities along with the priority bit in making discard decisions. It is also possible in principle to establish priorities concerning delay while setting up a connection, thus enabling the subnet switches to serve those virtual circuits requiring small delay before other circuits. Designing switches that are simple and yet account flexibly for both delay and discarding requirements is clearly a challenge, but the lack of information in the cell header does not seem to be a critical drawback.

Connectionless traffic poses a particular challenge to ATM networks. In the preceding section it was suggested that this traffic should be routed through a packet network sitting on top of the adaptation layer (or perhaps within the convergence sublayer). One benefit of this is that rate could be allocated to virtual circuits used as links in that higher-layer net. Assuming buffering in these higher-layer packet switches, the aggregate connectionless traffic could stay within its allocated limit and thus not interfere with any of the other classes of users. On the other hand, since the connectionless traffic is aggregated on these virtual circuits, individual connectionless users would obtain the benefits of statistical multiplexing. One could also adjust the rates of the virtual circuits over time to provide yet better service for connectionless traffic over periods of light loading from other classes.

SUMMARY

In this chapter, an introductory survey of the physical layer, a fairly complete treatment of the data link control layer, and a few closely related topics at other layers were presented. From a conceptual standpoint, one important issue has been distributed algorithms between two stations in the presence of errors and arbitrary delays. This has been developed in the context of ARQ, framing, and error recovery at higher layers. The correctness proof of the go back n class of protocols is important here since it shows how to reason carefully about such algorithms. There are many ad hoc algorithms used in data networks that contain subtle bugs, and it is important for designers to learn how to think about such algorithms in a more fundamental and structured way.

Another conceptual issue has been the encoding of control information. This was developed in the context of framing and session identification. Although the emphasis here was on efficiency of representation, the most important issue is to focus on the information actually conveyed; for example, it is important to recognize that flags with bit stuffing, length count fields, and special communication control characters all convey the same information but in quite different ways.

From a more practical standpoint, discussions were given on how modems work, how CRCs work (and fail), how ARQ works and how it can be made to work more efficiently with carefully chosen time-outs and control feedback, how various kinds of framing techniques work (and fail), how session identification can be accomplished, and how error recovery at different layers can be effected. Many existing networks have been used to illustrate these ideas.

Various standards have been discussed, but not in enough detail for one actually to implement them. Unfortunately, if one wishes to implement a protocol in conformance with a standard, one must read the documentation of the standard, which is long and tedious. The purpose here was to give some conceptual understanding of the standard so as to see what the standard accomplishes and how. There are many special-purpose networks in which standard protocols are inappropriate, and it is helpful to know why.

NOTES, SOURCES, AND SUGGESTED READING

Section 2.2. For more information on linear filters and their representation in the time and frequency domain, see any standard undergraduate text (*e.g.*, [Sie86]) on linear systems. For an extensive and up-to-date treatment of modulation and demodulation of digital signals, see [Pro83]. Also, [Qur85] provides a complete and beautifully organized treatment of adaptive equalization. For the information-theoretic aspects of this chapter (*i.e.*, Shannon's theorem, entropy, error detection and correction coding, and source coding), see [Gal68]. Finally, for an elementary treatment of the topics in this section oriented toward current practice, see [Sta85].

Section 2.3. Treatments of parity check codes and cyclic codes (CRC) can be found in [Bla83], [ClC81], and [Gal68].

Sections 2.4 and 2.5. A classic early paper on ARQ and framing is [Gra72]. An entire subdiscipline has developed to prove the correctness of point-to-point protocols. Typically (see, *e.g.*, [BoS82]) those approaches use a combination of exhaustive computer search plus analytic tools to reduce the search space. More information on highly efficient ARQ and framing is given in [Gal81]. Selective repeat and the advantages of sending packets multiple times after errors is discussed in [Wel82]. A more complete analysis of choice of frame lengths is given in [Alt86].

Sections 2.6 and 2.7. An excellent description of the standard DLCs is given in [Car80]. Initialization and disconnection are treated in [BaS83] and [Spi89].

Sections 2.8 and 2.9. TYMNET is discussed in [Rin76] and [Tym81]. The Codex networks are discussed in [HuS86]. See [Ryb80] for more discussion of the X.25 standard and [Com88] for discussion of TCP/IP.

Section 2.10. Basic tutorials on ATM and SONET are given in [Min89] and [BaC89], respectively. The CCITT and T1S1 working documents on ATM are surprisingly readable for an in-depth treatment.

PROBLEMS

2.1 Suppose that the output in Fig. 2.3(a) is $1 - e^{-2t/T}$ for $0 \leq t \leq T$, $(e^2 - 1)e^{-2t/T}$ for $t > T$, and zero for $t < 0$. Find the output in Fig. 2.3(b) using the linearity and time invariance of the filter.

2.2 Let $s(t) = 1$, for $0 \leq t \leq T$, and $s(t) = 0$ elsewhere. Let $h(t) = \alpha e^{-\alpha t}$ for $t \geq 0$ and $h(t) = 0$ for $t < 0$. Use the convolution equation to find the output when $s(t)$ is passed through a filter of impulse response $h(t)$.

2.3 Integrate Eq. (2.1) for $s(\tau) = e^{j2\pi f\tau}$ to get Eqs. (2.2) and (2.3). *Hint*: Transform the variable of integration by $\tau' = t - \tau$.

2.4 Suppose that a channel has the ideal low-pass frequency response $H(f) = 1$ for $-f_0 \leq f \leq f_0$ and $H(f) = 0$ elsewhere. Find the impulse response of the channel. [Use the inverse Fourier transform, Eq. (2.6).]

2.5 Let $s(t)$ be a given waveform and let $s_1(t) = s(\beta t)$ for some given positive β. Sketch the relationship between $s(t)$ and $s_1(t)$ for $\beta = 2$. Let $S(f)$ be the Fourier transform of $s(t)$. Find $S_1(f)$, the Fourier transform of $s_1(t)$, in terms of $S(f)$. Sketch the relationship for $\beta = 2$ [assume $S(f)$ real for your sketch].

2.6 Let $S(f)$ be the Fourier transform of $s(t)$.

(a) Show that the Fourier transform of $s(t)\cos(2\pi f_0 t)$ is $[S(f - f_0) + S(f + f_0)]/2$.

(b) Find the Fourier transform of $s(t)\cos^2(2\pi f_0 t)$ in terms of $S(f)$.

2.7 Suppose that the expected frame length on a link is 1000 bits and the standard deviation is 500 bits.

(a) Find the expected time to transmit a frame on a 9600-bps link and on a 50,000-bps link.

(b) Find the expected time and standard deviation of the time required to transmit 10^6 frames on each of the links above (assume that all frame lengths are statistically independent of each other).

(c) The *rate* at which frames can be transmitted is generally defined as the reciprocal of the expected transmission time. Using the result in part (b), discuss whether this definition is reasonable.

2.8 Show that the final parity check in a horizontal and vertical parity check code, if taken as the modulo 2 sum of all the data bits, is equal to the modulo 2 sum of the horizontal parity checks and also equal to the modulo 2 sum of the vertical parity checks.

2.9 **(a)** Find an example of a pattern of six errors that cannot be detected by the use of horizontal and vertical parity checks. *Hint*: Each row with errors and each column with errors will contain exactly two errors.

(b) Find the number of different patterns of four errors that will not be detected by a horizontal and vertical parity check code; assume that the array is K bits per row and J bits per column.

2.10 Suppose that a parity check code has minimum distance d. Define the distance between a code word and a received string (of the same length) as the number of bit positions in which the two are different. Show that if the distance between one code word and a given string is less than $d/2$, the distance between any other code word and the given string must exceed $d/2$. Show that if a decoder maps a given received string into a code word at smallest distance from the string, all combinations of fewer than $d/2$ errors will be corrected.

2.11 Consider a parity check code with three data bits and four parity checks. Suppose that three of the code words are 1001011, 0101101, and 0011110. Find the rule for generating each of the parity checks and find the set of all eight code words. What is the minimum distance of this code?

2.12 Let $g(D) = D^4 + D^2 + D + 1$, and let $s(D) = D^3 + D + 1$. Find the remainder when $D^4 s(D)$ is divided by $g(D)$, using modulo 2 arithmetic.

2.13 Let $g(D) = D^L + g_{L-1}D^{L-1} + \cdots + g_1 D + 1$ (assume that $L > 0$). Let $z(D)$ be a nonzero polynomial with highest and lowest order terms of degree j and i, respectively; that is, $z(D) = D^j + z_{j-1}D^{j-1} + \cdots + D^i$ [or $z(D) = D^j$, if $i = j$]. Show that $g(D)z(D)$ has at least two nonzero terms. *Hint*: Look at the coefficients of D^{L+j} and of D^i.

2.14 Show that if $g(D)$ contains the factor $1 + D$, then all error sequences with an odd number of errors are detected. *Hint*: Recall that a nonzero error polynomial $e(D)$ is detected unless $e(D) = g(D)z(D)$ for some polynomial $z(D)$. Look at what happens if 1 is substituted for D in this equation.

2.15 For a given generator polynomial $g(D)$ of degree L and given data length K, let $c^{(i)}(D) = c^{(i)}_{L-1}D^{L-1} + \cdots + c^{(i)}_1 D + c^{(i)}_0$ be the CRC resulting from the data string with a single 1 in position i [*i.e.*, $s(D) = D^i$ for $0 \le i \le K - 1$].

(a) For an arbitrary data polynomial $s(D)$, show that the CRC polynomial is $c(D) = \sum_{i=0}^{K-1} s_i c^{(i)}(D)$ (using modulo 2 arithmetic).

(b) Letting $c(D) = c_{L-1}D^{L-1} + \cdots + c_1 D + c_0$, show that

$$c_j = \sum_{i=0}^{K-1} s_i c^{(i)}_j; \qquad 0 \le j < L$$

This shows that each c_j is a parity check and that a cyclic redundancy check code is a parity check code.

2.16 **(a)** Consider a stop-and-wait ARQ strategy in which, rather than using a sequence number for successive packets, the sending DLC sends the number of times the given packet has been retransmitted. Thus, the format of the transmitted frames is $\boxed{j}\boxed{\text{packet}}\boxed{\text{CRC}}$, where j is 0 the first time a packet is transmitted, 1 on the first retransmission, and so on. The receiving DLC returns an ack or nak (without any request number) for each frame it receives. Show by example that this strategy does not work correctly no matter what rule the receiving DLC uses for accepting packets. (Use the assumptions preceding Section 2.4.1.)

(b) Redo part (a) with the following two changes: (1) the number j above is 0 on the initial transmission of a packet and 1 on *all* subsequent retransmissions of that packet, and (2) frames have a fixed delay in each direction, are always recognized as frames, and the sending DLC never times-out, simply waiting for an ack or either a nak or error.

2.17 **(a)** Let T_t be the expected transmission time for sending a data frame. Let T_f be the feedback transmisson time for sending an ack or nak frame. Let T_d be the expected propagation and processing delay in one direction (the same in each direction). Find the expected time T between successive frame transmissions in a stop-and-wait system (assume that there are no other delays and that no frames get lost).

(b) Let p_t be the probability of frame error in a data frame (independent of frame length), and p_f the probability of error in a feedback ack or nak frame. Find the probability q that a data packet is correctly received and acked on a given transmission. Show that $1/q$ is the expected number of times a packet must be transmitted for a stop-and-wait system (assume independent errors on all frames).

(c) Combining parts (a) and (b), find the expected time required per packet. Evaluate for $T_t = 1$, $T_f = T_d = 0.1$, and $p_t = p_f = 10^{-3}$.

2.18 Redraw Figs. 2.25 and 2.26 with the same frame lengths and the same set of frames in error, but focusing on the packets from B to A. That is, show SN and the window for node B, and show RN and packets out for node A (assume that all frames are carrying packets).

2.19 Give an example in which go back n will deadlock if receiving DLCs ignore the request number RN in each frame not carrying the awaited packet. *Hint*: Construct a pattern of delays leading to a situation where RN at node B is n plus SN_{min} at node A, and similarly for A and B reversed.

2.20 Give an example in which go back n ARQ fails if the modulus m is equal to n. For $m > n$, give an example where the right hand inequality in Eq. (2.24) is satisfied with equality and an example where the right hand inequality in Eq. (2.25) is satisfied with equality.

2.21 Assume that at time t, node A has a given SN_{min} as the smallest packet number not yet acknowledged and $SN_{max} - 1$ as the largest number yet sent. Let T_m be the minimum packet transmission time on a link and T_d be the propagation delay. Show that RN at node B must lie in the range from SN_{min} to SN_{max} for times in the range from $t - T_m - T_d$ to $t + T_m + T_d$.

2.22 **(a)** Let T_{min} be the minimum transmission time for data frames and T_d be the propagation and processing delay in each direction. Find the maximum allowable value T_{max} for frame transmission time such that a go back n ARQ system (of given n) will never have to go back or wait in the absence of transmission errors or lost frames. *Hint*: Look at Fig. 2.31.

(b) Redo part (a) with the added possibility that isolated errors can occur in the feedback direction.

2.23 Assume that frame transmission times τ are exponentially distributed with, for example, probability density $p(\tau) = e^{-\tau}$, for $\tau \geq 0$. Assume that a new frame always starts in each direction as soon as the previous one in that direction terminates. Assume that transmission times are all independent and that processing and propagation delays are negligible. Show that the probability q that a given frame is not acked before $n - 1$ additional frames have been sent (*i.e.*, the window is exhausted) is $q = (1 + n)2^{-n}$. *Hint*: Note that the set of times at which frame transmissions terminate at either node is a Poisson process and that a point in this process is equally likely to be a termination at A or B, with independence between successive terminations.

2.24 Show that if an isolated error in the feedback direction occurs on the ack of the given packet, then q in Problem 2.23 becomes $q = [2 + n + (n + 1)n/2]2^{-n-1}$.

2.25 Assume that frame transmission times τ have an Erlang distribution with probability density $p(\tau) = \tau e^{-\tau}$. Find the corresponding value of q in Problem 2.23. *Hint*: The termination times at A can be regarded as alternative points in a Poisson process and the same is true for B.

2.26 Let γ be the expected number of transmitted frames from A to B per successfully accepted packet at B. Let β be the expected number of transmitted frames from A to B between the transmission of a given frame and the reception of feedback about that frame (including the frame in transmission when the feedback arrives). Let p be the probability that a frame arriving at B contains errors (with successive frames assumed independent). Assume that A is always busy transmitting frames, that n is large enough that A never goes back in the absence of feedback, and that A always goes back on the next frame after hearing that the awaited frame contained errors. Show that γ satisfies $\gamma = 1 + p(\beta + \gamma)$. Define the efficiency η as $1/\gamma$, and find η as a function of β and p.

2.27 Assume that a selective repeat system has the property that the sending DLC always finds out whether a given transmitted packet was successfully received during or before the β^{th} frame transmission following the frame in which the given packet was transmitted. Give a careful demonstration that the sending DLC need never save more than $\beta + 1$ unacknowledged packets.

2.28 Find the efficiency of the ARPANET ARQ system under the assumptions that all eight virtual channels are always busy sending packets and that feedback always arrives about the fate of a given packet during or before the seventh frame transmission following the frame carrying the given packet.

2.29 Consider a generalized selective repeat ARQ system that operates in the following way: as in conventional go back n and selective repeat, let RN be the smallest numbered packet not yet correctly received, and let SN_{min} be the transmitter's estimate of RN. Let y_{top} be the largest-numbered packet correctly received and accepted (thus, for go back n, y_{top} would be $RN - 1$, whereas for conventional selective repeat, y_{top} could be as large as $RN + n - 1$). The rule at the transmitter is the same as for conventional go back n or selective repeat: The number z of the packet transmitted must satisfy

$$SN_{min} \leq z \leq SN_{min} + n - 1$$

The rule at the receiver, for some given positive number k, is that a correctly received packet with number z is accepted if

$$RN \leq z \leq y_{\text{top}} + k \tag{2.46}$$

(a) Assume initially that z (rather than $SN = z \bmod m$) is sent with each packet and that RN (rather than $RN \bmod m$) is sent with each packet in the reverse direction. Show that for each received packet at the receiver,

$$z \leq RN + n - 1$$

$$z \geq y_{\text{top}} - n + 1 \tag{2.47}$$

(b) Now assume that $SN = z \bmod m$ and RN is sent mod m. When z is received Eq. (2.47) is satisfied, but erroneous operation will result if $z - m$ or $z + m$ lie in the window specified by Eq. (2.46). How large need m be to guarantee that $z + m > y_{\text{top}} + k$ [*i.e.*, that $z + m$ is outside the window of Eq. (2.46)]?

(c) Is the value of m determined in part (b) large enough to ensure that $z - m < RN$?

(d) How large need m be (as a function of n and k) to ensure correct operation?

(e) Interpret the special cases $k = 1$ and $k = n$.

2.30 (a) Apply the bit stuffing rules of Section 2.5.2 to the following frame:
0 1 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 0

(b) Suppose that the following string of bits is received:
0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1
0 0 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0
Remove the stuffed bits and show where the actual flags are.

2.31 Suppose that the bit stuffing rule of Section 2.5.2 is modified to stuff a 0 only after the appearance of 01^5 in the original data. Carefully describe how the destuffing rule at the receiver must be modified to work with this change. Show how your rule would destuff the following string:
0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 1 0
If your destuffing rule is correct, you should remove only two 0's and find only one actual flag.

2.32 Bit stuffing must avoid the appearance of 01^6 within the transmitted frame, so we accept as given that an original string 01^6 will always be converted into $01^5 01$. Use this, plus the necessity to destuff correctly at the receiver, to show that a 0 must always be stuffed after 01^5. *Hint*: Consider the data string $01^5 01 x_1 x_2 \ldots$. If a 0 is not stuffed after 01^5, the receiver cannot distinguish this from $01^6 x_1 x_2 \ldots$ after stuffing, so stuffing is required in this case. Extend this argument to $01^5 0^k 1 x_1 x_2 \ldots$ for any $k > 1$.

2.33 Suppose that the string 0101 is used as the bit string to indicate the end of a frame and the bit stuffing rule is to insert a 0 after each appearance of 010 in the original data; thus, 010101 would be modified by stuffing to 01001001. In addition, if the frame proper ends in 01, a 0 would be stuffed after the first 0 in the actual terminating string 0101. Show how the string 11011010010101011101 would be modified by this rule. Describe the destuffing rule required at the receiver. How would the string 11010001001001100101 be destuffed?

2.34 Let $A = E\{K\}2^{-j} + j + 1$ be the upper bound on overhead in Eq. (2.34), and let $j = \lfloor \log_2 E\{K\} \rfloor$. Show that A satisfies

$$1.914\ldots + \log_2 E\{K\} \le A \le 2 + \log_2 E\{K\}$$

Find the analytic expression for 1.914.... *Hint*: Define γ as $\log_2 E\{K\} - j$ and find the minimum and maximum of $A - \log_2 E\{K\}$ as γ varies between 0 and 1.

2.35 Show that the probability that errors will cause a flag $01^6 0$ to appear within a transmitted frame is approximately $(1/32)Kp$, where K is the number of bits in the frame proper before stuffing and p is the probability of bit error. Assume that p is very small and that the probability of multiple errors in a frame is negligible. Assume that the bits of the original frame are IID (independent, identically distributed) with equal probability of 0 and 1. *Hint*: This is trickier than it appears, since the bits after stuffing are no longer IID. First, find the probability that a stuffed bit in error causes a flag (not an abort) to appear and use the analysis of Section 2.5.2 to approximate the expected number of stuffed bits as $K2^{-6}$. Then find the probability of a flag appearing due to errors in the original bits of the frame.

2.36 A certain virtual circuit network is designed for the case where all sessions carry heavy, almost steady traffic. It is decided to serve the various sessions using each given link in round-robin order, first sending one packet from the first session, then one from the second, and so on, up to the last, and then starting over with the first, thus avoiding the necessity of sending a session number with each packet. To handle the unlikely possibility that a session has no packet to send on a link at its turn, we place a k-bit flag, 01^{k-1}, in front of the information packet for the next session that has a packet. This flag is followed by a unary encoding of the number of sessions in order with nothing to send, 1 for one session, 01 for two, 001 for three, and so on. Thus, for example, if a packet for the second session has just been sent, sessions 3 and 4 have no packets, and session 5 has the packet $\overline{x}_5$, we send $01^{k-1}01\overline{x}_5$ as the next packet. In addition, an insertion is used if an information packet happens to start with 01^{k-2}. Otherwise, insertions are not required in information packets, and information packets carry their own length information.

(a) Assume that p is the probability that a session has no packet when it is that session's turn; this event is assumed independent of other sessions and of past history. Find the expected number of overhead bits per transmitted packet for flags, unary code, and insertions. Ignore, as negligible, any problems of what to do if no session has a packet to send.

(b) Suppose that the packets above, as modified by flags, unary code, and insertions, are the inputs to a data link control system using ARQ for error control and flags to indicate the end of packets. Explain if any problems arise due to the use of flags both on the DLC system and in the higher-level scheme for identifying packets with sessions.

2.37 Consider a stop-and-wait data link control protocol. Each transmitted frame contains the sequence number, modulo 2, of the contained packet. When a frame arrives at the receiver, the sequence number of the packet is compared with the packet number awaited by the receiver; if the numbers are the same modulo 2 and the CRC is satisfied, the packet is accepted and the awaited packet number is incremented. An ack containing the new awaited packet number modulo 2 is sent back to the transmitter. If a frame is received with correct CRC but the wrong sequence number, an ack (containing the awaited packet number modulo 2) is also sent back to the transmitter. The new twist here is that frames can go out of order on the channel. More precisely, there is a known maximum delay T on the channel. If a packet is transmitted at some time t, it is either not received at all or is received at an arbitrary time in the interval from t to $t + T$, independently of other transmissions. The return channel for the acks behaves the same way with the same maximum delay T. Assume that the receiver sends acks instantaneously upon receiving packets with valid CRCs.

(a) Describe rules for the transmitter to follow so as to ensure that each packet is eventually accepted, once and in order, by the receiver. You are not to put any extra protocol information in the packets nor to change the receiver's operation. You should try to leave as much freedom as possible to the transmitter's operation subject to the restriction of correct operation.

(b) Explain why it is impossible to achieve correct operation if there is no bound on delay.

2.38 Consider a stop-and-wait system with two-way traffic between nodes A and B. Assume that data frames are all of the same length and require D seconds for transmission. Acknowledgment frames require R seconds for transmission and there is a propagation delay P on the link. Assume that A and B both have an unending sequence of packets to send, assume that no transmission errors occur, and assume that the time-out interval at which a node resends a previously transmitted packet is very large. Finally, assume that each node sends new data packets and acknowledgments as fast as possible, subject to the rules of the stop-and-wait protocol.

(a) Show that the rate at which packets are transmitted in each direction is $(D+R+2P)^{-1}$. Show that this is true whether or not the starting time between A and B is synchronized.

(b) Assume, in addition, that whenever a node has both an acknowledgment and a data packet to send, the acknowledgment is piggybacked onto the data frame (which still requires D seconds). Assume that node B does not start transmitting data until the instant that it receives the first frame from A (*i.e.*, B's first frame contains a piggybacked ack). Show that all subsequent frames in each direction are data frames with a piggybacked ack. Show that the rate in each direction is now $(2D + 2P)^{-1}$. Note that if $D > R$, this rate is less than that in part (a).

2.39 **(a)** Consider a data link control using fixed-length packets of K bits each. Each message is broken into packets using fill in the last packet of the message as required; for example, a message of 150 bits is broken into two packets of 100 bits each, using 50 bits of fill. Modify Eq. (2.42) to include fill for fixed-length packets.

(b) Find $E\{TC\}$ using the approximation $E\{\lceil M/K \rceil\} \approx E\{M/K\} + 1/2$; find the value of K that minimizes this approximation to $E\{TC\}$.

(c) Note that for $j = 1$ the value of K_{max} in Eq. (2.44) is infinite, whereas the minimizing fixed length K is finite. Explain why this is reasonable.

2.40 **(a)** Assume that nodes A and B both regard the link between them as up according to the balanced initialization and disconnect protocol of Section 2.7.3. Assume that at time t node A starts to disconnect (*i.e.*, transmits DISC). Show, using the same assumptions as in Section 2.4, that each node eventually regards the link as down.

(b) Give an example for the situation in part (a), where node A regards the link as down and then starts to initialize the link before node B regards the link as down; check your proof for part (a) in light of this example. Show that B regards the link as down before A again regards the link as up.

(c) Now suppose that node A regards the link as up and node B is initializing the link but has not yet received ACKI from A. Suppose that A starts to disconnect. Show that B eventually regards the link as up and then starts to disconnect and that after that A and B both eventually regard the link as down.

2.41 **(a)** Create an example of incorrect operation, analogous to that of Fig. 2.46, for the master–slave initialization protocol under the assumption that a node, after recovering from a node failure, starts in the up state and initializes by first sending DISC and then INIT. *Hint:* Use the same structure as Fig. 2.46; that is, assume that after each failure, node A receives the messages from B that were generated by B in response to the messages in the earlier period at A.

(b) Now suppose that an arbitrary protocol is being used for initialization. It is known that node A, on recovery from a failure, will send message x. Node B, on recovering from failure and receiving message x, will send message y, and node A, upon receiving y under these circumstances, will be correctly initialized. Construct a sequence of failures and delays such that the protocol fails, assuming only that the nodes operate as above under the above conditions. *Hint:* Follow the hint in part (a), but also assume a sequence of failures at B, with B receiving responses in one period to messages in the preceding period.

2.42 The balanced initialization protocol of Section 2.7.3 piggybacks acks on each INIT or DISC message. Explain why it is a bad idea to include also either an INIT or DISC message every time that an ACKI or ACKD must be sent.

2.43 Consider a datagram network and assume that M is the maximum number of packets that can be sent by a session while a given packet (in either direction) still exists within the network. Assume that selective repeat ARQ is used for the session with a window size of n. Show that the modulus m must satisfy

$$m \geq 2n + M$$