Chapter 5: Link Layer and Local Area Networks

In this chapter we examine the data link layer--its services, the principles underlying its operation, and a number of important data link layer protocols. We learn that the basic service of the data link layer is to move a network-layer datagram from one node (host or router) to an adjacent node. We investigate the different services a link layer protocol can provide in addition to this basic service, including link access services, delivery services, flow control services and transmission services. These differences are due in part to a wide variety of link types over which data link protocols must operate. We examine error detection and correction, services that are often present in link-layer protocols. We investigate multiple access protocols, commonly used in LANs (local area networks). We explore LAN addresses and how ARP (Address Resolution Protocol) is used to translate between LAN and IP addresses. We then study in some depth Ethernet, today's most prevalent LAN technology. We investigate hubs, bridges and layer-2 switches, and compare their features. Additionally, we cover wireless LANs and the IEEE 802.11 standard, PPP, ATM, X.25 and frame relay in this chapter.

Online Book

5.1: The Data Link Layer: Introduction, Services

In the previous chapter we learned that the network layer provides a communication service between two hosts. As shown in Figure 5.1, this communication path starts at the source host, passes through a series of routers, and ends at the destination host. We'll find it convenient here to refer to the hosts and the routers simply as **nodes** (since, as we'll see shortly, we will not be particularly concerned whether a node is a router or a host), and to the communication channels that connect adjacent nodes along the communication path as **links.** In order to move a datagram from source host to destination host, the datagram must be moved over each of the *individual links* in the path. In this chapter, we focus on the **data-link layer,** which is responsible for transferring a datagram across an individual link. We'll first identify and study the services provided by the link layer. In Sections 5.2 through 5.4, we'll then examine important principles behind the protocols that provide these services (including the topics of error detection and correction, so-called multiple access protocols that are used share a single physical link among multiple nodes, and link-level addressing). We'll see that many different types of link-level technology can be used to connect two nodes. In Sections 5.5 through 5.10, we'll examine specific link-level architectures and protocols in more detail.

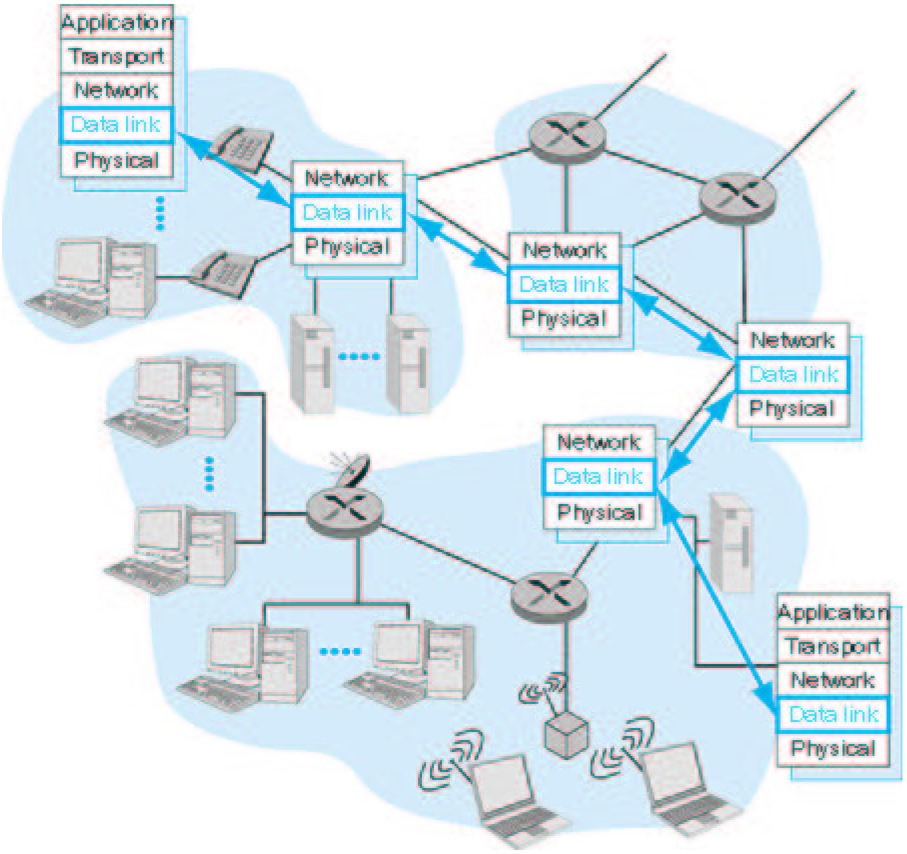


Figure 5.1: The data-link layer

5.1.1: The Services Provided by the Link Layer

A link-layer protocol is used to move a datagram over an individual link. The **link-layer protocol** defines the format of the units of data exchanged between the nodes at the ends of the link, as well as the actions taken by these nodes when sending and receiving these data units. Recall from Chapter 1 that the units of data exchanged by a link-layer protocol are called **frames,** and that each link-layer frame typically encapsulates one network-layer datagram. As we shall see shortly, the actions taken by a link-layer protocol when sending and receiving frames include error detection, retransmission, flow control, and random access. Examples of link-layer protocols include Ethernet, token ring, FDDI, and PPP; in many contexts, ATM and frame relay can be considered link-layer protocols as well. We'll cover these protocols in detail in the latter half of this chapter. Whereas the network layer has the end-to-end job of moving transport-layer segments from the source host to the destination host, a link-layer protocol has the node-to-node job of moving a network-layer datagram over a *single link* in the path. An important characteristic of the link layer is that a datagram may be handled by different link-layer protocols on the different links in the path. For example, a datagram may be handled by Ethernet on the first link, PPP on the last link, and frame relay on all intermediate links. It is important to note that the services provided by the different link-layer protocols may be different. For example, a link-layer protocol may or may not provide reliable delivery. Thus, the network layer must be able to accomplish its end-to-end job in the face of a varying set of individual link-layer services.

In order to gain insight to the link layer and how it relates to the network layer, let's consider a transportation analogy. Consider a travel agent who is planning a trip for a tourist traveling from Princeton, New Jersey, to Lausanne, Switzerland. Suppose the travel agent decides that it is most convenient for the tourist to take a limousine from Princeton to JFK airport, then a plane from JFK airport to Geneva airport, and finally a train from Geneva's airport to Lausanne's train station. Once the travel agent makes the three reservations, it is the responsibility of the Princeton limousine company to get the tourist from Princeton to JFK; it is the responsibility of the airline company to get the tourist from JFK to Geneva; and it is the responsibility of the Swiss train service to get the tourist from Geneva to Lausanne. Each of the three segments of the trip is "direct" between two "adjacent" locations. Note that the three transportation segments are managed by different companies and use entirely different transportation modes (limousine, plane, and train). Although the transportation modes are different, they each provide the basic service of moving passengers from one location to an adjacent location. In this transportation analogy, the tourist is analogous to a datagram, each transportation segment is analogous to a communication link, the transportation mode is analogous to the link-layer protocol, and the travel agent who plans the trip is analogous to a routing protocol.

Although the basic service of any link layer is to "move" a datagram from one node to an adjacent node over a single communication link, the details of service will depend on the specific link-layer protocol that is employed over the link. Possible services that can be offered by a link-layer protocol include:

- *Framing and link access.* Almost all link-layer protocols encapsulate each network-layer datagram within a link-layer frame before transmission onto the link. A frame consists of a data field, in which the network-layer datagram is inserted, and a number of header fields. (A frame may also include trailer fields; however, we will refer to both header and trailer fields as header fields.) A data-link protocol specifies the structure of the frame, as well as a channel access protocol that specifies the rules by which a frame is transmitted onto the link. For point-to-point links that have a single sender on one end of the link and a single receiver at the other end of the link, the link-access protocol is simple (or non-existent)--the sender can send a frame whenever the link is idle. The more interesting case is when multiple nodes share a single broadcast link--the so-called multiple access problem. Here, the channel access protocol serves to coordinate the frame transmissions of the many nodes; we cover multiple access protocols in detail in Section 5.3. We'll see several different frame formats when we examine specific link-layer protocols in the second half of this chapter. In Section 5.3, we'll see that frame headers also often include fields for a node's so-called **physical address,** which is completely *distinct* from the node's network layer (for example, IP) address.

- *Reliable delivery.* When a link-layer protocol provides reliable-delivery service, it guarantees to move each network-layer datagram across the link without error. Recall that certain transport-layer protocols (such as TCP) also provide a reliable-delivery service. Similar to a transport-layer reliable-delivery service, a link-layer reliable-delivery service is achieved with acknowledgments and retransmissions (see Section 3.4). A link-layer reliable-delivery service is often used for links that are prone to high error rates, such as a wireless link, with the goal of correcting an error locally, on the link where the error occurs, rather than forcing an end-to-end retransmission of the data by a transport- or application-layer protocol. However, link-layer reliable delivery can be considered an unnecessary overhead for low bit-error links, including fiber, coax, and many twisted-pair copper links. For this reason, many of the most popular link-layer protocols do not provide a reliable-delivery service.

- *Flow control.* The nodes on each side of a link have a limited amount of frame buffering capacity. This is a potential problem, as a receiving node may receive frames at a rate faster than it can process the frames over some time interval. Without flow control, the receiver's buffer can overflow and frames can get lost. Similar to the transport layer, a link-layer protocol can provide flow control in order to prevent the sending node on one side of a link from overwhelming the receiving node on the other side of the link.

- *Error detection.* A node's receiver can incorrectly decide that a bit in a frame is zero when it was transmitted as a one, and vice versa. Such bit errors are introduced by signal attenuation and electromagnetic noise. Because there is no need to forward a datagram that has an error, many link-layer protocols provide a mechanism to detect the presence of one or more errors. This is done by having the transmitting node set error-detection bits in the frame, and having the receiving node perform an error check. Error detection is a very common service among link-layer protocols. Recall from Chapters 3 and 4 that the transport layer and network layers in the Internet also provide a limited form of error detection. Error detection in the link layer is usually more sophisticated and implemented in hardware.

- *Error correction.* Error correction is similar to error detection, except that a receiver cannot only detect whether errors have been introduced in the frame but can also determine exactly where in the frame the errors have occurred (and hence correct these errors). Some protocols (such as ATM) provide link-layer error correction for the packet header rather than for the entire packet. We cover error detection and correction in Section 5.2.

- *Half-duplex and full-duplex.* With full-duplex transmission, the nodes at both ends of a link may transmit packets at the same time. With half-duplex transmission, a node cannot both transmit and receive at the same time.

As noted above, many of the services provided by the link layer have strong parallels with services provided at the transport layer. For example, both the link layer and the transport layer can provide reliable delivery. Although the mechanisms used to provide reliable delivery in the two layers are similar (see Section 3.4), the two reliable delivery services are not the same. A transport protocol provides reliable delivery between two processes on an end-to-end basis; a reliable link-layer protocol provides the reliable-delivery service between two nodes connected by a single link. Similarly, both link-layer and transport-layer protocols can provide flow control and error detection; again, flow control in a transport-layer protocol is provided on an end-to-end basis, whereas it is provided in a link-layer protocol on a node-to-adjacent-node basis.

5.1.2: Adapters Communicating

For a given communication link, the link-layer protocol is, for the most part, implemented in an **adapter.** An adapter is a board (or a PCMCIA card) that typically contains RAM, DSP chips, a host bus interface, and a link interface. Adapters are also commonly known as **network interface cards** or **NIC**s. As shown in Figure 5.2, the network layer in the transmitting node (that is, a host or router) passes a network-layer datagram to the adapter that handles the sending side of the communication link. The adapter encapsulates the datagram in a frame and then transmits the frame into the communication link. At the other side, the receiving adapter receives the entire frame, extracts the network-layer datagram, and passes it to the network layer. If the link-layer protocol provides error detection, then it is the sending adapter that sets the error detection bits and it is the receiving adapter that performs error checking. If the link-layer protocol provides reliable delivery, then the mechanisms for reliable delivery (for example, sequence numbers, timers, and acknowledgments) are entirely implemented in the adapters. If the link-layer protocol provides random access (see Section 5.3), then the random access protocol is entirely implemented in the adapters.

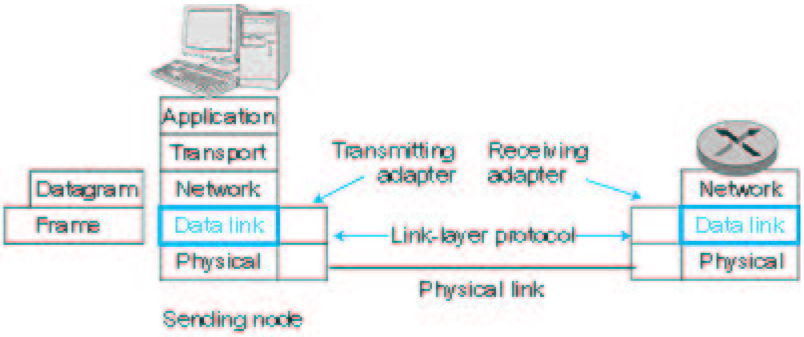


Figure 5.2: The link-layer protocol for a communication link is implemented in the adapters at the two ends of the link

An adapter is a semi-autonomous unit. For example, an adapter can receive a frame, determine if a frame is in error and discard the frame without notifying its "parent" node. An adapter that receives a frame only interrupts its parent node when it wants to pass a network-layer datagram up the protocol stack. Similarly, when a node passes a datagram down the protocol stack to an adapter, the node fully delegates to the adapter the task of transmitting the datagram across that link. On the other hand, an adapter is not a completely autonomous unit. Although we have shown the adapter as a separate "box" in Figure 5.3, the adapter is typically housed in the same physical box as the rest of the node, shares power and busses with the rest of the node, and is ultimately under the control of the node.

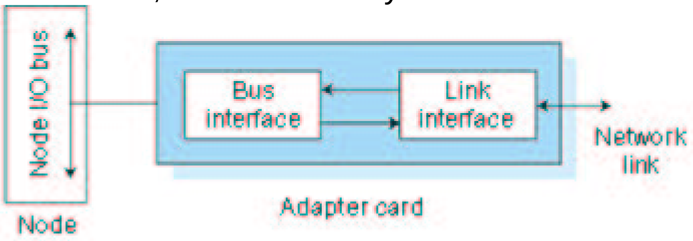


Figure 5.3: The adapter is a semi-autonomous unit

As shown in Figure 5.3, the main components of an adapter are the bus interface and the link interface. The bus interface is responsible for communicating with the adapter's parent node. It transfers data and control information between the node and the NIC. The link interface is responsible for implementing the link-layer protocol. In addition to framing and de-framing datagrams, it may provide error detection, random access, and other link-layer functions. It also includes the transmit and receive circuitry. For popular link-layer technologies, such as Ethernet, the link interface is implemented by chip set that can be bought on the commodity market. For this reason, Ethernet adapters are incredibly cheap--often less than $30 for 10 Mbps and 100 Mbps transmission rates.

Adapter design has become very sophisticated over the years. One of the critical issues in adapter performance has always been whether the adapter can move data in and out of a node at the full line speed, that is, at the transmission rate of the link. You can learn more about adapter architecture for 10 Mbps Ethernet, 100 Mbps Ethernet, and 155 Mbps ATM by visiting the 3Com adapter page [3Com 1999]. *Data Communications* magazine provides a nice introduction to Gbps Ethernet adapters [GigaAdapter 2000].

Online Book

5.2: Error Detection and Correction Techniques

In the previous section, we noted that **bit-level error detection and correction**--detecting and correcting the corruption of bits in a data-link-layer frame sent from one node to another physically connected neighboring node--are two services often provided by the data-link layer. We saw in Chapter 3 that error detection and correction services are also often offered at the transport layer as well. In this section, we'll examine a few of the simplest techniques that can be used to detect and, in some cases, correct such bit errors. A full treatment of the theory and implementation of this topic is itself the topic of many textbooks (for example, [Schwartz 1980]), and our treatment here is necessarily brief. Our goal here is to develop an intuitive feel for the capabilities that error detection and correction techniques provide, and to see how a few simple techniques work and are used in practice in the data link layer.

Figure 5.4 illustrates the setting for our study. At the sending node, data, *D*, to be protected against bit errors is augmented with error-detection and correction bits, *EDC.* Typically, the data to be protected includes not only the datagram passed down from the network layer for transmission across the link, but also link-level addressing information, sequence numbers, and other fields in the data-link frame header. Both *D* and *EDC* are sent to the receiving node in a link-level frame. At the receiving node, a sequence of bits, *D*' and *EDC*' are received. Note that *D*' and *EDC*' may differ from the original *D* and *EDC* as a result of in-transit bit flips.

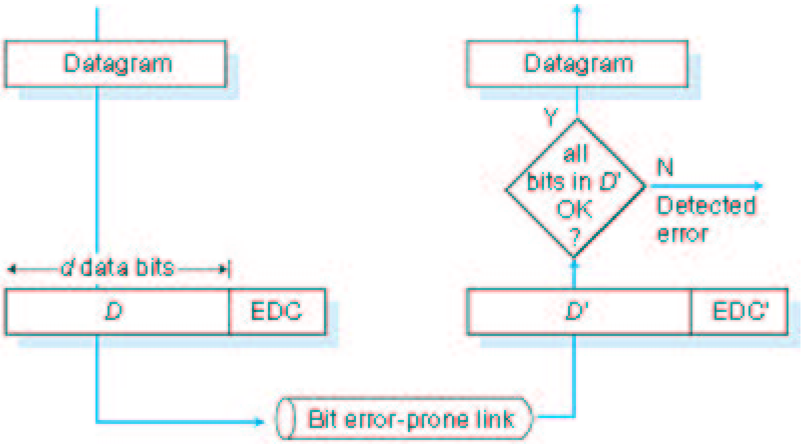


Figure 5.4: Error detection and correction scenario

The receiver's challenge is to determine whether or not *D*' is the same as the original *D*, given that it has only received *D*' and *EDC*'. The exact wording of the receiver's decision in Figure 5.4 (we ask whether an error is detected, not whether an error has occurred!) is important. Error-detection and correction techniques allow the receiver to sometimes, *but not always,* detect that bit errors have occurred. That is, even with the use of error-detection bits there will still be a possibility that **undetected bit errors** will occur, that is, that the receiver will be unaware that the received information contains bit errors. As a consequence, the receiver might deliver a corrupted datagram to the network layer, or be unaware that the contents of some other field in the frame's header have been corrupted. We thus want to choose an error-detection scheme so that the probability of such occurrences is small. Generally, more sophisticated error-detection and correction techniques (that is, those that have a smaller probability of allowing undetected bit errors) incur a larger overhead--more computation is needed to compute and transmit a larger number of error-detection and correction bits.

Let's now examine three techniques for detecting errors in the transmitted data--parity checks (to illustrate the basic ideas behind error detection and correction), checksumming methods (which are more typically employed in the transport layer) and cyclic redundancy checks (which are typically employed in the data-link layer).

5.2.1: Parity Checks

Perhaps the simplest form of error detection is the use of a single **parity bit.** Suppose that the information to be sent, *D* in Figure 5.4, has *d* bits. In an even parity scheme, the sender simply includes one additional bit and chooses its value such that the total number of 1s in the *d* + 1 bits (the original information plus a parity bit) is even. For odd parity schemes, the parity bit value is chosen such that there are an odd number of 1s. Figure 5.5 illustrates an even parity scheme, with the single parity bit being stored in a separate field.

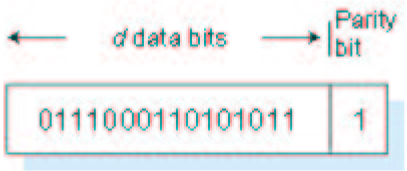


Figure 5.5: One-bit even parity

Receiver operation is also simple with a single parity bit. The receiver need only count the number of 1s in the received *d* + 1 bits. If an odd number of 1-valued bits are found with an even parity scheme, the receiver knows that at least one bit error has occurred. More precisely, it knows that some *odd* number of bit errors have occurred.

But what happens if an even number of bit errors occur? You should convince yourself that this would result in an undetected error. If the probability of bit errors is small and errors can be assumed to occur independently from one bit to the next, the probability of multiple bit errors in a packet would be extremely small. In this case, a single parity bit might suffice. However, measurements have shown that rather than occurring independently, errors are often clustered together in "bursts." Under burst error conditions, the probability of undetected errors in a frame protected by single-bit-parity can approach 50 percent [Spragins 1991]. Clearly, a more robust error-detection scheme is needed (and, fortunately, is used in practice!). But before examining error-detection schemes that are used in practice, let’s consider a simple generalization of one-bit parity that will provide us with insight into error-correction techniques.

Figure 5.6 shows a two-dimensional generalization of the single-bit parity scheme. Here, the *d* bits in *D* are divided into *i* rows and *j* columns. A parity value is computed for each row and for each column. The resulting *i* + *j* + 1 parity bits comprise the data-link frame’s error-detection bits.

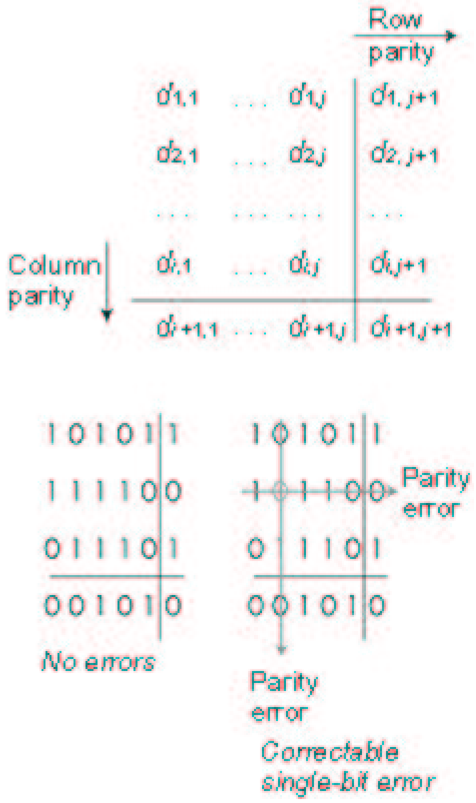

Row parity
$d_{1,1}$ . . . $d_{1,j}$ | $d_{1,j+1}$
$d_{2,1}$ . . . $d_{2,j}$ | $d_{2,j+1}$
. . . . | . . .
Column parity
$d_{i,1}$ . . . $d_{i,j}$ | $d_{i,j+1}$
$d_{i+1,1}$ . . . $d_{i+1,j}$ | $d_{i+1,j+1}$
1 0 1 0 1 | 1
1 1 1 1 0 | 0
0 1 1 1 0 | 1
0 0 1 0 1 | 0
No errors
1 0 1 0 1 | 1
1 0 1 1 0 | 0
0 1 1 1 0 | 1
0 0 1 0 1 | 0
Parity error
Parity error
Correctable single-bit error

Figure 5.6: Two-dimensional even parity

Suppose now that a single bit error occurs in the original *d* bits of information. With this **two-dimensional parity** scheme, the parity of both the column and the row containing the flipped bit will be in error. The receiver can thus not only *detect* the fact that a single bit error has occurred, but can use the column and row indices of the column and row with parity errors to actually identify the bit that was corrupted and *correct* that error! Figure 5.6 shows an example in which the 1-valued bit in position (2,2) is corrupted and switched to a 0--an error that is both detectable and correctable at the receiver. Although our discussion has focused on the original *d* bits of information, a single error in the parity bits themselves is also detectable and correctable. Two-dimensional parity can also detect (but not correct!) any combination of two errors in a packet. Other properties of the two-dimensional parity scheme are explored in the problems at the end of the chapter.

The ability of the receiver to both detect and correct errors is known as **forward error correction (FEC).** These techniques are commonly used in audio storage and playback devices such as audio CDs. In a network setting, FEC techniques can be used by themselves, or in conjunction with the ARQ techniques we examined in Chapter 3. FEC techniques are valuable because they can decrease the number of sender retransmissions required. Perhaps more importantly, they allow for immediate correction of errors at the receiver. This avoids having to wait for the round-trip propagation delay needed for the sender to receive a NAK packet and for the retransmitted packet to propagate back to the receiver--a potentially important advantage for real-time network applications [Rubenstein 1998]. Recent work examining the use of FEC in error-control protocols include [Biersack 1992; Nonnenmacher 1998; Byers 1998; Shacham 1990].

5.2.2: Checksumming Methods

In checksumming techniques, the *d* bits of data in Figure 5.4 are treated as a sequence of *k*-bit integers. One simple checksumming method is to simply sum these *k*-bit integers and use the resulting sum as the error detection bits. The so-called **Internet checksum** is based on this approach--bytes of data are treated as 16-bit integers and their ones-complement sum forms the Internet checksum. As discussed in Section 3.3.2, the receiver calculates the checksum over the received data and checks whether it matches the checksum carried in the received packet. RFC 1071 discusses the Internet checksum algorithm and its implementation in detail. In the TCP/IP protocols, the Internet checksum is computed over all fields (header and data fields included). In other protocols, for example, XTP [Strayer 1992], one checksum is computed over the header, with another checksum computed over the entire packet. McAuley [McAuley 1994] describes improved weighted checksum codes that are suitable for high-speed software implementation and Feldmeier [Feldmeier 1995] presents fast software implementation techniques for not only weighted check sum codes, but CRC (see below) and other codes as well.

5.2.3: Cyclic Redundancy Check (CRC)

An error-detection technique used widely in today’s computer networks is based on **cyclic redundancy check (CRC) codes.** CRC codes are also known as **polynomial codes,** since it is possible to view the bit string to be sent as a polynomial whose coefficients are the 0 and 1 values in the bit string, with operations on the bit string interpreted as polynomial arithmetic. CRC codes operate as follows. Consider the *d*-bit piece of data, *D*, that the sending node wants to send to the receiving node. The sender and receiver must first agree on an *r* + 1 bit pattern, known as a **generator,** that we will denote as *G*. We will require that the most significant (leftmost) bit of *G* be a 1. The key idea behind CRC codes is shown in Figure 5.7. For a given piece of data, *D*, the sender will choose *r* additional bits, *R*, and append them to *D* such that the resulting *d* + *r* bit pattern (interpreted as a binary number) is exactly divisible by *G* using modulo 2 arithmetic. The process of error checking with CRCs is thus simple: The receiver divides the *d* + *r* received bits by *G*. If the remainder is nonzero, the receiver knows that an error has occurred; otherwise the data is accepted as being correct.

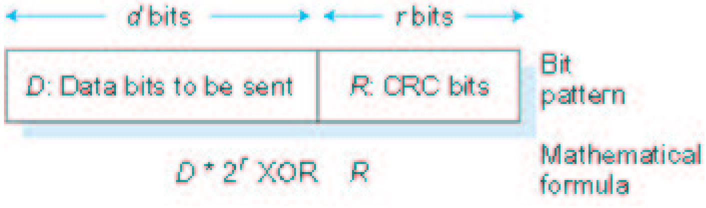


Figure 5.7: CRC codes

All CRC calculations are done in modulo 2 arithmetic without carries in addition or borrows in subtraction. This means that addition and subtraction are identical, and both are equivalent to the bitwise exclusive-or (XOR) of the operands. Thus, for example,

1011 XOR 0101 = 1110

1001 XOR 1101 = 0100

Also, we similarly have

1011 – 0101 = 1110

1001 – 1101 = 0100

Multiplication and division are the same as in base-2 arithmetic, except that any required addition or subtraction is done without carries or borrows. As in regular binary arithmetic, multiplication by 2^k left shifts a bit pattern by k places. Thus, given D and R, the quantity $D \bullet 2^r$ XOR R yields the $d + r$ bit pattern shown in Figure 5.7. We'll use this algebraic characterization of the $d + r$ bit pattern from Figure 5.7 in our discussion below.

Let us now turn to the crucial question of how the sender computes R. Recall that we want to find R such that there is an n such that

$$D \bullet 2^r \text{ XOR } R = nG$$

That is, we want to choose R such that G divides into $D \bullet 2^r$ XOR R without remainder. If we exclusive-or (that is, add modulo 2, without carry) R to both sides of the above equation, we get

$$D \bullet 2^r = nG \text{ XOR } R$$

This equation tells us that if we divide $D \bullet 2^r$ by G, the value of the remainder is precisely R. In other words, we can calculate R as

$$R = \text{remainder } \frac{D \cdot 2^r}{G}$$

Figure 5.8 illustrates this calculation for the case of D = 101110, d = 6 and G = 1001, r = 3. The nine bits transmitted in this case are 101110 011. You should check these calculations for yourself and also check that indeed $D \bullet 2^r$ = 101011 • G XOR R.

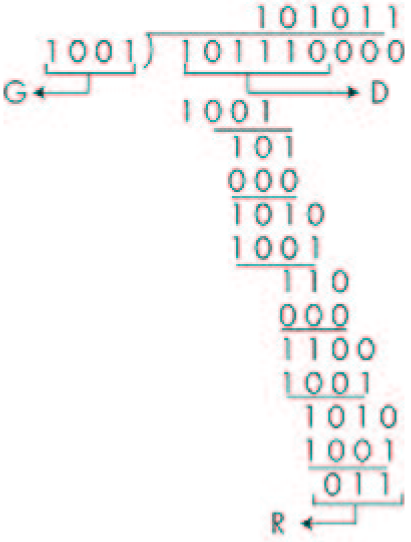


Figure 5.8: An example CRC calculation

International standards have been defined for 8-, 12-, 16- and 32-bit generators, *G*. An 8-bit CRC is used to protect the 5-byte header in ATM cells. The CRC-32 32-bit standard, which has been adopted in a number of link-level IEEE protocols, uses a generator of

$$G_{CRC\text{-}32} = 100000100110000010001110110110111$$

Each of the CRC standards can detect burst errors of less than $r + 1$ bits and any odd number of bit errors. Furthermore, under appropriate assumptions, a burst of length greater than $r + 1$ bits is detected with probability $1 - 0.5^r$. The theory behind CRC codes and even more powerful codes is beyond the scope of this text. The text [Schwartz 1980] provides an excellent introduction to this topic.

Online Book

5.3: Multiple Access Protocols and LANs

In the introduction to this chapter, we noted that there are two types of network links: point-to-point links, and broadcast links. A **point-to-point link** consists of a single sender on one end of the link, and a single receiver at the other end of the link. Many link-layer protocols have been designed for point-to-point links; PPP (the point-to-point protocol) and HDLC are two such protocols that we’ll cover later in this chapter. The second type of link, a **broadcast link,** can have multiple sending and receiving nodes all connected to the same, single, shared broadcast channel. The term *broadcast* is used here because when any one node transmits a frame, the channel broadcasts the frame and each of the other nodes receives a copy. Ethernet is probably the most widely deployed broadcast link technology; we’ll cover Ethernet in detail in Section 5.5. In this section we’ll take a step back from specific link-layer protocols and first examine a problem of central importance to the data-link layer: how to coordinate the access of multiple sending and receiving nodes to a shared broadcast channel--the so-called **multiple access problem.** Broadcast channels are often used in **local area networks (LANs),** networks that are geographically concentrated in a single building (or on a corporate or university campus). Thus, we’ll also look at how multiple access channels are used in LANs at the end of this section.

We are all familiar with the notion of broadcasting, as television has been using it since its invention. But traditional television is a one-way broadcast (that is, one fixed node transmitting to many receiving nodes), while nodes on a computer network broadcast channel can both send and receive. Perhaps a more apt human analogy for a broadcast channel is a cocktail party, where many people gather together in a large room (the air providing the broadcast medium) to talk and listen. A second good analogy is something many readers will be familiar with--a classroom--where teacher(s) and student(s) similarly share the same, single, broadcast medium. A central problem in both scenarios is that of determining who gets to talk (that is, transmit into the channel), and when. As humans, we’ve evolved an elaborate set of protocols for sharing the broadcast channel:

> "Give everyone a chance to speak."
> "Don’t speak until you are spoken to."
> "Don’t monopolize the conversation."
> "Raise your hand if you have a question."
> "Don’t interrupt when someone is speaking."
> "Don’t fall asleep when someone else is talking."

Computer networks similarly have protocols--so-called **multiple access protocols**--by which nodes regulate their transmission onto the shared broadcast channel. As shown in Figure 5.9, multiple access protocols are needed in a wide variety of network settings, including both wired and wireless local area networks, and satellite networks. Figure 5.10 takes a more abstract view of the broadcast channel and of the nodes sharing that channel. Although technically each node accesses the broadcast channel through its adapter, in this section we will refer to the *node* as the sending and receiving device. In practice, hundreds or even thousands of nodes can directly communicate over a broadcast channel.

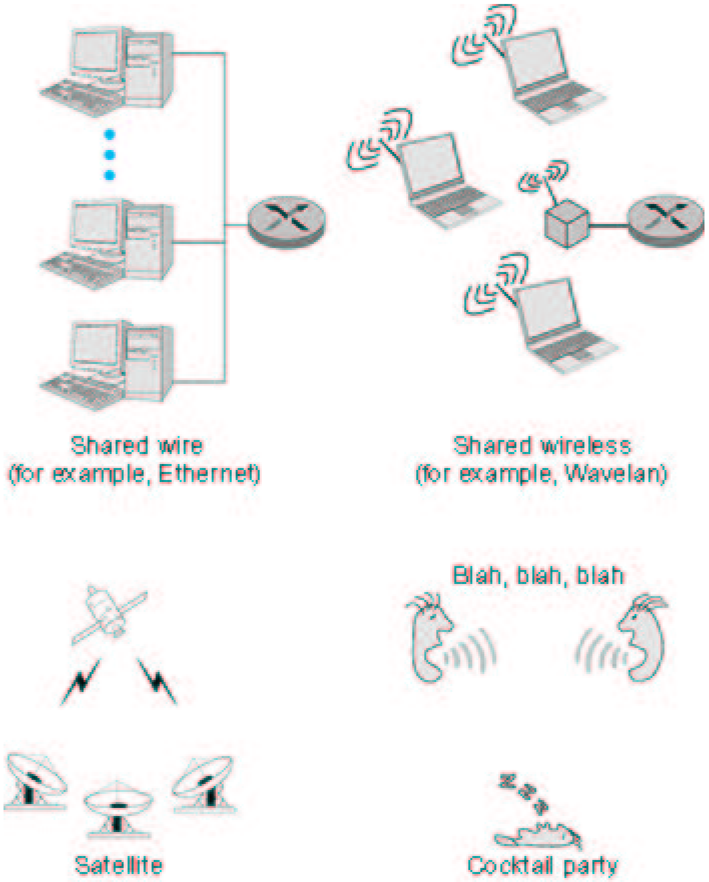


Figure 5.9: Various multiple access channels

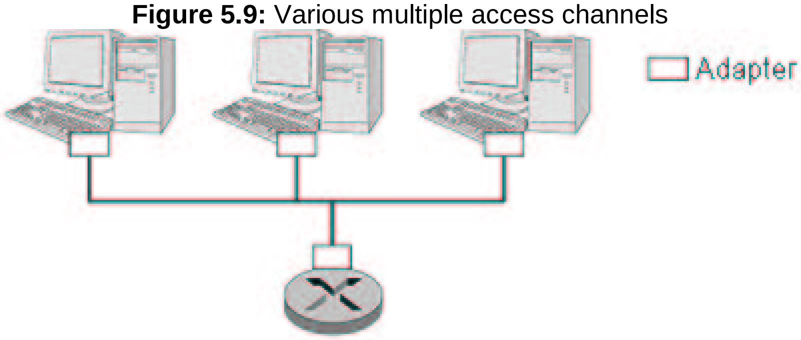


Figure 5.10: A broadcast channel interconnecting four nodes

Because all nodes are capable of transmitting frames, more than two nodes can transmit frames at the same time. When this happens, all of the nodes receive multiple frames at the same time, that is, the transmitted frames **collide** at all of the receivers. Typically, when there is a collision, none of the receiving nodes can make any sense of any of the frames that were transmitted; in a sense, the signals of the colliding frame become inextricably tangled together. Thus, all the frames involved in the collision are lost, and the broadcast channel is wasted during the collision interval. Clearly, if many nodes want to frequently transmit frames, many transmissions will result in collisions, and much of the bandwidth of the broadcast channel will be wasted.

In order to ensure that the broadcast channel performs useful work when multiple nodes are active, it is necessary to somehow coordinate the transmissions of the active nodes. This coordination job is the responsibility of the multiple access protocol. Over the past thirty years, thousands of papers and hundreds of Ph.D. dissertations have been written on multiple access protocols; a comprehensive survey of this body of work is [Rom 1990]. Furthermore, dozens of different protocols have been implemented in a variety of link-layer technologies. Nevertheless, we can classify just about any multiple access protocol as belonging to one of three categories: **channel partitioning protocols, random access protocols,** and **taking-turns protocols.** We'll cover these categories of multiple access protocols in the following three subsections. Let us conclude this overview by noting that ideally, a multiple access protocol for a broadcast channel of rate *R* bits per second should have the following desirable characteristics:

1. When only one node has data to send, that node has a throughput of *R* bps.

2. When *M* nodes have data to send, each of these nodes has a throughput of *R*/*M* bps. This need not necessarily imply that each of the *M* nodes always have an instantaneous rate of *R*/*M*, but rather that each node should have an average transmission rate of *R*/*M* over some suitably defined interval of time.

3. The protocol is decentralized, that is, there are no master nodes that can fail and bring down the entire system.

4. The protocol is simple, so that it is inexpensive to implement.

5.3.1: Channel Partitioning Protocols

Recall from our early discussion back in Section 1.4 that time division multiplexing (TDM) and frequency division multiplexing (FDM) are two techniques that can be used to partition a broadcast channel's bandwidth among all nodes sharing that channel. As an example, suppose the channel supports *N* nodes and that the transmission rate of the channel is *R* bps. TDM divides time into **time frames** (not to be confused with the unit of data, the frame, at the data-link layer) and further divides each time frame into *N* **time slots.** Each slot time is then assigned to one of the *N* nodes. Whenever a node has a frame to send, it transmits the frame's bits during its assigned time slot in the revolving TDM frame. Typically, frame sizes are chosen so that a single frame can be transmitting during a slot time. Figure 5.11 shows a simple four-node TDM example. Returning to our cocktail party analogy, a TDM-regulated cocktail party would allow one partygoer to speak for a fixed period of time, and then allow another partygoer to speak for the same amount of time, and so on. Once everyone has had their chance to talk, the pattern repeats.

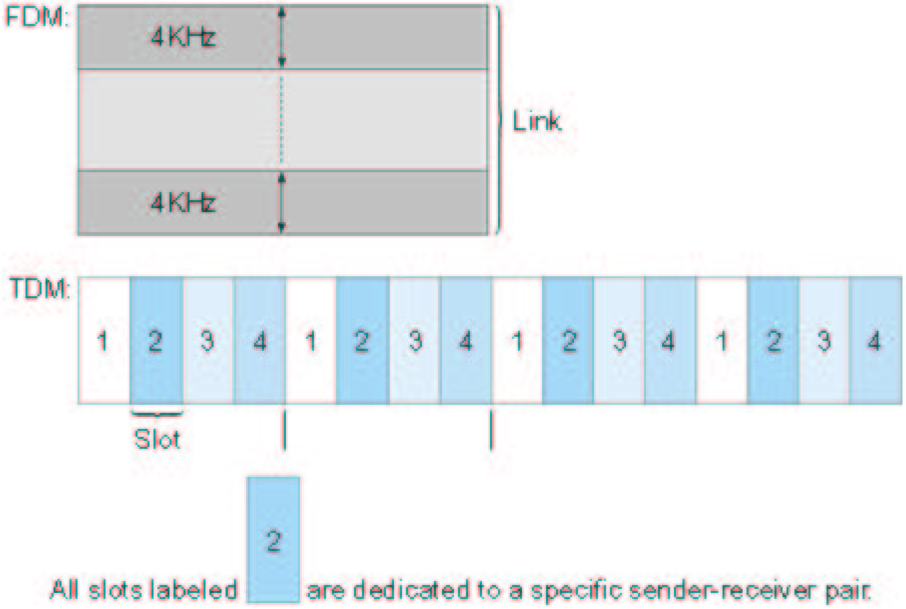


Figure 5.11: A four-node TDM and FDM example

TDM is appealing as it eliminates collisions and is perfectly fair: each node gets a dedicated transmission rate of *R*/*N* bps during each frame time. However, it has two major drawbacks. First, a node is limited to an average rate of *R*/*N* bps even when it is the only node with frames to send. A second drawback is that a node must always wait for its turn in the transmission sequence--again, even when it is the only node with a frame to send. Imagine the partygoer who is the only one with anything to say (and imagine that this is the even rarer circumstance where everyone at the party wants to hear what that one person has to say). Clearly, TDM would be a poor choice for a multiple access protocol for this particular party.

While TDM shares the broadcast channel in time, FDM divides the *R* bps channel into different frequencies (each with a bandwidth of *R*/*N*) and assigns each frequency to one of the *N* nodes. FDM thus creates *N* smaller channels of *R*/*N* bps out of the single, larger *R* bps channel. FDM shares both the advantages and drawbacks of TDM. It avoids collisions and divides the bandwidth fairly among the *N* nodes. However, FDM also shares a principal disadvantage with TDM--a node is limited to a bandwidth of *R*/*N*, even when it is the only node with frames to send.

A third channel partitioning protocol is **code division multiple access (CDMA).** While TDM and FDM assign time slots and frequencies, respectively, to the nodes, CDMA assigns a different *code* to each node. Each node then uses its unique code to encode the data bits it sends, as discussed below. We'll see that CDMA allows different nodes to transmit *simultaneously* and yet have their respective receivers correctly receive a sender's encoded data bits (assuming the receiver knows the sender's code) in spite of interfering transmissions by other nodes. CDMA has been used in military systems for some time (due to its antijamming properties) and is now beginning to find widespread civilian use, particularly for use in wireless multiple access channels.

In a CDMA protocol, each bit being sent by the sender is encoded by multiplying the bit by a signal (the code) that changes at a much faster rate (known as the **chipping rate**) than the original sequence of data bits. Figure 5.12 shows a simple, idealized CDMA encoding/decoding scenario. Suppose that the rate at which original data bits reach the CDMA encoder defines the unit of time; that is, each original data bit to be transmitted requires one bit-slot time. Let d_i be the value of the data bit for the *i*th bit slot. For mathematical convenience, we represent a data bit with a 0 value as -1. Each bit slot is further subdivided into *M* minislots; in Figure 5.12, *M* = 8, although in practice *M* is much larger. The CDMA code used by the sender consists of a sequence of *M* values, c_m, *m* = 1, . . . , *M*, each taking a +1 or -1 value. In the example in Figure 5.12, the *M*-bit CDMA code being used by the sender is (1, 1, 1, -1, 1, -1, -1, -1).

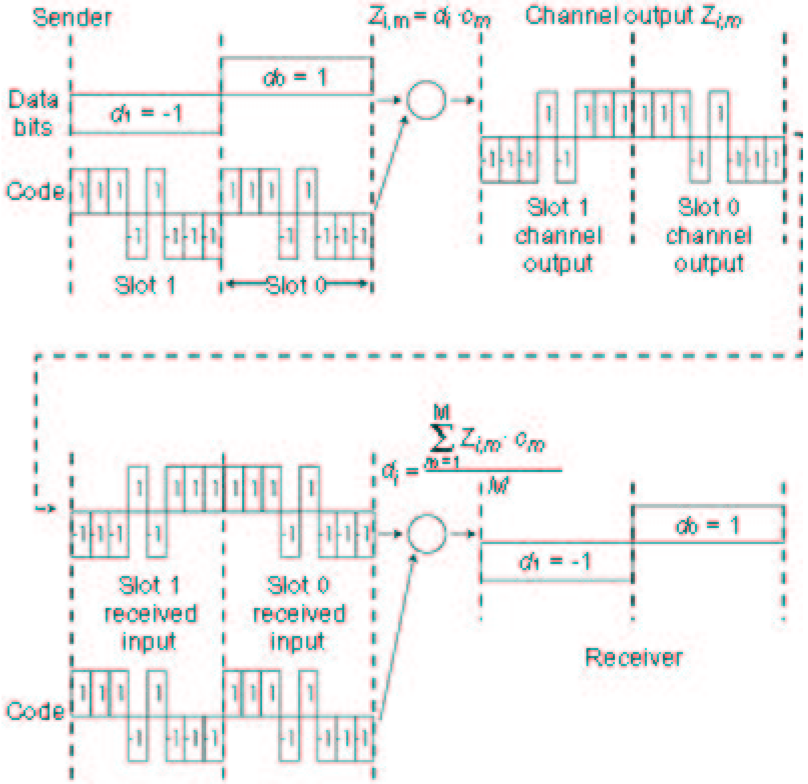


Figure 5.12: A simple CDMA example: Sender encoding, receiver decoding

To illustrate how CDMA works, let us focus on the *i*th data bit, d_i. For the *m*th mini-slot of the bit-transmission time of d_i, the output of the CDMA encoder, $Z_{i,m}$, is the value of d_i multiplied by the *m*th bit in the assigned CDMA code, c_m:

$$Z_{i,m} = d_i \bullet c_m \quad (5.1)$$

In a simple world, with no interfering senders, the receiver would receive the encoded bits, $Z_{i,m}$, and recover the original data bit, d_i, by computing:

$$d_i = \frac{1}{M}\sum_{m=1}^{M} Z_{i,m} \cdot c_m \quad (5.2)$$

The reader might want to work through the details of the example in Figure 5.12 to see that the original data bits are indeed correctly recovered at the receiver using Equation 5.2.

The world is far from ideal, however, and as noted above, CDMA must work in the presence of interfering senders that are encoding and transmitting their data using a different assigned code. But how can a CDMA receiver recover a sender's original data bits when those data bits are being tangled with bits being transmitted by other senders? CDMA works under the assumption that the interfering transmitted bit signals are additive, for example, that if three senders send a 1 value, and a fourth sender sends a -1 value during the same minislot, then the received signal at all receivers during that minislot is a 2 (since 1 + 1 + 1 - 1 = 2). In the presence of multiple senders, sender *s* computes its encoded transmissions, $Z^s_{i,m}$, in exactly the same manner as in Equation 5.1. The value received at a receiver during the *m*th minislot of the *i*th bit slot, however, is now the *sum* of the transmitted bits from all *N* senders during that minislot:

$$Z^*_{i,m} = \sum_{s=1}^{M} Z^s_{i,m}$$

Amazingly, if the senders' codes are chosen carefully, each receiver can recover the data sent by a given sender out of the aggregate signal simply by using the sender's code in exactly the same manner as in Equation 5.2:

$$d_i = \frac{1}{M}\sum_{m=1}^{M} Z^*_{i,m} \cdot c_m \quad (5.3)$$

Figure 5.13 illustrates a two-sender CDMA example. The *M*-bit CDMA code being used by the upper sender is (1, 1, 1, -1, 1, -1, -1, -1), while the CDMA code being used by the lower sender is (1, -1, 1, 1, 1,-1, 1, 1). Figure 5.13 illustrates a receiver recovering the original data bits from the upper sender. Note that the receiver is able to extract the data from sender 1 in spite of the interfering transmission from sender 2.

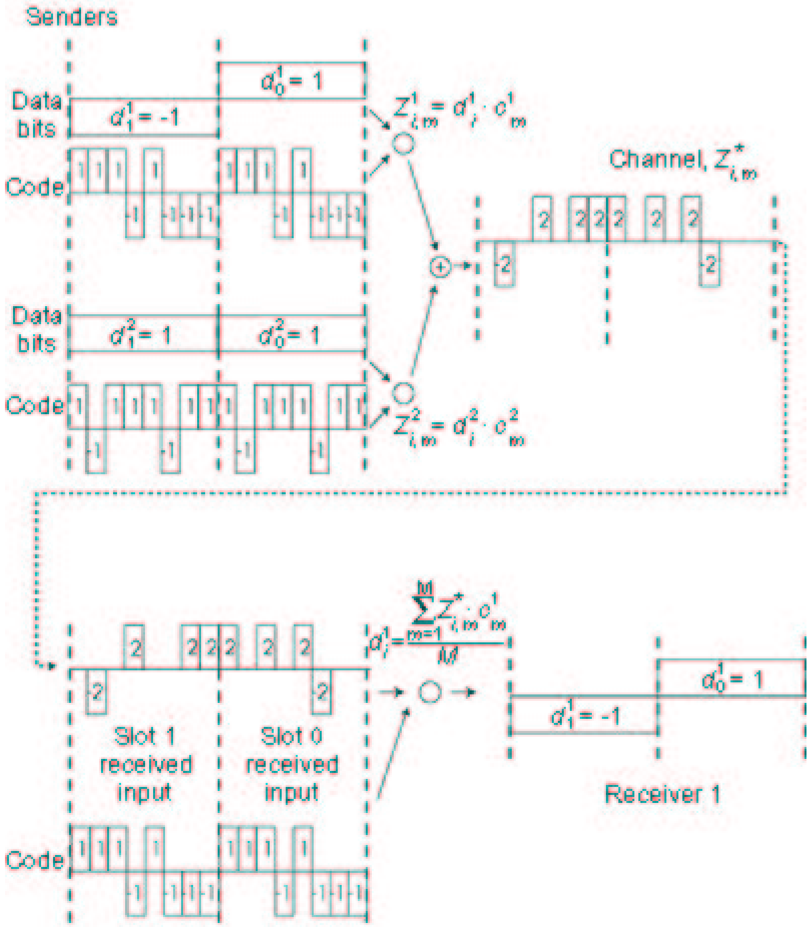


Figure 5.13: A two-sender CDMA example

Returning to our cocktail party analogy, a CDMA protocol is similar to having partygoers speaking in multiple languages; in such circumstances humans are actually quite good at locking into the conversation in the language they understand, while filtering out the remaining conversations. We see here that CDMA is a partitioning protocol in that it partitions the codespace (as opposed to time or frequency) and assigns each node a dedicated piece of the codespace.

Our discussion here of CDMA is necessarily brief and a number of difficult issues must be addressed in practice. First, in order for the CDMA receivers to be able to extract out a particular sender's signal, the CDMA codes must be carefully chosen. Secondly, our discussion has assumed that the received signal strengths from various senders at a receiver are the same; this can be difficult to achieve in practice. There is a considerable body of literature addressing these and other issues related to CDMA; see [Pickholtz 1982; Viterbi 1995] for details.

5.3.2: Random Access Protocols

The second broad class of multiple access protocols are so-called random access protocols. In a random access protocol, a transmitting node always transmits at the full rate of the channel, namely, *R* bps. When there is a collision, each node involved in the collision repeatedly retransmits its frame until the frame gets through without a collision. But when a node experiences a collision, it doesn't necessarily retransmit the frame right away. *Instead it waits a random delay before retransmitting the frame.* Each node involved in a collision chooses independent random delays. Because after a collision the random delays are independently chosen, it is possible that one of the nodes will pick a delay that is sufficiently less than the delays of the other colliding nodes and will therefore be able to sneak its frame into the channel without a collision.

There are dozens if not hundreds of random access protocols described in the literature [Rom 1990; Bertsekas 1991]. In this section we'll describe a few of the most commonly used random access protocols--the ALOHA protocols [Abramson 1970; Abramson 1985] and the carrier sense multiple access (CSMA) protocols [Kleinrock 1975b]. Later, in Section 5.5, we'll cover the details of Ethernet [Metcalfe 1976], a popular and widely deployed CSMA protocol.

Slotted ALOHA

Let's begin our study of random access protocols with one of the most simple random access protocols, the so-called slotted ALOHA protocol. In our description of slotted ALOHA, we assume the following:

- All frames consist of exactly *L* bits.
- Time is divided into slots of size *L*/*R* seconds (that is, a slot equals the time to transmit one frame).
- Nodes start to transmit frames only at the beginnings of slots.
- The nodes are synchronized so that each node knows when the slots begin.
- If two or more frames collide in a slot, then all the nodes detect the collision event before the slot ends.

Let *p* be a probability, that is, a number between 0 and 1. The operation of slotted ALOHA in each node is simple:

- When the node has a fresh frame to send, it waits until the beginning of the next slot and transmits the entire frame in the slot.

- If there isn’t a collision, the node has successfully transmitted its frame and thus need not consider retransmitting the frame. (The node can prepare a new frame for transmission, if it has one.)
- If there is a collision, the node detects the collision before the end of the slot. The node retransmits its frame in each subsequent slot with probability *p* until the frame is transmitted without a collision.

By retransmitting with probability *p*, we mean that the node effectively tosses a biased coin; the event heads corresponds to retransmit, which occurs with probability *p*. The event tails corresponds to "skip the slot and toss the coin again in the next slot"; this occurs with probability (1 - *p*). Each of the nodes involved in the collision toss their coins independently.

Slotted ALOHA would appear to have many advantages. Unlike channel partitioning, slotted ALOHA allows a single active node (that is, a node with a frame to send) to continuously transmit frames at the full rate of the channel. Slotted ALOHA is also highly decentralized, because each node detects collisions and independently decides when to retransmit. (Slotted ALOHA does, however, require the slots to be synchronized in the nodes; we’ll shortly discuss an unslotted version of the ALOHA protocol, as well as CSMA protocols, none of which require such synchronization and are therefore fully decentralized.) Slotted ALOHA is also an extremely simple protocol.

Slotted ALOHA works well when there is only one active node, but how efficient is it when there are multiple active nodes? There are two possible efficiency concerns here. First, as shown in Figure 5.14, when there are multiple active nodes, a certain fraction of the slots will have *collisions* and will therefore be "wasted." The second concern is that another fraction of the slots will be *empty* because all active nodes refrain from transmitting as a result of the probabilistic transmission policy. The only "unwasted" slots will be those in which exactly one node transmits. A slot in which exactly one node transmits is said to be a **successful slot.** The **efficiency** of a slotted multiple access protocol is defined to be the long-run fraction of successful slots in the case when there are a large number of active nodes, each always having a large number of frames to send. Note that if no form of access control were used, and each node were to immediately retransmit after each collision, the efficiency would be zero. Slotted ALOHA clearly increases the efficiency beyond zero, but by how much?

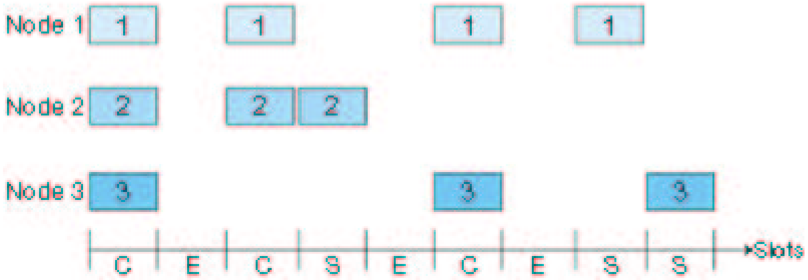


Figure 5.14: Nodes 1, 2, and 3 collide in the first slot. Node 2 finally succeeds in the fourth slot, node 1 in the eighth slot, and node 3 in the ninth slot. The notation C, E, and S represent "collision slot," "empty slot," and "successful slot," respectively.

We now proceed to outline the derivation of the maximum efficiency of slotted ALOHA. To keep this derivation simple, let's modify the protocol a little and assume that each node attempts to transmit a frame in each slot with probability *p*. (That is, we assume that each node always has a frame to send and that the node transmits with probability *p* for a fresh frame as well as for a frame that has already suffered a collision.) Suppose first there are *N* nodes. Then the probability that a given slot is a successful slot is the probability that one of the nodes transmits and that the remaining *N* - 1 nodes do not transmit. The probability that a given node transmits is *p*; the probability that the remaining nodes do not transmit is $(1 - p)^{N-1}$. Therefore the probability a given node has a success is $p(1 - p)^{N-1}$. Because there are *N* nodes, the probability that an arbitrary node has a success is $Np(1 - p)^{N-1}$.

Thus, when there are *N* active nodes, the efficiency of slotted ALOHA is $Np(1 - p)^{N-1}$. To obtain the *maximum* efficiency for *N* active nodes, we have to find the *p** that maximizes this expression. (See the homework problems for a general outline of this derivation.) And to obtain the maximum efficiency for a large number of active nodes, we take the limit of $Np^*(1 - p^*)^{N-1}$ as *N* approaches infinity. (Again, see the homework problems.) After performing these calculations, we'll find that the maximum efficiency of the protocol is given by 1/*e* = 0.37. That is, when a large number of nodes have many frames to transmit, then (at best) only 37% of the slots do useful work. Thus the effective transmission rate of the channel is not *R* bps but only 0.37 *R* bps! A similar analysis also shows that 37% of the slots go empty and 26% of slots have collisions. Imagine the poor network administrator who has purchased a 100 Mbps slotted ALOHA system, expecting to be able to use the network to transmit data among a large number of users at an aggregate rate of, say, 80 Mbps! Although the channel is capable of transmitting a given frame at the full channel rate of 100 Mbps, in the long term, the successful throughput of this channel will be less than 37 Mbps.

ALOHA

The slotted ALOHA protocol required that all nodes synchronize their transmissions to start at the beginning of a slot. The first ALOHA protocol [Abramson 1970] was actually an unslotted, fully decentralized protocol. In so-called pure ALOHA, when a frame first arrives (that is, a network-layer datagram is passed down from the network layer at the sending node), the node immediately transmits the frame in its entirety into the broadcast channel. If a transmitted frame experiences a collision with one or more other transmissions, the node will then immediately (after completely transmitting its collided frame) retransmit the frame with probability p. Otherwise, the node waits for a frame transmission time. After this wait, it then transmits the frame with probability p, or waits (remaining idle) for another frame time with probability $1 - p$.

To determine the maximum efficiency of pure ALOHA, we focus on an individual node. We'll make the same assumptions as in our slotted ALOHA analysis and take the frame transmission time to be the unit of time. At any given time, the probability that a node is transmitting a frame is p. Suppose this frame begins transmission at time t_0. As shown in Figure 5.15, in order for this frame to be successfully transmitted, no other nodes can begin their transmission in the interval of time $[t_0 - 1, t_0]$. Such a transmission would overlap with the beginning of the transmission of node i's frame. The probability that all other nodes do not begin a transmission in this interval is $(1 - p)^{N-1}$. Similarly, no other node can begin a transmission while node i is transmitting, as such a transmission would overlap with the latter part of node i's transmission. The probability that all other nodes do not begin a transmission in this interval is also $(1 - p)^{N-1}$. Thus, the probability that a given node has a successful transmission is $p(1 - p)^{2(N-1)}$. By taking limits as in the slotted ALOHA case, we find that the maximum efficiency of the pure ALOHA protocol is only $1/(2e)$--exactly half that of slotted ALOHA. This then is the price to be paid for a fully decentralized ALOHA protocol.

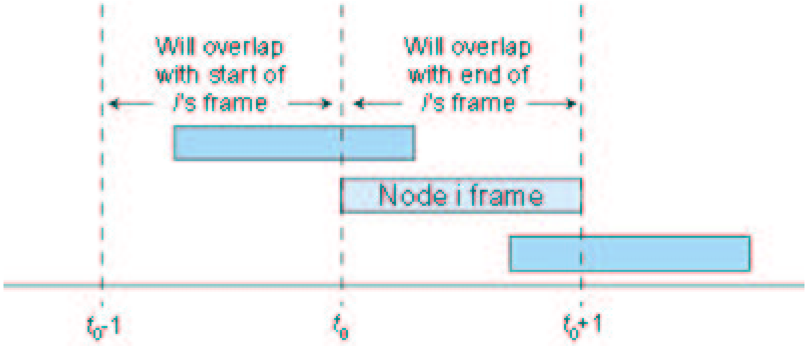

Will overlap with start of i's frame
Will overlap with end of i's frame
Node i frame
t_0-1
t_0
t_0+1

Figure 5.15: Interfering transmissions in pure ALOHA

Case History

Norm Abramson and Alohanet

Norm Abramson, a Ph.D. engineer, had a passion for surfing and an interest in packet switching. This combination of interests brought him to the University of Hawaii in 1969. Hawaii consists of many mountainous islands, making it difficult to install and operate land-based networks. When not surfing, Abramson thought about how to design a network that does packet switching over radio. The network he designed had one central host and several satellite nodes scattered over the Hawaiian islands. The network had two channels, each using a different frequency band. The downlink channel broadcasted packets from the central host to the satellite hosts; and the upstream channel sent packets from the satellite hosts to the central host. In addition to sending informational packets, the central host also sent on the downstream channel an acknowledgement for each packet successfully received from the satellite hosts.

Because the satellite hosts transmitted packets in a decentralized fashion, collisions on the upstream channel inevitably occurred. This observation led Abramson to devise the pure Aloha protocol, as described in this chapter. In 1970, with continued funding from ARPA, Abramson connected his Alohanet to the ARPAnet. Abramson's work is important not only because it was the first example of a radio packet network, but also because it inspired Bob Metcalfe. A few years after Abramson's invention, Metcalfe modified the Aloha protocol to create the CSMA/CD protocol and the Ethernet LAN.

CSMA--Carrier Sense Multiple Access

In both slotted and pure ALOHA, a node's decision to transmit is made independently of the activity of the other nodes attached to the broadcast channel. In particular, a node neither pays attention to whether another node happens to be transmitting when it begins to transmit, nor stops transmitting if another node begins to interfere with its transmission. In our cocktail party analogy, ALOHA protocols are quite like a boorish partygoer who continues to chatter away regardless of whether other people are talking. As humans, we have human protocols that allow us to not only behave with more civility, but also to decrease the amount of time spent "colliding" with each other in conversation and consequently increase the amount of data we exchange in our conversations. Specifically, there are two important rules for polite human conversation:

- *Listen before speaking.* If someone else is speaking, wait until they are done. In the networking world, this is called **carrier sensing**--a node listens to the channel before transmitting. If a frame from another node is currently being transmitted into the channel, a node then waits ("backs off") a random amount of time and then again senses the channel. If the channel is sensed to be idle, the node then begins frame transmission. Otherwise, the node waits another random amount of time and repeats this process.

- *If someone else begins talking at the same time, stop talking.* In the networking world, this is called **collision detection**--a transmitting node listens to the channel while it is transmitting. If it detects that another node is transmitting an interfering frame, it stops transmitting and uses some protocol to determine when it should next attempt to transmit.

These two rules are embodied in the family of **CSMA** (carrier sense multiple access) and **CSMA/CD** (CSMA with collision detection) protocols [Kleinrock 1975b; Metcalfe 1976; Lam 1980; Rom 1990]. Many variations on CSMA and CSMA/CD have been proposed, with the differences being primarily in the manner in which nodes perform backoff. The reader can consult these references for the details of these protocols. We'll study the CSMA/CD scheme used in Ethernet in detail in Section 5.5. Here, we'll consider a few of the most important, and fundamental, characteristics of CSMA and CSMA/CD.

The first question that one might ask about CSMA is that if all nodes perform carrier sensing, why do collisions occur in the first place? After all, a node will refrain from transmitting whenever it senses that another node is transmitting. The answer to the question can best be illustrated using space-time diagrams [Molle 1987]. Figure 5.16 shows a space-time diagram of four nodes (A, B, C, D) attached to a linear broadcast bus. The horizontal axis shows the position of each node in space; the vertical axis represents time.

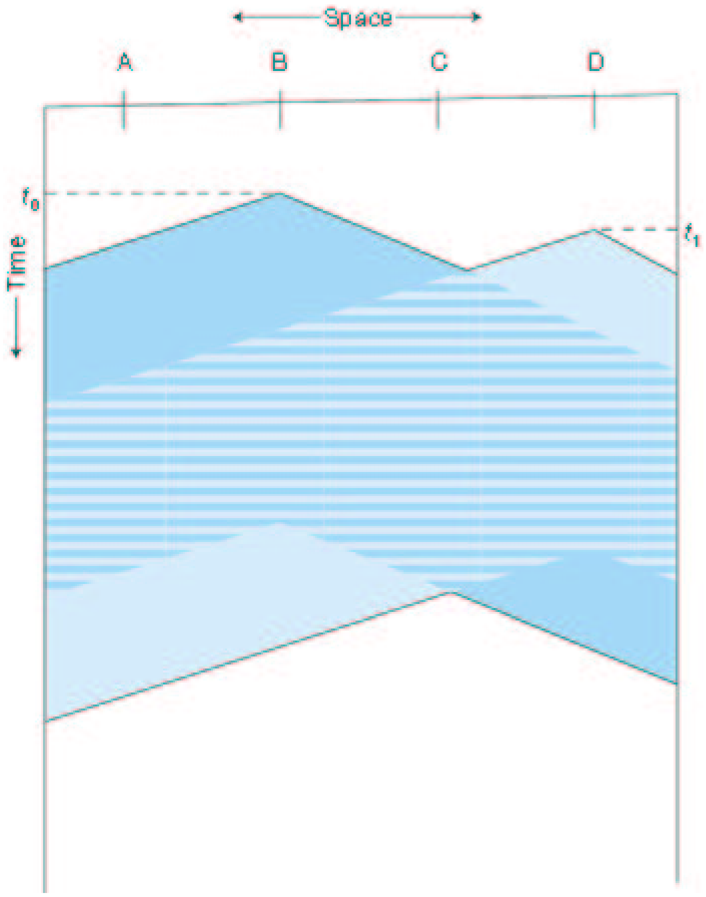


Figure 5.16: Space-time diagram of two CSMA nodes with colliding transmissions

At time t_0, node B senses the channel is idle, as no other nodes are currently transmitting. Node B thus begins transmitting, with its bits propagating in both directions along the broadcast medium. The downward propagation of B's bits in Figure 5.16 with increasing time indicates that a nonzero amount of time is needed for B's bits to actually propagate (albeit at near the speed-of-light) along the broadcast medium. At time t_1 ($t_1 > t_0$), node D has a frame to send. Although node B is currently transmitting at time t_1, the bits being transmitted by B have yet to reach D, and thus D senses the channel idle at t_1. In accordance with the CSMA protocol, D thus begins transmitting its frame. A short time later, B's transmission begins to interfere with D's transmission at D. From Figure 5.16, it is evident that the end-to-end **channel propagation delay** of a broadcast channel--the time it takes for a signal to propagate from one of the channels to another--will play a crucial role in determining its performance. The longer this propagation delay, the larger the chance that a carrier-sensing node is not yet able to sense a transmission that has already begun at another node in the network.

In Figure 5.16, nodes do not perform collision detection; both B and D continue to transmit their frames in their entirety even though a collision has occurred. When a node performs collision detection, it will cease transmission as soon as it detects a collision. Figure 5.17 shows the same scenario as in Figure 5.16, except that the two nodes each abort their transmission a short time after detecting a collision. Clearly, adding collision detection to a multiple access protocol will help protocol performance by not transmitting a useless, damaged (by interference with a frame from another node) frame in its entirety. The Ethernet protocol we will study in Section 5.5 is a CSMA protocol that uses collision detection.

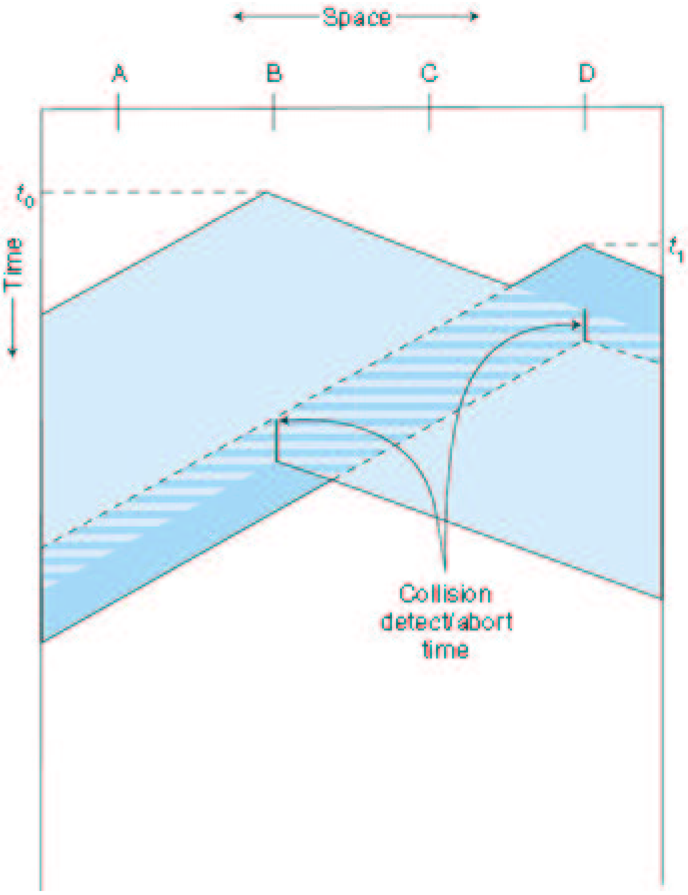


Figure 5.17: CSMA with collision detection

5.3.3: Taking-Turns Protocols

Recall that two desirable properties of a multiple access protocol are (1) when only one node is active, the active node has a throughput of *R* bps, and (2) when *M* nodes are active, then each active node has a throughput of nearly *R*/*M* bps. The ALOHA and CSMA protocols have this first property but not the second. This has motivated researchers to create another class of protocols--the **taking-turns protocols.** As with random-access protocols, there are dozens of taking-turns protocols, and each one of these protocols has many variations. We'll discuss two of the more important protocols here. The first one is the **polling protocol.** The polling protocol requires one of the nodes to be designated as a master node. The master node **polls** each of the nodes in a round-robin fashion. In particular, the master node first sends a message to node 1, saying that it can transmit up to some maximum number of frames. After node 1 transmits some frames, the master node tells node 2 it can transmit up to the maximum number of frames. (The master node can determine when a node has finished sending its frames by observing the lack of a signal on the channel.) The procedure continues in this manner, with the master node polling each of the nodes in a cyclic manner.

The polling protocol eliminates the collisions and the empty slots that plague the random access protocols. This allows it to have a much higher efficiency. But it also has a few drawbacks. The first drawback is that the protocol introduces a polling delay--the amount of time required to notify a node that it can transmit. If, for example, only one node is active, then the node will transmit at a rate less than *R* bps, as the master node must poll each of the inactive nodes in turn, each time the active node has sent its maximum number of frames. The second drawback, which is potentially more serious, is that if the master node fails, the entire channel becomes inoperative.

The second taking-turn protocol is the **token-passing protocol.** In this protocol there is no master node. A small, special-purpose frame known as a **token** is exchanged among the nodes in some fixed order. For example, node 1 might always send the token to node 2, node 2 might always send the token to node 3, node *N* might always send the token to node 1. When a node receives a token, it holds onto the token only if it has some frames to transmit; otherwise, it immediately forwards the token to the next node. If a node does have frames to transmit when it receives the token, it sends up to a maximum number of frames and then forwards the token to the next node. Token passing is decentralized and has a high efficiency. But it has its problems as well. For example, the failure of one node can crash the entire channel. Or if a node accidentally neglects to release the token, then some recovery procedure must be invoked to get the token back in circulation. Over the years many token-passing products have been developed, and each one had to address these as well as other sticky issues.

5.3.4: Local Area Networks (LANs)

Multiple access protocols are used in conjunction with many different types of broadcast channels. They have been used for satellite and wireless channels, whose nodes transmit over a common frequency spectrum. They are currently used in the upstream channel for cable access to the Internet (see Section

1.5), and they are extensively used in local area networks (LANs). Recall that a **LAN** is a computer network that is concentrated in a geographical area, such as in a building or on a university campus. When a user accesses the Internet from a university or corporate campus, the access is almost always by way of a LAN. For this type of Internet access, the user's host is a node on the LAN, and the LAN provides access to the Internet through a router, as shown in Figure 5.18. The LAN is a single "link" between each user host and the router; it therefore uses a link-layer protocol, part of which is a multiple access protocol. The transmission rate, *R*, of most LANs is very high. Even in the early 1980s, 10 Mbps LANs were common; today, 100 Mbps LANs are common, and 1 Gbps LANs are available.

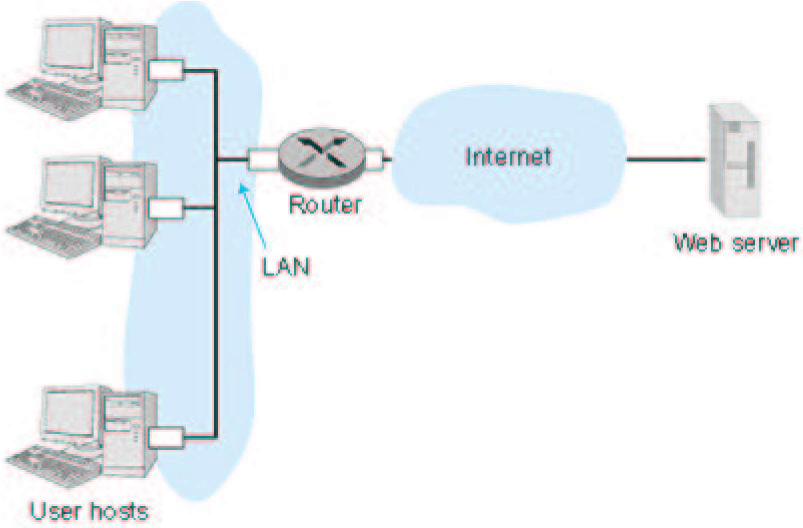


Figure 5.18: User hosts access an Internet Web server through a LAN. The broadcast channel between a user host and the router consists of one "link."

In the 1980s and the early 1990s, two classes of LAN technologies were popular in the workplace. The first class consists of the Ethernet LANs (also known as 802.3 LANs [IEEE 802.3 1998; Spurgeon 1999]), which are random-access-based. The second class of LAN technologies are token-passing technologies, including token ring (also known as IEEE 802.5 [IEEE 802.5 1998]) and **FDDI** (also known as fiber distributed data interface [Jain 1994]). Because we shall explore the Ethernet technologies in some detail in Section 5.4, we focus our discussion here on the token-passing LANs. Our discussion on token-passing technologies is intentionally brief, since these technologies have become relatively minor players in the face of relentless Ethernet competition. Nevertheless, in order to provide examples of token-passing technology and to give a little historical perspective, it is useful to say a few words about token rings.

In a token ring LAN, the *N* nodes of the LAN (hosts and routers) are connected in a ring by direct links. The topology of the token ring defines the token-passing order. When a node obtains the token and sends a frame, the frame propagates around the entire ring, thereby creating a virtual broadcast channel. The node that sends the frame has the responsibility of removing the frame from the ring. FDDI was designed for geographically larger LANs, including so-called metropolitan area networks (MANs). For geographically large LANs (spread out over several kilometers) it is inefficient to let a frame propagate back to the sending node once the frame has passed the destination node. FDDI has the destination node remove the frame from the ring. (Strictly speaking, FDDI is not a pure broadcast channel, as every node does not receive every transmitted frame.) You can learn more about token ring and FDDI by visiting the 3Com adapter page [3Com 1999].

Online Book

5.4: LAN Addresses and ARP

As we learned in the previous section, nodes in LANs send frames to each other over a broadcast channel. This means that when a node in a LAN transmits a frame, every other node connected to the LAN receives the frame. But usually, a node in the LAN doesn’t want to send a frame to *all* of the other LAN nodes but instead wants to send to some *particular* LAN node. To provide this functionality, the nodes on the LAN must be able to address each other when sending frames, that is, the nodes need LAN addresses and the link-layer frame needs a field to contain such a destination address. In this manner, when a node receives a frame, it can determine whether the frame was intended for it or for some other node in the LAN:

- If the destination address of the frame matches a receiving node’s LAN address, then the node extracts the network-layer datagram from the data-link layer frame and passes the datagram up the protocol stack.

- If the destination address does not match the address of the receiving node, the node simply discards the frame.

5.4.1: LAN Addresses

In truth, it is not a node that has a LAN address but instead a node’s adapter that has a LAN address. This is illustrated in Figure 5.19. A **LAN address** is also variously called a **physical address,** an **Ethernet address,** or a **MAC** (media access control) **address.** For most LANs (including Ethernet and token-passing LANs), the LAN address is six-bytes long, giving 2^{48} possible LAN addresses. These six-byte addresses are typically expressed in hexadecimal notation, with each byte of the address expressed as a pair of hexadecimal numbers. An adapter's LAN address is permanent--when an adapter is manufactured, a LAN address is burned into the adapter's ROM.

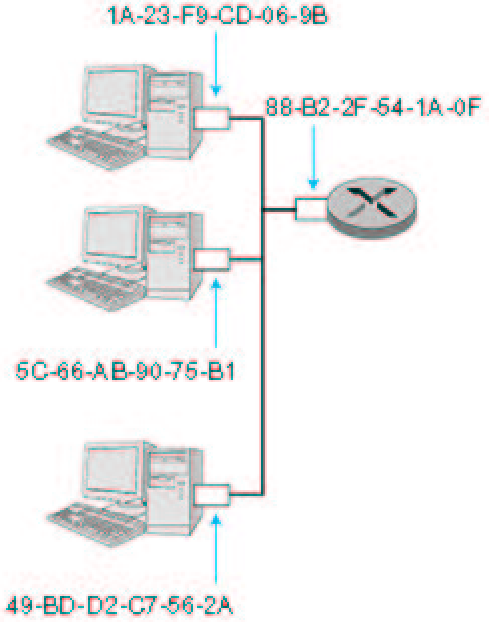


Figure 5.19: Each adapter connected to a LAN has a unique LAN address

One interesting property of LAN addresses is that no two adapters have the same address. This might seem surprising given that adapters are manufactured in many different countries by many different companies. How does a company manufacturing adapters in Taiwan make sure that it is using different addresses from a company manufacturing adapters in Belgium? The answer is that IEEE manages the physical address space. In particular, when a company wants to manufacture adapters, it purchases a chunk of the address space consisting of 2^{24} addresses for a nominal fee. IEEE allocates the chunk of 2^{24} addresses by fixing the first 24 bits of a physical address and letting the company create unique combinations of the last 24 bits for each adapter.

An adapter's LAN address has a flat structure (as opposed to a hierarchical structure), and doesn't change no matter where the adapter goes. A portable computer with an Ethernet card always has the same LAN address, no matter where the portable goes. Recall that, in contrast, an IP address has a hierarchical structure (that is, a network part and a host part), and a node's IP address needs to be changed when the host moves. An adapter's LAN address is analogous to a person's social security number, which also has a flat addressing structure and which doesn’t change no matter where the person goes. An IP address is analogous to a person’s postal address, which is hierarchical and which needs to be changed whenever a person moves.

As we described at the beginning of this section, when an adapter wants to send a frame to some destination adapter on the same LAN, the sending adapter inserts the destination’s LAN address into the frame. When the destination adapter receives the frame, it extracts the enclosed datagram and passes the datagram up the protocol stack. All the other adapters on the LAN also receive the frame. However, these other adapters discard the frame without passing the network-layer datagram up the protocol stack. Thus, these other adapters do not have to interrupt their hosting node when they receive datagrams destined to other nodes. However, sometimes a sending adapter *does* want all the other adapters on the LAN to receive and *process* the frame it is about to send. In this case, the sending adapter inserts a special **LAN broadcast address** into the destination address field of the frame. For LANs that use six-byte addresses (such as Ethernet and token-passing LANs), the broadcast address is a string of 48 consecutive 1s (that is, `FF-FF-FF-FF-FF-FF` in hexadecimal notation).

Principles in Practice

Keeping the Layers Independent

There are several reasons why nodes have LAN addresses in addition to also having network-layer addresses. First, LANs are designed for arbitrary network-layer protocols, not just for IP and the Internet. If adapters were to get assigned IP addresses rather than "neutral" LAN addresses, then adapters would not be able to easily support other network-layer protocols (for example, IPX or DECNet). Second, if adapters were to use network-layer addresses instead of LAN addresses, the network-layer address would have to be stored in the adapter RAM and reconfigured every time the adapter was moved (or powered up). Another option is to not use any addresses in the adapters, and have each adapter pass the data (typically, an IP datagram) of each frame it receives to its parent node. The parent node could then check for a matching network-layer address. One problem with this option is that the parent node will be interrupted by every frame sent on the LAN, as well as by frames that are destined for other nodes on the same broadcast LAN. In summary, in order for the layers to be largely independent building blocks in a network architecture, many layers need to have their own address scheme. We have now seen three different types of addresses: host names for the application layer, IP addresses for the network layer, and LAN addresses for the link layer.

5.4.2: Address Resolution Protocol

Because there are both network-layer addresses (for example, Internet IP addresses) and link-layer addresses (that is, LAN addresses), there is a need to translate between them. For the Internet, this is the job of the address resolution protocol (ARP) [RFC 826]. Every Internet host and router on a LAN has an **ARP module.**

To motivate ARP, consider the network shown in Figure 5.20. In this simple example, each node has a single IP address and each node's adapter has a LAN address. As usual, IP addresses are shown in dotted-decimal notation and LAN addresses are shown in hexadecimal notation. Now suppose that the node with IP address 222.222.222.220 wants to send an IP datagram to node 222.222.222.222. To accomplish this task, the sending node must give its adapter not only the IP datagram but also the LAN address for node 222.222.222.222. When passed the IP datagram and the LAN address, the sending node's adapter will construct a data-link layer frame containing the receiving node's LAN address and send the frame into the LAN. But how does the sending node determine the LAN address for the node with IP address 222.222.222.222? It does this by providing its ARP module with the IP address 222.222.222.222. ARP then responds with the corresponding LAN address, namely, 49-BD-D2-C7-56-2A.

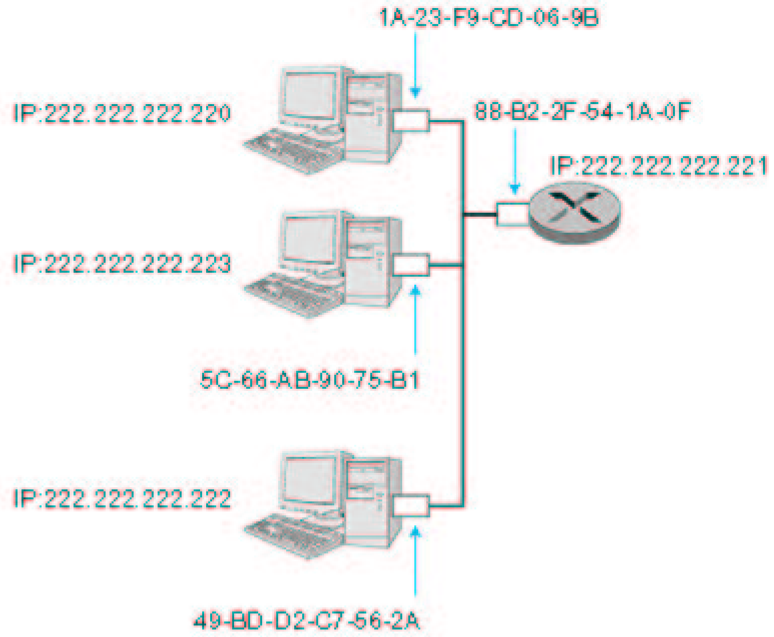


Figure 5.20: Each node on a LAN has an IP address, and each node's adapter has a LAN address

So we see that ARP resolves an IP address to a LAN address. In many ways it is analogous to DNS (studied in Section 2.5), which resolves hostnames to IP addresses. However, one important difference between the two resolvers is that DNS resolves hostnames for hosts anywhere in the Internet, whereas ARP only resolves IP addresses for nodes on the same LAN. If a node in California were to try to use ARP to resolve the IP address for a node in Mississippi, ARP would return with an error.

Now that we have explained what ARP does, let's look at how it works.

The ARP module in each node has a table in its RAM called an **ARP table**. This table contains the mappings of IP addresses to LAN addresses. Figure 5.21 shows what an ARP table in node 222.222.222.220 might look like. For each address mapping the table also contains a time-to-live (TTL) entry, which indicates when the entry will be deleted from the table. Note that the table does not necessarily contain an entry for every node on the LAN; some nodes may have had entries that expired over time, whereas other nodes may have never been entered into the table. A typical expiration time for an entry is 20 minutes from when an entry is placed in an ARP table.

IP address	LAN address	TTL
222.222.222.221	88-B2-2F-54-1A-0F	13:45:00
222.222.222.223	5C-66-AB-90-75-B1	13:52:00

Figure 5.21: A possible ARP table in node 222.222.222.220

Now suppose that node 222.222.222.220 wants to send a datagram that is IP-addressed to another node on that LAN. The sending node needs to obtain the LAN address of the destination node, given the IP address of that node. This task is easy if the sending node’s ARP table has an entry for the destination node. But what if the ARP table doesn’t currently have an entry for the destination node? In particular, suppose node 222.222.222.220 wants to send a datagram to node 222.222.222.222. In this case, the sending node uses the ARP protocol to resolve the address. First, the sending node constructs a special packet called an **ARP packet.** An ARP packet has several fields, including the sending and receiving IP and LAN addresses. Both ARP query and response packets have the same format. The purpose of the ARP query packet is to query all the other nodes on the LAN to determine the LAN address corresponding to the IP address that is being resolved.

Returning to our example, node 222.222.222.220 passes an ARP query packet to the adapter along with an indication that the adapter should send the packet to the LAN broadcast address, namely, FF-FF-FF-FF-FF-FF. The adapter encapsulates the ARP packet in a data-link frame, uses the broadcast address for the frame’s destination address, and transmits the frame into the LAN. Recalling our social security number/postal address analogy, note that an ARP query is equivalent to a person shouting out in a crowded room of cubicles in some company (say, AnyCorp): "What is the social security number of the person whose postal address is Cubicle 13, Room 112, AnyCorp, Palo Alto, CA?" The frame containing the ARP query is received by all the other adapters on the LAN, and (because of the broadcast address) each adapter passes the ARP packet within the frame up to its hosting node. Each node checks to see if its IP address matches the destination IP address in the ARP packet. The one node with a match sends back to the querying node a response ARP packet with the desired mapping. The querying node (222.222.222.220) can then update its ARP table and send its IP datagram.

There are a couple of interesting things to note about the ARP protocol. First, the query ARP message is sent within a broadcast frame whereas the response ARP message is sent within a standard frame. Before reading on you should think about why this is so. Second, ARP is plug-and-play, that is, a node's ARP table gets built automatically--it doesn't have to be configured by a systems administrator. And if a node is disconnected from the LAN, its entry is eventually deleted from the table.

Sending a datagram to a node off the LAN

It should now be clear how ARP operates when a node wants to send a datagram to another node *on the same LAN.* But now let's look at the more complicated situation when a node on a LAN wants to send a network-layer datagram to a node *off the LAN.* Let us discuss this issue in the context of Figure 5.22, which shows a simple network consisting of two LANs interconnected by a router.

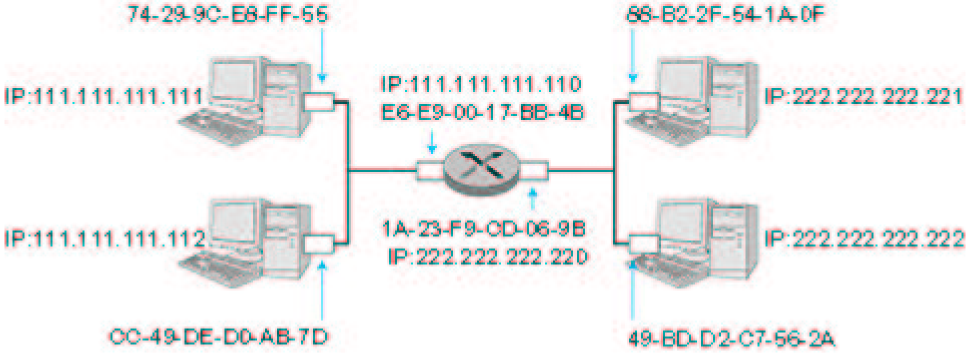


Figure 5.22: Two LANs interconnected by a router

There are several interesting things to note about Figure 5.22. First, there are two types of nodes: hosts and routers. Each host has exactly one IP address and one adapter. But, as discussed in Section 4.4, a router has an IP address for *each* of its interfaces. Each router interface also has its own ARP module (in the router) and its own adapter. Because the router in Figure 5.22 has two interfaces, it has two IP addresses, two ARP modules, and two adapters. Of course, each adapter in the network has its own LAN address.

Also note that all of the interfaces connected to LAN 1 have addresses of the form 111.111.111.xxx and all of the interfaces connected to LAN 2 have the form 222.222.222.xxx. Thus, in this example, the first three bytes of the IP address specifies the "network," whereas the last byte specifies the specific interface in the network.

Now suppose that host 111.111.111.111 wants to send an IP datagram to host 222.222.222.222. The sending host passes the datagram to its adapter, as usual. But the sending host must also indicate to its adapter an appropriate destination LAN address. What LAN address should the adapter use? One might venture to guess that the appropriate LAN address is that of the adapter for host 222.222.222.222, namely, 49-BD-D2-C7-56-2A. This guess is, however, wrong. If the sending adapter were to use that LAN address, then none of the adapters on LAN 1 would bother to pass the IP datagram up to its network layer, since the frame's destination address would not match the LAN address of any adapter on LAN 1. The datagram would just die and go to datagram heaven.

If we look carefully at Figure 5.22, we see that in order for a datagram to go from 111.111.111.111 to a node on LAN 2, the datagram must first be sent to the router interface 111.111.111.110. As discussed in Section 4.4, the routing table in host 111.111.111.111 would indicate that to reach host 222.222.222. 222, the datagram must first be sent to router interface 111.111.111.110. Thus, the appropriate LAN address for the frame is the address of the adapter for router interface 111.111.111.110, namely, E6-E9-00-17-BB-4B. How does the sending host acquire the LAN address of 111.111.111.110? By using ARP, of course! Once the sending adapter has this LAN address, it creates a frame and sends the frame into LAN 1. The router adapter on LAN 1 sees that the data-link frame is addressed to it, and therefore passes the frame to the network layer of the router. Hooray, the IP datagram has successfully been moved from source host to the router! But we are not done. We still have to move the datagram from the router to the destination! The router now has to determine the correct interface on which the datagram is to be forwarded. As discussed in Section 4.4, this is done by consulting a routing table in the router. The routing table tells the router that the datagram is to be forwarded via router interface 222.222.222.220. This interface then passes the datagram to its adapter, which encapsulates the datagram in a new frame and sends the frame into LAN 2. This time, the destination LAN address of the frame is indeed the LAN address of the ultimate destination. And how does the router obtain this destination LAN address? From ARP, of course!

ARP for Ethernet is defined in RFC 826. A nice introduction to ARP is given in the TCP/IP tutorial, RFC 1180. We shall explore ARP in more detail in the homework problems.

Online Book

5.5: Ethernet

Ethernet has pretty much taken over the LAN market. As recently as the 1980s and the early 1990s, Ethernet faced many challenges from other LAN technologies, including token ring, FDDI, and ATM. Some of these other technologies succeeded at capturing a part of the market share for a few years. But since its invention in the mid-1970s, Ethernet has continued to evolve and grow, and has held on to its dominant market share. Today, Ethernet is by far the most prevalent LAN technology, and is likely to remain so for the foreseeable future. One might say that Ethernet has been to local area networking what the Internet has been to global networking.

There are many reasons for Ethernet's success. First, Ethernet was the first widely deployed high-speed LAN. Because it was deployed early, network administrators became intimately familiar with Ethernet--its wonders and its quirks--and were reluctant to switch over to other LAN technologies when they came on the scene. Second, token ring, FDDI, and ATM are more complex and expensive than Ethernet, which further discouraged network administrators from switching over. Third, the most compelling reason to switch to another LAN technology (such as FDDI or ATM) was usually the higher data rate of the new technology; however, Ethernet always fought back, producing versions that operated at equal data rates or higher. Switched Ethernet was also introduced in the early 1990s, which further increased its effective data rates. Finally, because Ethernet has been so popular, Ethernet hardware (in particular, network interface cards) has become a commodity and is remarkably cheap. This low cost is also due to the fact that Ethernet's multiple access protocol, CSMA/CD, is completely decentralized, which has also contributed to a simple design.

The original Ethernet LAN, as shown in Figure 5.23, was invented in the mid 1970s by Bob Metcalfe and David Boggs. An excellent source of online information about Ethernet is Spurgeon's Ethernet Web Site [Spurgeon 1999].

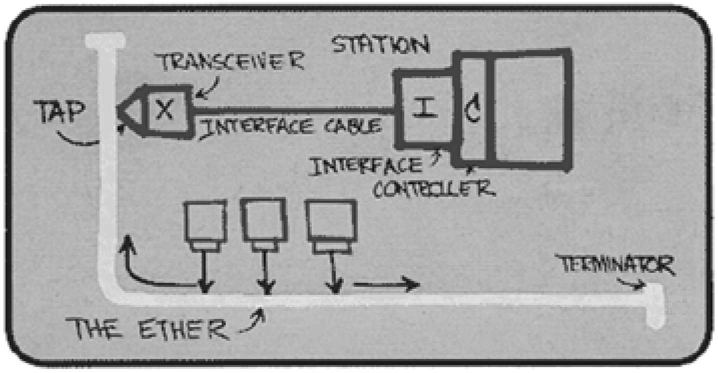


Figure 5.23: The original Metcalfe design led to the 10Base5 Ethernet standard, which included an interface cable that connected the Ethernet adapter (that is, interface) to an external tranceiver

Case History

Bob Metcalfe and Ethernet

As a Ph.D. student at Harvard University in the early 1970s, Bob Metcalfe worked on the ARPAnet at M.I.T. During his studies, he also became exposed to Abramson's work on Aloha and random-access protocols. After completing his Ph.D. and just before beginning a job at Xerox Palo Alto Research Center (Xerox PARC), he visited Abramson and his University of Hawaii colleagues for three months, getting a first-hand look at Alohanet. At Xerox PARC, Metcalfe became exposed to Alto computers, which in many ways were the forerunners of the personal computers of the 1980s. Metcalfe saw the need to network these computers in an inexpensive manner. So armed wth his knowledge about ARPAnet, Alohanet, and random-access protocols, Metcalfe--along with colleague David Boggs--invented Ethernet.

Metcalfe's and Boggs' original Ethernet ran at 2.94 Mbps, and linked up to 256 hosts separated by up to one mile. Metcalfe and Boggs succeeded at getting most of the researchers at Xerox PARC to communicate through their Alto computers. Metcalfe then forged an alliance between Xerox, Digital, and Intel to establish Ethernet as a 10 Mbps Ethernet standard, ratified by the IEEE. Xerox did not show much interest in commercializing Ethernet. In 1979, Metcalfe formed his own company, 3Com, which developed and commercialized networking technology, including Ethernet technology. In particular, 3Com developed and marketed Ethernet cards in the early 1980s for the immensely popular IBM PCs. Metcalfe left 3Com in 1990, when it had two thousand people and $400 milllion dollars in revenue. As of early January 2000, 3Com has a market capitalization of $15 billion and employs thirteen thousand people.

5.5.1: Ethernet Basics

Today's Ethernet comes in many shapes and forms. An Ethernet LAN can have a bus topology or a star topology. An Ethernet LAN can run over coaxial cable, twisted-pair copper wire, or fiber optics. Furthermore, Ethernet can transmit data at different rates, specifically, at 10 Mbps, 100 Mbps, and 1 Gbps. But even though Ethernet comes in many flavors, all of the Ethernet technologies share a few important characteristics. Before examining the different technologies, let's first take a look at the common characteristics.

Ethernet frame structure

Given that there are many different Ethernet technologies on the market today, what do they have in common, what binds them together with a common name? First and foremost is the Ethernet frame structure. All of the Ethernet technologies--whether they use coaxial cable or copper wire, whether they run at 10 Mbps, 100 Mbps or 1 Gbps--use the same frame structure.

The Ethernet frame is shown in Figure 5.24. Once we understand the Ethernet frame, we will already know a lot about Ethernet. To put our discussion of the Ethernet frame in a tangible context, let us consider sending an IP datagram from one host to another host, with both hosts on the same Ethernet LAN. (We note, though, that Ethernet can carry other network-layer packets, as well.) Let the sending adapter, adapter A, have the physical address AA-AA-AA-AA-AA-AA and the receiving adapter, adapter B, have the physical address BB-BB-BB-BB-BB-BB. The sending adapter encapsulates the IP datagram within an Ethernet frame and passes the frame to the physical layer. The receiving adapter receives the frame from the physical layer, extracts the IP datagram, and passes the IP datagram to the network layer. In this context, let us now examine the six fields of the Ethernet frame:

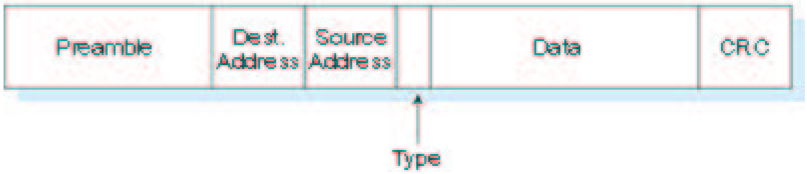


Figure 5.24: Ethernet frame structure

- *Data Field (46 to 1500 bytes).* This field carries the IP datagram. The maximum transfer unit (MTU) of Ethernet is 1500 bytes. This means that if the IP datagram exceeds 1500 bytes, then the host has to fragment the datagram, as discussed in Section 4.4.4. The minimum size of the data field is 46 bytes. This means that if the IP datagram is less than 46 bytes, the data field has to be "stuffed" to fill it out to 46 bytes. When stuffing is used, the data passed to the network layer contains the stuffing as well as an IP datagram. The network layer uses the length field in the IP datagram header to remove the stuffing.

- *Destination Address (6 bytes).* This field contains the LAN address of the destination adapter, namely, BB-BB-BB-BB-BB-BB. When adapter B receives an Ethernet frame with a destination address *other* than its own physical address, BB-BB-BB-BB-BB-BB, or the LAN broadcast address, it discards the frame. Otherwise, it passes the contents of the data field to the network layer.

- *Source Address (6 bytes).* This field contains the LAN address of the adapter that transmits the frame onto the LAN, namely, AA-AA-AA-AA-AA-AA.
- *Type Field (2 bytes).* The type field permits Ethernet to "multiplex" network-layer protocols. To understand this idea, we need to keep in mind that hosts can use other network-layer protocols besides IP. In fact, a given host may support multiple network-layer protocols, and use different protocols for different applications. For this reason, when the Ethernet frame arrives at adapter B, adapter B needs to know to which network-layer protocol it should pass (i.e., demultiplex) the contents of the data field. IP and other data-link layer protocols (for example, Novell IPX or AppleTalk) each have their own, standardized type number. Furthermore, the ARP protocol (discussed in the previous section) has its own type number. Note that the type field is analogous to the protocol field in the network-layer datagram and the port number fields in the transport-layer segment; all of these fields serve to glue a protocol at one layer to a protocol at the layer above.
- *Cyclic Redundancy Check (CRC) (4 bytes).* As discussed in Section 5.2.3, the purpose of the CRC field is to allow the receiving adapter, adapter B, to detect whether any errors have been introduced into the frame, that is, if bits in the frame have been toggled. Causes of bit errors include attenuation in signal strength and ambient electromagnetic energy that leaks into the Ethernet cables and interface cards. Error detection is performed as follows. When host A constructs the Ethernet frame, it calculates a CRC field, which is obtained from a mapping of the other bits in frame (except for the preamble bits). When host B receives the frame, it applies the same mapping to the frame and checks to see if the result of the mapping is equal to what is in the CRC field. This operation at the receiving host is called the **CRC check.** If the CRC check fails (that is, if the result of the mapping does not equal the contents of the CRC field), then host B knows that there is an error in the frame.
- *Preamble (8 bytes).* The Ethernet frame begins with an eight-byte preamble field. Each of the first seven bytes of the preamble has a value of 10101010; the last byte is 10101011. The first seven bytes of the preamble serve to "wake up" the receiving adapters and to synchronize their clocks to that of the sender's clock. Why should the clocks be out of synchronization? Keep in mind that adapter A

aims to transmit the frame at 10 Mbps, 100 Mbps, or 1 Gbps, depending on the type of Ethernet LAN. However, because nothing is absolutely perfect, adapter A will not transmit the frame at exactly the target rate; there will always be some *drift* from the target rate, a drift which is not known *a priori* by the other adapters on the LAN. A receiving adapter can lock onto adapter A’s clock by simply locking onto the bits in the first seven bytes of the preamble. The last two bits of the eighth byte of the preamble (the first two consecutive 1s) alert adapter B that the "important stuff" is about to come. When host B sees the two consecutive 1s, it knows that the next six bytes are the destination address. An adapter can tell when a frame ends by simply detecting absence of current.

An unreliable connectionless service

All of the Ethernet technologies provide **connectionless service** to the network layer. That is, when adapter A wants to send a datagram to adapter B, adapter A encapsulates the datagram in an Ethernet frame and sends the frame into the LAN, without first "handshaking" with adapter B. This layer-2 connectionless service is analogous to IP’s layer-3 datagram service and UDP’s layer-4 connectionless service.

All of the Ethernet technologies provide an **unreliable service** to the network layer. In particular, when adapter B receives a frame from A, adapter B does not send an acknowledgment when a frame passes the CRC check (nor does it send a negative acknowledgment when a frame fails the CRC check). Adapter A hasn’t the slightest idea whether its transmitted frame was received correctly or incorrectly. When a frame fails the CRC check, adapter B simply discards the frame. This lack of reliable transport (at the link layer) helps to make Ethernet simple and cheap. But it also means that the stream of datagrams passed to the network layer can have gaps.

If there are gaps due to discarded Ethernet frames, does the application-layer protocol at host B see gaps as well? As we learned in Chapter 3, this solely depends on whether the application is using UDP or TCP. If the application is using UDP, then the application-layer protocol in host B will indeed suffer from gaps in the data. On the other hand, if the application is using TCP, then TCP in host B will not acknowledge the discarded data, causing TCP in host A to retransmit. Note that when TCP retransmits data, the data will eventually find its way to the Ethernet adapter at which it was discarded. And in this sense, Ethernet eventually retransmits the data as well. But we should keep in mind that Ethernet doesn’t know that it is retransmitting. Ethernet thinks it is receiving a brand new datagram with brand new data, even though this datagram contains data that has already been transmitted at least once.

Baseband transmission and manchester encoding

Ethernet uses baseband transmission, that is, the adapter sends a digital signal directly into the broadcast channel. The interface card does not shift the signal into another frequency band, as is done in ADSL and cable modem systems. Ethernet also uses Manchester encoding, as shown in Figure 5.25. With Manchester encoding, each bit contains a transition; a 1 has a transition from up to down, whereas a 0 has a transition from down to up. The reason for Manchester encoding is that the clocks in the sending and receiving adapters are not perfectly synchronized. By including a transition in the middle of each bit, the receiving host can synchronize its clock to that of the sending host. Once the receiving adapter's clock is synchronized, the receiver can delineate each bit and determine whether it is a 1 or 0. Manchester encoding is a physical layer operation rather than a link-layer operation; however, we have briefly described it here as it is used extensively in Ethernet.

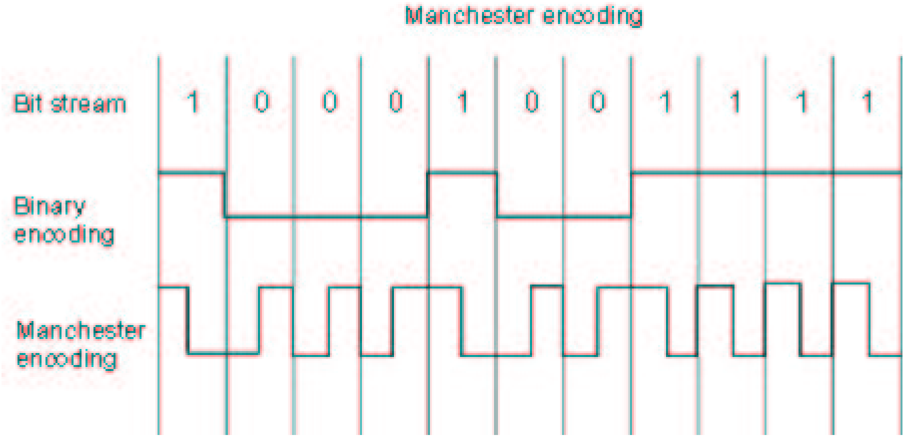


Figure 5.25: Manchester encoding

5.5.2: CSMA/CD: Ethernet's Multiple Access Protocol

Nodes in an Ethernet LAN are interconnected by a broadcast channel, so that when an adapter transmits a frame, all the adapters on the LAN receive the frame. As we mentioned in Section 5.3, Ethernet uses a CSMA/CD multiple access algorithm. Summarizing our discussion from Section 5.3, recall that CSMA/CD employs the following mechanisms:

1. An adapter may begin to transmit at any time, that is, no slots are used.
2. An adapter never transmits a frame when it senses that some other adapter is transmitting, that is, it uses carrier-sensing.

3. A transmitting adapter aborts its transmission as soon as it detects that another adapter is also transmitting, that is, it uses collision detection.

4. Before attempting a retransmission, an adapter waits a random time that is typically small compared to a frame time.

These mechanisms give CSMA/CD much better performance than slotted ALOHA in a LAN environment. In fact, if the maximum propagation delay between stations is very small, the efficiency of CSMA/CD can approach 100%. But note that the second and third mechanisms listed above require each Ethernet adapter to be able to (1) sense when some other adapter is transmitting, and (2) detect a collision while it is transmitting. Ethernet adapters perform these two tasks by measuring voltage levels before and during transmission.

Each adapter runs the CSMA/CD protocol without explicit coordination with the other adapters on the Ethernet. Within a specific adapter, the CSMA/CD protocol works as follows:

1. The adapter obtains a network-layer PDU from its parent node, prepares an Ethernet frame, and puts the frame in an adapter buffer.

2. If the adapter senses that the channel is idle (that is, there is no signal energy from the channel entering the adapter), it starts to transmit the frame. If the adapter senses that the channel is busy, it waits until it senses no signal energy (plus 96 bit times) and then starts to transmit the frame.

3. While transmitting, the adapter monitors for the presence of signal energy coming from other adapters. If the adapter transmits the entire frame without detecting signal energy from other adapters, the adapter is done with the frame.

4. If the adapter detects signal energy from other adapters while transmitting, it stops transmitting its frame and instead transmits a 48-bit jam signal.

5. After aborting (that is, transmitting the jam signal), the adapter enters an **exponential backoff** phase. Specifically, when transmitting a given frame, after experiencing the *n*th collision in a row for this frame, the adapter chooses a value for *K* at random from {0,1,2, . . ., 2^m - 1} where *m:* = min(*n*,10). The adapter then waits *K* • 512 bit times and then returns to Step 2.

A few comments about the CSMA/CD protocol are certainly in order. The purpose of the jam signal is to make sure that all other transmitting adapters become aware of the collision. Let’s look at an example. Suppose adapter A begins to transmit a frame, and just before A’s signal reaches adapter B, adapter B begins to transmit. So B will have transmitted only a few bits when it aborts its transmission. These few bits will indeed propagate to A, but they may not constitute enough energy for A to detect the collision. To make sure that A detects the collision (so that it too can also abort), B transmits the 48-bit jam signal.

Next consider the exponential backoff algorithm. The first thing to notice here is that a bit time (that is, the time to transmit a single bit) is very short; for a 10 Mbps Ethernet, a bit time is 0.1 microsecond. Now let’s look at an example. Suppose that an adapter attempts for the first time to transmit a frame, and while transmitting it detects a collision. The adapter then chooses $K = 0$ with probability 0.5 and chooses $K = 1$ with probability 0.5. If the adapter chooses $K = 0$, then it immediately jumps to Step 2 after transmitting the jam signal. If the adapter chooses $K = 1$, it waits 51.2 microseconds before returning to Step 2. After a second collision, K is chosen with equal probability from {0,1,2,3}. After three collisions, K is chosen with equal probability from {0,1,2,3,4,5,6,7}. After ten or more collisions, K is chosen with equal probability from {0,1,2, . . ., 1023}. Thus the size of the sets from which K is chosen grows exponentially with the number of collisions (until $n = 10$); it is for this reason that Ethernet’s backoff algorithm is referred to as "exponential backoff."

The Ethernet standard imposes limits on the distance between any two nodes. These limits ensure that if adapter A chooses a lower value of K than all the other adapters involved in a collision, then adapter A will be able to transmit its frame without experiencing a new collision. We will explore this property in more detail in the homework problems.

Why use exponential backoff? Why not, for example, select K from {0,1,2,3,4,5,6,7} after every collision? The reason is that when an adapter experiences its first collision, it has no idea how many adapters are involved in the collision. If there are only a small number of colliding adapters, it makes sense to choose K from a small set of small values. On the other hand, if many adapters are involved in the collision, it makes sense to choose K from a larger, more dispersed set of values (why?). By increasing the size of the set after each collision, the adapter appropriately adapts to these different scenarios.

We also note here that each time an adapter prepares a new frame for transmission, it runs the CSMA/CD algorithm presented above. In particular, the adapter does not take into account any collisions that may have occurred in the recent past. So it is possible that an adapter with a new frame will be able to immediately sneak in a successful transmission while several other adapters are in the exponential backoff state.

Ethernet efficiency

When only one node has a frame to send, the node can transmit at the full rate of the Ethernet technology (either 10 Mbps, 100 Mbps, or 1 Gbps). However, if many nodes have frames to transmit, the effective transmission rate of the channel can be much less. We define the **efficiency of Ethernet** to be the long-run fraction of time during which frames are being transmitted on the channel without collisions when there is a large number of active nodes, with each node having a large number of frames to send. In order to present a closed-form approximation of the efficiency of Ethernet, let t_{prop} denote the maximum time it takes signal energy to propagate between any two adapters. Let t_{trans} be the time to transmit a maximum size Ethernet frame (approximately 1.2 msecs for a 10 Mbps Ethernet). A derivation of the efficiency of Ethernet is beyond the scope of this book (see [Lam 1980] and [Bertsekas 1991]). Here we simply state the following approximation:

$$\text{Efficiency} = \frac{1}{1 + 5t_{prop}/t_{trans}}$$

We see from this formula that as t_{prop} approaches 0, the efficiency approaches 1. This matches our intuition that if the propagation delay is zero, colliding nodes will abort immediately without wasting the channel. Also, as t_{trans} becomes very large, efficiency approaches 1. This is also intuitive because when a frame grabs the channel, it will hold on to the channel for a very long time; thus the channel will be doing productive work most of the time.

5.5.3: Ethernet Technologies

The most common Ethernet technologies today are 10Base2, which uses thin coaxial cable in a bus topology and has a transmission rate of 10 Mbps; 10BaseT, which uses twisted-pair copper wire in a star topology and has a transmission rate of 10 Mbps; 100BaseT, which typically uses twisted-pair copper wire in a star topology and has a transmission rate of 100 Mbps; and Gigabit Ethernet, which uses both fiber and twisted-pair copper wire and transmits at a rate of 1 Gbps. These Ethernet technologies are standardized by the IEEE 802.3 working groups. For this reason, an Ethernet LAN is often referred to as an 802.3 LAN.

Before discussing specific Ethernet technologies, we need to discuss **repeaters,** which are commonly used in LANs as well as in longer-distance wide-area links. A repeater is a physical-layer device that acts on individual bits rather than on frames. It has two or more interfaces. When a bit, representing a zero or a one, arrives from one interface, the repeater simply recreates the bit, boosts its energy strength, and transmits the bit onto all the other interfaces. Repeaters are commonly used in LANs in order to extend their geographical range. When used with Ethernet, it is important to keep in mind that repeaters do not implement carrier sensing or any other part of CSMA/CD; a repeater repeats an incoming bit on all outgoing interfaces even if there is signal energy on some of the interfaces.

10Base2 Ethernet

10Base2 is a very popular Ethernet technology. If you look at how your computer (at work or at school) is connected to the network, it is very possible you will see a 10Base2 connection. The "10" in 10Base2 stands for "10 Mbps"; the "2" stands for "200 meters," which is the approximate maximum distance between any two nodes without repeaters between them. (The actual maximum distance is 185 meters.) A 10Base2 Ethernet is shown in Figure 5.26.

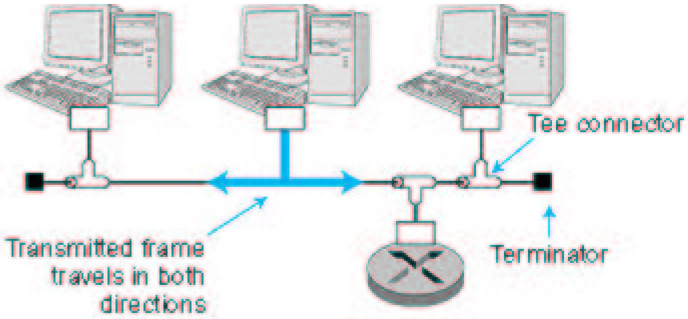


Figure 5.26: A 10Base Ethernet

We see from Figure 5.26 that 10Base2 uses a bus topology; that is, nodes are connected (through their adapters) in a linear fashion. The physical medium used to connect the nodes is **thin coaxial cable,** which is similar to what is used in cable TV, but with a thinner and lighter cable. When an adapter transmits a frame, the frame passes through a "tee connector"; two copies of the frame leave the tee connector, one copy going in one direction and one copy in the other direction. As the frames travel toward the terminators, they leave a copy at every node they pass. (More precisely, as a bit passes in front of a node, part of the energy of the bit leaks into the adapter.) When the frame finally reaches a terminator, it gets absorbed by the terminator. Note that when an adapter transmits a frame, the frame is received by every other adapter on the Ethernet. Thus, 10Base2 is indeed a broadcast technology.

Suppose you want to connect a dozen PCs in your office using 10Base2 Ethernet. To do this, you would need to purchase 12 Ethernet cards with thin Ethernet ports; 12 BNC tees, which are small metallic objects that attach to the adapters (less than one dollar each); a dozen or so thin coax segments, 5-20 meters each;

and two "terminators," which you put at the two ends of the bus. The cost of the whole network, including adapters, is likely to be less than the cost of a single PC! Because 10Base2 is incredibly inexpensive, it is often referred to as "cheapnet."

Without a repeater, the maximum length of a 10Base2 bus is 185 meters. If the bus becomes any longer, signal attenuation can cause the system to malfunction. Also, without a repeater, the maximum number of nodes is 30, as each node contributes to signal attenuation. Repeaters can be used to connect 10Base2 segments in a linear fashion, with each segment having up to 30 nodes and having a length up to 185 meters. Up to four repeaters can be included in a 10Base2 Ethernet, which creates up to five "segments." Thus a 10Base2 Ethernet bus can have a total length of 985 meters and support up to 150 nodes. Note that the CSMA/CD access protocol is completely oblivious to the repeaters; if any two of 150 nodes transmit at the same time, there will be a collision.

10BaseT and 100BaseT

We discuss 10BaseT and100BaseT Ethernet together, as they are similar technologies. The most important difference between them is that 10BaseT transmits at 10 Mbps and 100BaseT Ethernet transmits at 100 Mbps. 100BaseT is also commonly called "fast Ethernet" and "100 Mbps Ethernet." 10BaseT and 100BaseT are also very popular Ethernet technologies; in fact, for new installations, 10BaseT and 100BaseT Ethernet are often today the technology of choice. Both 10BaseT and 100BaseT Ethernet use a star topology, as shown in Figure 5.27.

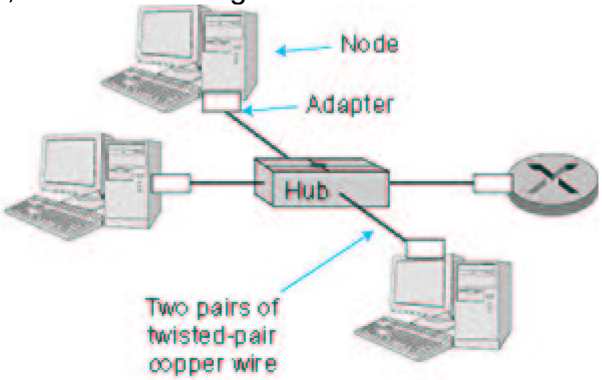


Figure 5.27: Star topology for 10BaseT and 100BaseT

In the star topology there is a central device called a **hub** (also sometimes called a concentrator.) Each adapter on each node has a direct, point-to-point connection to the hub. This connection consists of two pairs of twisted-pair copper wire, one for transmitting and the other for receiving. At each end of the connection there is a connector that resembles the RJ-45 connector used for ordinary telephones. The "T" in 10BaseT and 100BaseT stands for "twisted pair." For both 10BaseT and

100BaseT, the maximum length of the connection between an adapter and the hub is 100 meters; the maximum length between any two nodes is thus 200 meters. As we will discuss in the next section, this maximum distance can be increased by using tiers of hubs, bridges, switches, and fiber links.

In essence, a hub is a repeater: when it receives a bit from an adapter, it sends the bit to all the other adapters. In this manner, each adapter can (1) sense the channel to determine if it is idle, and (2) detect a collision while it is transmitting. But hubs are popular because they also provide network management features. For example, if an adapter malfunctions and continually sends Ethernet frames (a so-called jabbering adapter), then a 10Base2 Ethernet will become totally dysfunctional; none of the nodes will be able to communicate. But a 10BaseT network will continue to function, because the hub will detect the problem and internally disconnect the malfunctioning adapter. With this feature, the network administrator doesn’t have to get out of bed and drive back to work in order to correct the problem. Also, most hubs can gather information and report the information to a host that connects directly to the hub. This monitoring host provides a graphical interface that displays statistics and graphs, such as bandwidth usage, collision rates, average frame sizes, etc. Network administrators can use this information not only to debug and correct problems, but also to plan how the LAN should evolve in the future.

Many Ethernet adapters today are 10/100 Mbps adapters. This means that they can be used for both 10BaseT and 100BaseT Ethernets. 100BaseT, which typically uses category-5 twisted pair (a high-quality twisted pair of wires with many twists). Unlike the 10Base2 and 10BaseT, 100BaseT does not use Manchester encoding, but instead a more efficient encoding called 4B5B: every group of five clock periods is used to send four bits in order to provide enough transitions to allow clock synchronization.

We briefly mention at this point that both 10 Mbps and 100 Mbps Ethernet technologies can employ fiber links. A fiber link is often used to interconnect to hubs that are in different buildings on the same campus. Fiber is expensive because of the cost of its connectors, but it has excellent noise immunity. The IEEE 802 standards permit a LAN to have a larger geographical reach when fiber is used to connect backbone nodes.

Gigabit Ethernet

Gigabit Ethernet is an extension to the highly successful 10 Mbps and 100 Mbps Ethernet standards. Offering a raw data rate of 1,000 Mbps, Gigabit Ethernet maintains full compatibility with the huge installed base of Ethernet equipment. The standard for Gigabit Ethernet, referred to as IEEE 802.3z, does the following:

- Uses the standard Ethernet frame format (Figure 5.24), and is backward compatible with 10BaseT and 100BaseT technologies. This allows for easy integration of Gigabit Ethernet with the existing installed base of Ethernet equipment.
- Allows for point-to-point links as well as shared broadcast channels. Point-to-point links use switches (see Section 5.6) whereas broadcast channels use hubs, as described above for 10BaseT and 100 BaseT. In Gigabit Ethernet jargon, hubs are called "buffered distributors."
- Uses CSMA/CD for shared broadcast channels. In order to have acceptable efficiency, the maximum distance between nodes must be severely restricted.
- Allows for full-duplex operation at 1,000 Mbps in both directions for point-to-point channels.

Like 10BaseT and 100BaseT, Gigabit Ethernet has a star topology with a hub or switch at its center. (Ethernet switches will be discussed in Section 5.6.) Gigabit Ethernet often serves as a backbone for interconnecting multiple 10 Mbps and 100 Mbps Ethernet LANs. Initially operating over optical fiber, Gigabit Ethernet will also be able to use Category 5 UTP cabling.

The Gigabit Ethernet Alliance is an open forum whose purpose is to promote industry cooperation in the development of Gigabit Ethernet. Their Web site is a rich source of information on Gigabit Ethernet [Alliance 1999]. The Interoperability Lab at the University of New Hampshire also maintains a nice page on Gigabit Ethernet [Interoperability 1999].

Online Book

5.6: Hubs, Bridges, and Switches

Institutions--including companies, universities, and high schools--typically consist of many departments, with each department having and managing its own Ethernet LAN. Naturally, an institution will want its departments to interconnect their departmental LAN segments. In this section, we consider a number of different approaches with which LANs can be connected together. We'll cover three approaches--hubs, bridges, and switches--in the following subsections. All three of these approaches are in widespread use today.

5.6.1: Hubs

The simplest way to interconnect LANs is to use a hub. A **hub** is a simple device that takes an input (that is, a frame's bits) and retransmits the input on the hub's outgoing ports. Hubs are essentially repeaters, operating on bits. They are thus physical-layer devices. When a bit comes into a hub interface, the hub simply broadcasts the bit on all the other interfaces. Figure 5.28 shows how three academic departments in a university might interconnect their LANs. In this figure, each of the three departments has a 10BaseT Ethernet that provides network access to the faculty, staff, and students of the departments. Each host in a department has a point-to-point connection to the departmental hub. A fourth hub, called a **backbone hub,** has point-to-point connections to the departmental hubs, interconnecting the LANs of the three departments. The design shown in Figure 5.28 is a **multi-tier hub design** because the hubs are arranged in a hierarchy. It is also possible to create multi-tier designs with more than two tiers--for example, one tier for the departments, one tier for the schools within the university (for example, engineering school, business school, etc.) and one tier at the highest university level. Multiple tiers can also be created out of 10Base2 (bus topology Ethernets) with repeaters.

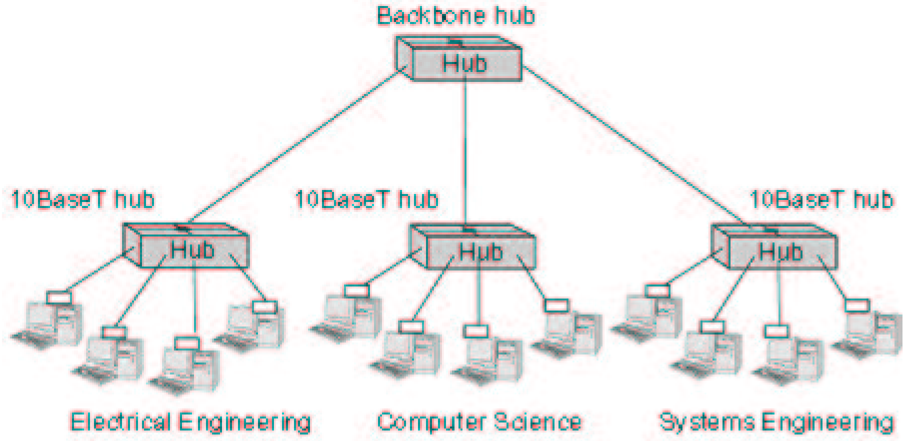


Figure 5.28: Three departmental Ethernets interconnected with a hub

In a multi-tier design, we refer to the entire interconnected network as a LAN, and we refer to each of the departmental portions of the LAN (that is, the departmental hub and the hosts that connect to the hub) as a **LAN segment.** It is important to note that all of the LAN segments in Figure 5.28 belong to the same **collision domain,** that is, whenever two or more nodes on the LAN segments transmit at the same time, there will be a collision and all of the transmitting nodes will enter exponential backoff. Interconnecting departmental LANs with a backbone hub has many benefits. First and foremost, it provides interdepartmental communication among the hosts in the various departments. Second, it extends the maximum distance between any pair of nodes on the LAN. For example, with 10BaseT the maximum distance between a node and its hub is 100 meters; therefore, in a single LAN segment the maximum distance between any pair of nodes is 200 meters. By interconnecting the hubs, this maximum distance can be extended, since the distance between directly connected hubs can also be 100 meters when using twisted pair (and more when using fiber). A third benefit is that the multi-tier design provides a degree of graceful degradation. Specifically, if any one of the departmental hubs starts to malfunction, the backbone hub can detect the problem and disconnect the departmental hub from the LAN; in this manner, the remaining departments can continue to operate and communicate while the faulty departmental hub gets repaired.

Although a backbone hub is a useful interconnection device, it has three serious limitations that hinder its deployment. First, and perhaps more important, when departmental LANs are interconnected with a hub (or a repeater), then the independent collision domains of the departments are transformed into one large, common collision domain. Let us explore this issue in the context of Figure 5.28. Before interconnecting the three departments, each departmental LAN had a maximum throughput of 10 Mbps, so that maximum aggregate throughput of the three LANs was 30 Mbps. But once the three LANs are interconnected with a hub, all of the hosts in the three departments belong to the same collision domain, and the maximum aggregate throughput is reduced to 10 Mbps.

A second limitation is that if the various departments use different Ethernet technologies, then it may not be possible to interconnect the departmental hubs with a backbone hub. For example, if some departments use 10BaseT and the remaining departments use 100BaseT, then it is impossible to interconnect all the departments without some frame buffering at the interconnection point; since hubs are essentially repeaters and do not buffer frames, they cannot interconnect LAN segments operating at different rates.

A third limitation is that each of the Ethernet technologies (10Base2, 10BaseT, 100BaseT, and so on) has restrictions on the maximum allowable number of nodes in a collision domain, the maximum distance between two hosts in a collision domain, and the maximum allowable number of tiers in a multi-tier design. These restrictions constrain both the total number of hosts that can connect to a multi-tier LAN as well as the geographical reach of the multi-tier LAN.

5.6.2: Bridges

In contrast to hubs, which are physical-level devices, bridges operate on Ethernet frames and thus are layer-2 devices. In fact, **bridges** are full-fledged packet switches that forward and filter frames using the LAN destination addresses. When a frame comes into a bridge interface, the bridge does not just copy the frame onto all of the other interfaces. Instead, the bridge examines the layer-2 destination address of the frame and attempts to forward the frame on the interface that leads to the destination. Figure 5.29 shows how the three academic departments of our previous example might be interconnected with a bridge. The three numbers next to the bridge are the interface numbers for the three bridge interfaces. When the departments are interconnected by a bridge, as in Figure 5.29, we again refer to the entire interconnected network as a LAN, and we again refer to each of the departmental portions of the network as LAN segments. But in contrast to the multi-tier hub design in Figure 5.28, each LAN segment is now an isolated collision domain.

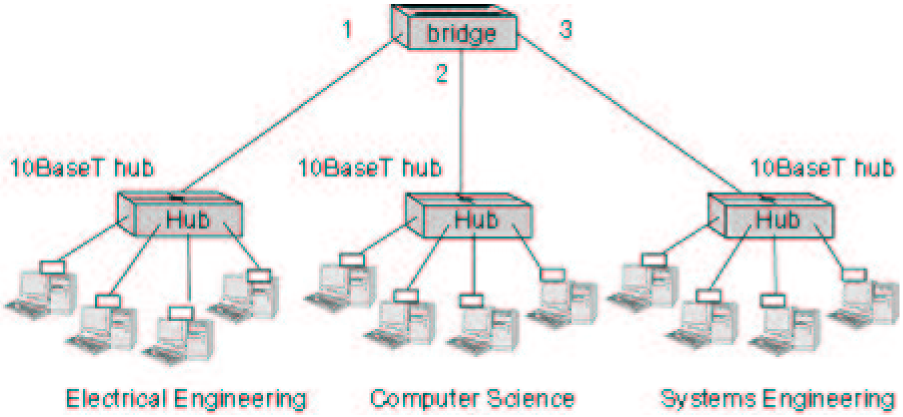


Figure 5.29: Three departmental LANs interconnected with a bridge

Bridges can overcome many of the problems that plague hubs. First, bridges permit interdepartmental communication while preserving isolated collision domains for each of the departments. Second, bridges can interconnect different LAN technologies, including 10 Mbps and 100 Mbps Ethernets. Third, there is no limit to how large a LAN can be when bridges are used to interconnect LAN segments; in theory, using bridges, it is possible to build a LAN that spans the entire globe.

Bridge forwarding and filtering

Filtering is the ability of a bridge to determine whether a frame should be forwarded to some interface or should just be dropped. **Forwarding** is the ability to determine the interfaces to which a frame should be directed. Bridge filtering and forwarding are done with a **bridge table.** The bridge table contains entries for some, but not necessarily all, of the nodes on a LAN. A node’s entry in the bridge table contains (1) the LAN address of the node, (2) the bridge interface that leads toward the node, (3) and the time at which the entry for the node was placed in the table. An example bridge table for the LAN in Figure 5.29 is shown in Figure 5.30. Although this description of frame forwarding may sound similar to our discussion of datagram forwarding in Chapter 4, we’ll see shortly that there are important differences. We note here that the addresses used by bridges are physical addresses rather than network-layer addresses. We will also see shortly that a bridge table is constructed in a very different manner than routing tables.

Address	Interface	Time
62-FE-F7-11-89-A3	1	9:32

7C-BA-B2-B4-91-10
3
9:36

....
....
....

Figure 5.30: Portion of a bridge table for the LAN in Figure 5.29

To understand how bridge filtering and forwarding works, suppose a frame with destination address DD-DD-DD-DD-DD-DD arrives to the bridge on interface *x*. The bridge indexes its table with the LAN address DD-DD-DD-DD-DD-DD and finds its corresponding interface *y* that is known to lead to destinator address DD-DD-DD-DD-DD-DD. We'll see shortly what happens if such an interface, *y* is not found in the table.

- If *x* equals *y*, then the frame is coming from a LAN segment that contains adapter DD-DD-DD-DD-DD-DD. There being no need to forward the frame to any of the other interfaces, the bridge performs the filtering function by discarding the frame.

- If *x* does not equal *y*, then the frame needs to be routed to the LAN segment attached to interface *y*. The bridge performs its forwarding function by putting the frame in an output buffer that precedes interface *y*.

These simple rules allow a bridge to preserve separate collision domains for each of the different LAN segments connected to its interfaces. The rules also allow two sets of nodes on different LAN segments to communicate simultaneously without interfering with each other.

Let's walk through these rules for the network in Figure 5.29 and its bridge table in Figure 5.30. Suppose that a frame with destination address 62-FE-F7-11-89-A3 arrives to the bridge from interface 1. The bridge examines its table and sees that the destination is on the LAN segment connected to interface 1 (that is, the Electrical Engineering LAN). This means that the frame has already been broadcast on the LAN segment that contains the destination. The bridge therefore filters (that is, discards) the frame. Now suppose a frame with the same destination address arrives from interface 2. The bridge again examines its table and sees that the destination is the direction of interface 1; it therefore forwards the frame to the output buffer preceding interface 1. It should be clear from this example that as long as the bridge table is complete and accurate, the bridge isolates the departmental collision domains while permitting the departments to communicate.

Recall that when a hub (or a repeater) forwards a frame onto a link, it just sends the bits onto the link without bothering to sense whether another transmission is currently taking place on the link. In contrast, when a bridge wants to forward a frame onto a link, it runs the CSMA/CD algorithm discussed in Section 5.3. In particular, the bridge refrains from transmitting if it senses that some other node on the LAN segment into which it wants to send a frame is transmitting; furthermore, the bridge uses exponential backoff when one of its transmissions results in a collision. Thus bridge interfaces behave very much like node adapters. But, technically speaking, they are *not* node adapters because neither a bridge nor its interfaces have LAN addresses. Recall that a node adapter always inserts its LAN address into the source address of every frame it transmits. This statement is true for router adapters as well as host adapters. A bridge, on the other hand, does not change the source address of the frame.

One significant feature of bridges is that they can be used to combine Ethernet segments using different Ethernet technologies. For example, if in Figure 5.29, Electrical Engineering has a 10Base2 Ethernet, Computer Science has a 100BaseT Ethernet, and Systems Engineering has a 10BaseT Ethernet, then a bridge can be purchased that can interconnect the three LANs. With Gigabit Ethernet bridges, it is possible to have an additional 1 Gbps connection to a router, which in turn connects to a larger university network. As we mentioned earlier, this feature of being able to interconnect different link rates is not available with hubs.

Also, when bridges are used as interconnection devices, there is no theoretical limit to the geographical reach of a LAN. In theory, we can build a LAN that spans the globe by interconnecting hubs in a long, linear topology, with each pair of neighboring hubs interconnected by a bridge. With this design, each of the hubs has its own collision domain, and there is no limit on how long the LAN can be. We shall see shortly, however, that it is undesirable to build very large networks exclusively using bridges as interconnection devices--large networks need routers as well.

Self-learning

A bridge has the wonderful property (particularly for the already-overworked network administrator) that its table is built automatically, dynamically, and autonomously--without any intervention from a network administrator or from a configuration protocol. In other words, bridges are **self-learning.** This capability is accomplished as follows.

1. The bridge table is initially empty.

2. When a frame arrives on one of the interfaces and the frame’s destination address is not in the table, then the bridge forwards copies of the frame to the output buffers of all of the other interfaces. (At each of these other interfaces, the frame is transmitted into that LAN segment using CSMA/CD.)

3. For each frame received, the bridge stores in its table (1) the LAN address in the frame’s *source address field,* (2) the interface from which the frame arrived, (3) the current time. In this manner the bridge records in its table the LAN segment on which the sending node resides. If every node in the LAN eventually sends a frame,

then every node will eventually get recorded in the table.

4. When a frame arrives on one of the interfaces and the frame's destination address is in the table, then the bridge forwards the frame to the appropriate interface.

5. The bridge deletes an address in the table if no frames are received with that address as the source address after some period of time (the *aging time*). In this manner, if a PC is replaced by another PC (with a different adapter), the LAN address of the original PC will eventually be purged from the bridge table.

Let's walk through the self-learning property for the network in Figures 5.29 and its corresponding bridge table in Figure 5.30. Suppose at time 9:39 a frame with source address 01-12-23-34-45-56 arrives from interface 2. Suppose that this address is not in the bridge table. Then the bridge appends a new entry in the table, as shown in Figure 5.31.

Address	Interface	Time
01-12-23-34-45-56	2	9:39
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
....	....	....

Figure 5.31: Bridge learns about the location of an adapter with address 01-12-23-34-45-56

Continuing with this same example, suppose that the aging time for this bridge is 60 minutes and no frames with source address 62-FE-F7-11-89-A3 arrive to the bridge between 9:32 and 10:32. Then at time 10:32, the bridge removes this address from its table.

Bridges are **plug-and-play devices** because they require no intervention from a network administrator or user. A network administrator wanting to install a bridge need do nothing more than connect the LAN segments to the bridge interfaces. The administrator need not configure the bridge tables at the time of installation or when a host is removed from one of the LAN segments. Because bridges are plug-and-play, they are also referred to as **transparent bridges.**

Spanning tree

One of the problems with a pure hierarchical design for interconnected LAN segments is that if a hub or a bridge near the top of the hierarchy fails, then pieces of the LAN will become disconnected. For this reason it is desirable to build networks with multiple paths between LAN segments. An example of such a network is shown in Figure 5.32.

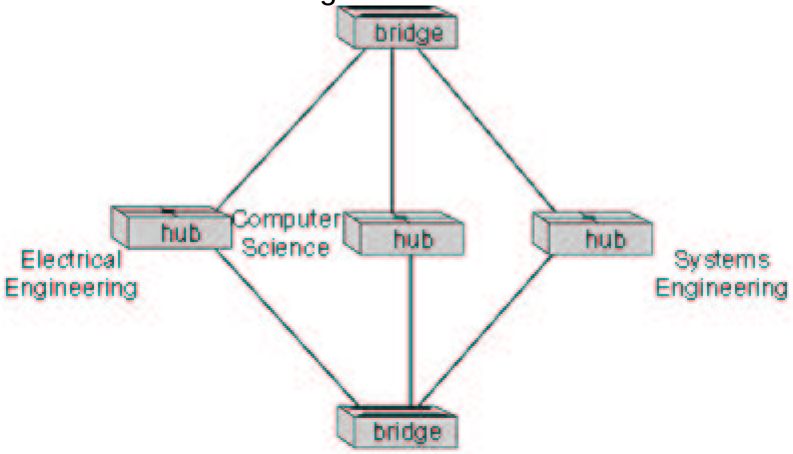


Figure 5.32: Interconnected LAN segments with redundant paths

Multiple redundant paths between LAN segments (such as departmental LANs) can greatly improve fault tolerance. But, unfortunately, multiple paths have a serious side effect--frames can cycle and multiply within the interconnected LAN, unless care is taken [Perlman 1999]. To see this, suppose that the bridge tables in Figure 5.32 are empty, and a host in Electrical Engineering sends a frame to a host in Computer Science. When the frame arrives to the Electrical Engineering hub, the hub will generate two copies of the frame and send one copy to each of the two bridges. When each of the bridges receives the frame, it will generate two copies, send one copy to the Computer Science hub and the other copy to the Systems Engineering hub. Since both bridges do this, there will be four identical frames in the LAN. This multiplying of copies could continue indefinitely if the bridges do not know where the destination host resides. (Recall that for the destination host's LAN address to appear in the forwarding table, the destination host must first generate a frame so that its address can be recorded in the bridge tables.) The number of copies of the original frame grows exponentially fast, crashing the entire network.

To prevent the cycling and multiplying of frames, bridges use a spanning tree protocol [Perlman 1999]. In the **spanning tree protocol,** bridges communicate with each other over the LANs in order to determine a spanning tree, that is, a subset of the original topology that has no loops. Once the bridges determine a spanning tree, the bridges disconnect appropriate interfaces in order to create the spanning tree out of the original topology. For example, in Figure 5.32, a spanning tree is created by having the top bridge disconnect its interface to Electrical Engineering and the bottom bridge disconnect its interface to Systems Engineering. With the interfaces disconnected and the loops removed, frames will no longer cycle and multiply. If, at some later time, one of links in the spanning tree fails, the bridges can reconnect the interfaces, run the spanning tree algorithm again, and determine a new set of interfaces that should be disconnected.

Bridges versus routers

As we learned in Chapter 4, routers are store-and-forward packet switches that forward packets using network-layer addresses. Although a bridge is also a store-and-forward packet switch, it is fundamentally different from a router in that it forwards packets using LAN addresses. Whereas a router is a layer 3 packet switch, a bridge is a layer-2 packet switch.

Even though bridges and routers are fundamentally different, network administrators must often choose between them when installing an interconnection device. For example, for the network in Figure 5.29, the network administrator could have just as easily used a router instead of a bridge. Indeed, a router would have also kept the three collision domains separate while permitting interdepartmental communication. Given that both bridges and routers are candidates for interconnection devices, what are the pros and cons of the two approaches?

First consider the pros and cons of bridges. As mentioned above, bridges are plug and play, a property that is cherished by all the overworked network administrators of the world. Bridges can also have relatively high packet filtering and forwarding rates--as shown in Figure 5.33, bridges only have to process packets up through layer 2, whereas routers have to process frames up through layer 3. On the other hand, the spanning tree protocol restricts the effective topology of a bridged network to a spanning tree. This means that all frames must flow along the spanning tree, even when there are more direct (but disconnected) paths between source and destination. The spanning tree restriction also concentrates the traffic on the spanning tree links when it could have otherwise been spread among all the links of the original topology. Furthermore, bridges do not offer any protection against broadcast storms--if one host goes haywire and transmits an endless stream of Ethernet broadcast frames, the bridges will forward all of these frames, causing the entire network to collapse.

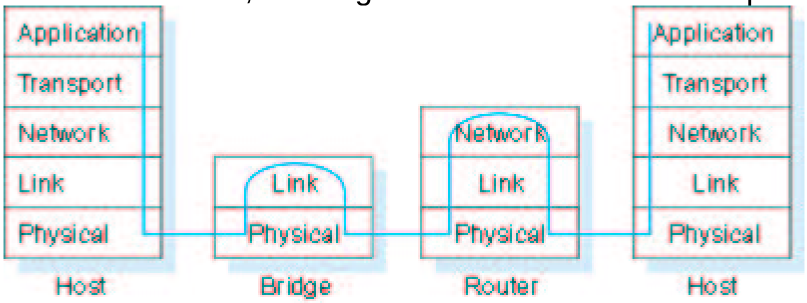


Figure 5.33: Packet processing in bridges, routers, and hosts

Now consider the pros and cons of routers. Because network addressing is often hierarchical (and not flat as is LAN addressing), packets do not normally cycle through routers even when the network has redundant paths. (Actually, packets can cycle when router tables are misconfigured; but as we learned in Chapter 4, IP uses a special datagram header field to limit the cycling.) Thus, packets are not restricted to a spanning tree and can use the best path between source and destination. Because routers do not have the spanning tree restriction, they have allowed the Internet to be built with a rich topology that includes, for example, multiple active links between Europe and North America. Another feature of routers is that they provide firewall protection against layer 2 broadcast storms. Perhaps the most significant drawback of routers, though, is that they are not plug and play--they and the hosts that connect to them need their IP addresses to be configured. Also, routers often have a larger per-packet processing time than bridges, because they have to process up through the layer 3 fields. Finally, there are two different ways to pronounce the word "router," either as "rootor" or as "rowter," and people waste a lot of time arguing over the proper pronunciation [Perlman 1999].

Given that both bridges and routers have their pros and cons, when should an institutional network (for example, university campus network or a corporate campus network) use bridges, and when should it use routers? Typically, small networks consisting of a few hundred hosts have a few LAN segments. Bridges suffice for these small networks, as they localize traffic and increase aggregate throughput without requiring any configuration of IP addresses. But larger networks consisting of thousands of hosts typically include routers within the network (in addition to bridges). The routers provide a more robust isolation of traffic, control broadcast storms, and use more "intelligent" routes among the hosts in the network.

Connecting LAN segments with backbones

Consider once again the problem of interconnecting the Ethernets in the three departments in Figure 5.29 with bridges. An alternative design is shown in Figure 5.34. This alternative design uses two two-interface bridges (that is, bridges with two interfaces), with one bridge connecting Electrical Engineering to Computer Science, and the other bridge connecting Computer Science to Systems Engineering. Although two-interface bridges are very popular due to their low cost and simplicity, the design in Figure 5.34 is *not recommended.* There are two reasons. First, if the Computer Science hub were to fail, then Electrical Engineering and Systems Engineering would no longer be able to communicate. Second, and more important, all the interdepartmental traffic between Electrical and Systems Engineering has to pass through Computer Science, which may overly burden the Computer Science LAN segment.

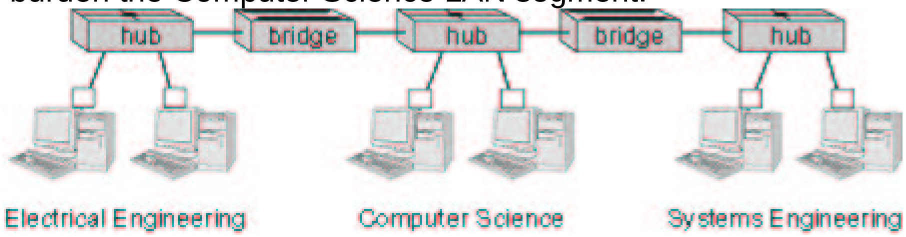


Figure 5.34: An example of an institutional LAN *without* a backbone

One important principle that guides the design of an interconnected LAN is that the various LAN segments should be interconnected with a **backbone**-

-a network that has direct connections to all the LAN segments. When a LAN has a backbone, then each pair of LAN segments can communicate without passing through a third-party LAN segment. The design shown in Figure 5.29 uses a three-interface bridge for a backbone. In the homework problems at the end of this chapter, we shall explore how to design backbone networks with two-interface bridges.

5.6.3: Switches

Up until the mid 1990s, three types of LAN interconnection devices were essentially available: hubs (and their cousins, repeaters), bridges, and routers. More recently yet another interconnection device became widely available, namely, Ethernet switches. Ethernet **switches,** often trumpeted by network equipment manufacturers with great fanfare, are in essence high-performance multi-interface bridges. As do bridges, they forward and filter frames using LAN destination addresses, and they automatically build forwarding tables using the source addresses in the traversing frames. The most important difference between a bridge and switch is that bridges usually have a small number of interfaces (that is, 2-4), whereas switches may have dozens of interfaces. A large number of interfaces generates a high aggregate forwarding rate through the switch fabric, therefore necessitating a high-performance design (especially for 100 Mbps and 1 Gbps interfaces).

Switches can be purchased with various combinations of 10 Mbps, 100 Mbps and 1 Gbps interfaces. For example, one can purchase switches with four 100 Mbps interfaces and twenty 10 Mbps interfaces; or switches with four 100 Mbps interfaces and one 1 Gbps interface. Of course, the more interfaces and the higher transmission rates of the various interfaces, the more one pays. Many switches also operate in a **full-duplex mode;** that is, they can send and receive frames at the same time over the same interface. With a full-duplex switch (and corresponding full-duplex Ethernet adapters in the hosts), host A can send a file to host B while that host B simultaneously sends to host A.

One of the advantages of having a switch with a large number of interfaces is that it facilitates direct connections between hosts and the switch. When a host has a full-duplex direct connection to a switch, it can transmit (and receive) frames at the full transmission rate of its adapter; in particular, the host adapter always senses an idle channel and never experiences a collision. When a host has a direct connection to a switch (rather than a shared LAN connection), the host is said to have **dedicated access.** In Figure 5.35, an Ethernet switch provides dedicated access to six hosts. This dedicated access allows A to send a file to A’ while B is sending a file to B’ and C is sending a file to C’. If each host has a 10 Mbps adapter card, then the aggregate throughput during the three simultaneous file transfers is 30 Mbps. If A and A’ have 100 Mbps adapters and the remaining hosts have 10 Mbps adapters, then the aggregate throughput during the three simultaneous file transfers is 120 Mbps.

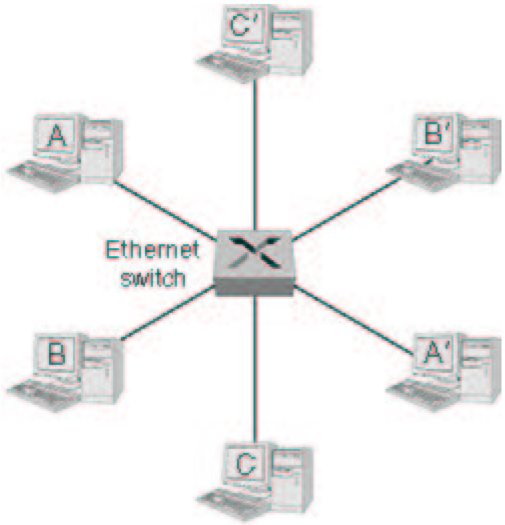


Figure 5.35: An Ethernet switch providing dedicated Ethernet access to six hosts

Figure 5.36 shows how an institution with several departments and several critical servers might deploy a combination of hubs, Ethernet switches, and routers. In Figure 5.36, each of the three departments has its own 10 Mbps Ethernet segment with its own hub. Because each departmental hub has a connection to the switch, all intradepartmental traffic is confined to the Ethernet segment of the department (assuming the forwarding tables in the Ethernet switch are complete). The Web and mail servers each have dedicated 100 Mbps access to the switch. Finally, a router, leading to the Internet, has dedicated 100 Mbps access to the switch. Note that this switch has at least three 10 Mbps interfaces and three 100 Mbps interfaces.

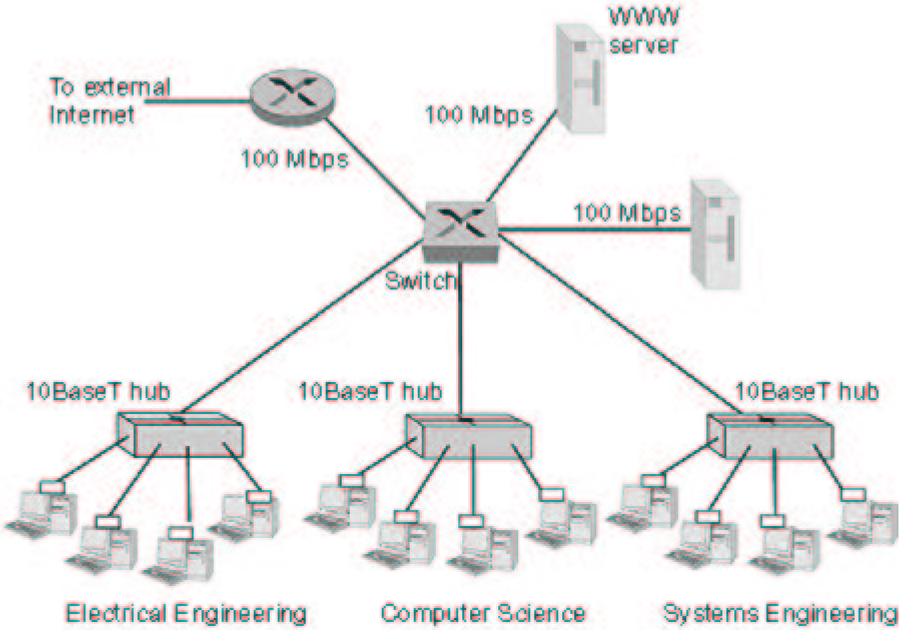


Figure 5.36: An institutional network using a combination of hubs, Ethernet switches, and a router

Cut-through switching

In addition to large numbers of interfaces, support for multitudes of physical media types and transmission rates, and enticing network management features, Ethernet switch manufacturers often tout that their switches use **cut-through switching** rather than store-and-forward packet switching, used by routers and bridges. The difference between store-and-forward and cut-through switching is subtle. To understand this difference consider a packet that is being forwarded through a packet switch (that is, a router, a bridge, or an Ethernet switch). The packet arrives to the switch on an *inbound link* and leaves the switch on an *outbound link.* When the packet arrives, there may or may not be other packets queued in the outbound link's output buffer. When there are packets in the output buffer, there is absolutely no difference between store-and-forward and cut-through switching. The two switching techniques only differ when the output buffer is empty.

Recall from Chapter 1, when a packet is forwarded through a store-and-forward packet switch, the packet is first gathered and stored in its entirety before the switch begins to transmit it on the outbound link. In the case that the output buffer becomes empty before the whole packet has arrived to the switch, this gathering generates a store-and-forward delay at the switch--a delay that contributes to the total end-to-end delay (see Section 1.6). An upper bound on this delay is L/R, where L is the length of the packet and R is transmission rate of the *inbound* link. Note that a packet only incurs a store-and-forward delay if the output buffer becomes empty before the entire packet arrives to the switch.

With cut-through switching, if the buffer becomes empty before the entire packet has arrived, the switch can start to transmit the front of the packet while the back of the packet continues to arrive. Of course, before transmitting the packet on the outbound link, the portion of the packet that contains the destination address must first arrive. (This small delay is inevitable for all types of switching, as the switch must determine the appropriate outbound link.) In summary, with cut-through switching, a packet need not be fully "stored" before it is forwarded; instead the packet is forwarded through the switch when the output link is free. If the output link is a multiple access network that is shared with other hosts (for example, the output link connects to a hub), then the switch must also sense the link as idle before it can "cut-through" a packet.

To shed some insight on the difference between store-and-forward and cut-through switching, let us recall the caravan analogy introduced in Section 1.6. In this analogy, there is a highway with occasional toll booths, with each toll booth having a single attendant. On the highway there is a caravan of 10 cars traveling together, each at the same constant speed. The cars in the caravan are the only cars on the highway. Each toll booth services the cars at a constant rate, so that when the cars leave the toll booth they are equally spaced apart. As before, we can think of the caravan as being a packet, each car in the caravan as being a bit, and the toll booth service rate as the link transmission rate of a link. Consider now what the cars in the caravan do when they arrive to a toll booth. If each car proceeds directly to the toll booth upon arrival, then the toll booth is a "cut-through toll booth." If, on the other hand, each car waits at the entrance until all the remaining cars in the caravan have arrived, then the toll booth is a store-and-forward toll booth. The store-and-forward toll booth clearly delays the caravan more than the cut-through toll booth.

A cut-through switch can reduce a packet's end-to-end delay, but by how much? As we mentioned above, the maximum store-and-forward delay is L/R, where L is the packet size and R is the rate of the inbound link. The maximum delay is approximately 1.2 msec for 10 Mbps Ethernet and 0.12 msec for 100 Mbps Ethernet (corresponding to a maximum size Ethernet packet). Thus, a cut-through switch only reduces the delay by 0.12 to 1.2 msec, and this reduction only occurs when the outbound link is lightly loaded. How significant is this delay? Probably not very much in most practical applications, so you may want to think about selling the family house before investing in the cut-through feature.

We have learned in this section that hubs, bridges, routers, and switches can all be used as an interconnection device for hosts and LAN segments. Table 5.1 provides a summary of the features of each of these interconnection devices. The Cisco Web site provides numerous comparisons of the different interconnection technologies [Cisco LAN Switches 1999].

Table 5.1: Comparison of the typical features of popular interconnection devices

	hubs	bridges	routers	Ethernet switches
traffic isolation	*no*	*yes*	*yes*	*yes*
plug and play	*yes*	*yes*	*no*	*yes*
optimal routing	*no*	*no*	*yes*	*no*
cut-through	*yes*	*no*		

no
yes

Online Book

5.7: IEEE 802.11 LANs

In Section 5.5, we examined the dominant wired LAN protocol--Ethernet. In the previous section we examined how LAN segments can be connected together via hubs, bridges, and switches to form larger LANs. In this section we examine a LAN standard (belonging to the same IEEE 802 family as Ethernet) that is being increasingly deployed for untethered (wireless) LAN communication. The IEEE 802.11 standard [Brenner 1997; Crow 1997; IEEE 802.11 1999] defines the physical layer and media access control (MAC) layer for a wireless local area network. The standard defines three different physical layers for the 802.11 wireless LAN, each operating in a different frequency range and at rates of 1 Mbps and 2 Mbps. In this section we focus on the architecture of 802.11 LANs and their media access protocols. We'll see that although it belongs to the same standard family as Ethernet, it has a significantly different architecture and media access protocol.

5.7.1: 802.11 LAN Architecture

Figure 5.37 illustrates the principal components of the 802.11 wireless LAN architecture. The fundamental building block of the 802.11 architecture is the cell, known as the **basic service set (BSS)** in 802.11 parlance. A BSS typically contains one or more wireless stations and a central **base station,** known as an **access point (AP)** in 802.11 terminology. The wireless stations, which may be either fixed or mobile, and the central base station communicate among themselves using the IEEE 802.11 wireless MAC protocol. Multiple APs may be connected together (for example, using a wired Ethernet or another wireless channel) to form a so-called **distribution system (DS).** The DS appears to upper-level protocols (for example, IP) as a single 802 network, in much the same way that a bridged, wired 802.3 Ethernet network appears as a single 802 network to the upper-layer protocols.

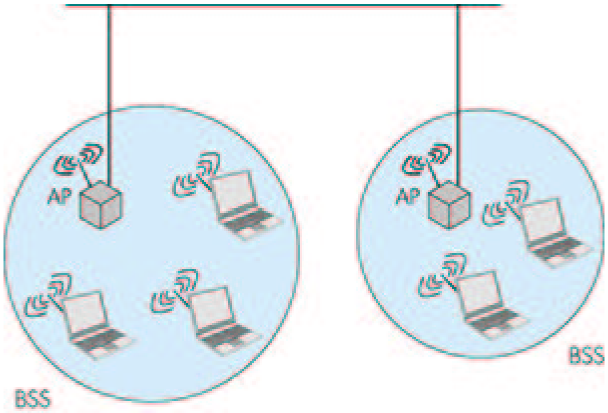


Figure 5.37: IEEE 80.11 LAN architecture

Figure 5.38 shows that IEEE 802.11 stations can also group themselves together to form an **ad hoc network**--a network with no central control and with no connections to the "outside world." Here, the network is formed "on the fly," simply because there happen to be mobile devices that have found themselves in proximity to each other, that have a need to communicate, and that find no pre-existing network infrastructure (for example, a pre-existing 802.11 BSS with an AP) in the location. An ad hoc network might be formed when people with laptops meet together (for example, in a conference room, a train, or a car) and want to exchange data in the absence of a centralized AP. There has been a tremendous recent increase in interest in ad hoc networking, as communicating portable devices continue to proliferate. Within the IETF, activity in ad hoc networking is centered around the mobile ad hoc networks (manet) working group [manet 2000].

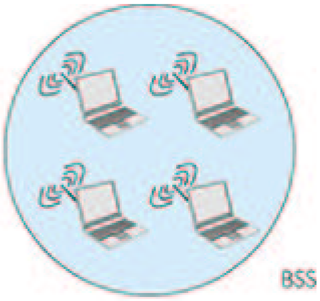


Figure 5.38: An IEEE 8022.11 ad hoc network

5.7.2: 802.11 Media Access Protocols

Just as in a wired 802.3 Ethernet network, stations in an IEEE 802.11 wireless LAN must coordinate their access and use of the shared communication media (in this case the radio frequency). Once again, this is the job of the Media Access Control (MAC) protocol. The IEEE 802.11 MAC protocol is a carrier-sense multiple access protocol with collision avoidance (**CSMA/CA**). Recall from our study of Ethernet in Section 5.5 that a CSMA protocol first senses the channel to determine if the channel is "busy" with the transmission of a frame from some other station. In the 802.11 specification, the physical layer monitors the energy level on the radio frequency to determine whether or not another station is transmitting and provides this carrier sensing information to the MAC protocol. If the channel is sensed idle for an amount of time equal to or greater than the Distributed Inter Frame Space (DIFS), a station is then allowed to transmit. As with any random access protocol, this frame will be successfully received at the destination station if no other station's transmission has interfered with the frame's transmission.

When a receiving station has correctly and completely received a frame for which it was the addressed recipient, it waits a short period of time (known as the Short Inter Frame Spacing--SIFS) and then sends an explicit acknowledgment frame back to the sender. This data-link layer acknowledgment lets the sender know that the receiver has indeed correctly received the sender's data frame. We will see shortly that this explicit acknowledgment is needed because, unlike the case of wired Ethernet, a wireless sender cannot itself determine whether or not its frame transmission was successfully received at the destination. The transmission of a frame by a sending station and its subsequent acknowledgment by the destination station is shown in Figure 5.39.

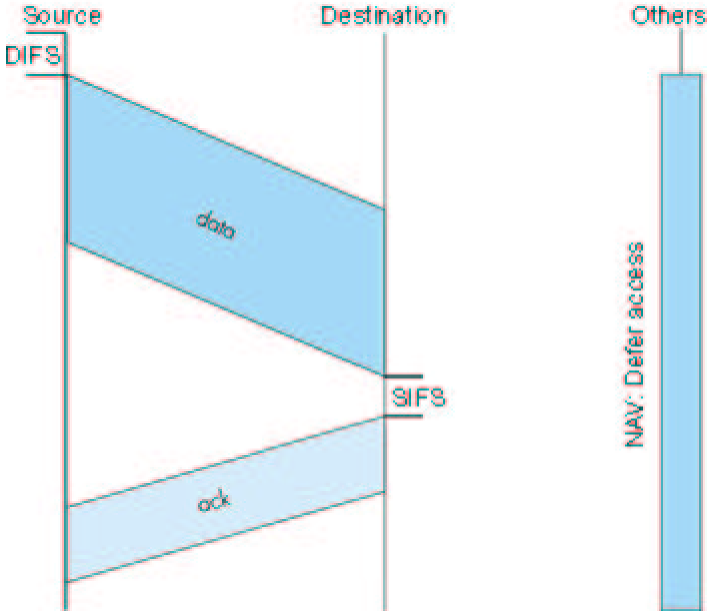


Figure 5.39: Data transmission and acknowledgement in IEEE 802.11

Figure 5.39 illustrates the case when the sender senses the channel to be idle. What happens if the sender senses the channel busy? In this case, the station performs a backoff procedure that is similar to that of Ethernet. More specifically, a station that senses the channel busy will defer its access until the channel is later sensed idle. Once the channel is sensed idle for an amount of time equal to DIFS, the station then computes an *additional* random backoff time and counts down this time as the channel is sensed idle. When the random backoff timer reaches zero, the station transmits its frame. As in the case of Ethernet, the random backoff timer serves to avoid having multiple stations immediately begin transmission (and thus collide) after a DIFS idle period. As in the case of Ethernet, the interval over which the backoff timer randomizes is doubled each time a transmitted frame experiences a collision.

We noted above that unlike the 802.3 Ethernet protocol, the wireless 802.11 MAC protocol does *not* implement collision detection. There are a couple of reasons for this:

- The ability to detect collisions requires the ability to both send (one's own signal) and receive (to determine if another station's transmissions is interfering with one's own transmission) at the same time. This can be costly.
- More importantly, even if one had collision detection and sensed no collision when sending, a collision could still occur at the receiver.

This situation results from the particular characteristics of the wireless channel. Suppose that station A is transmitting to station B. Suppose also that station C is transmitting to station B. With the so-called **hidden terminal problem,** physical obstructions in the environment (for example, a mountain) may prevent A and C from hearing each other's transmissions, even though A's and C's transmissions are indeed interfering at the destination, B. This is shown in Figure 5.40(a). A second scenario that results in undetectable collisions at the receiver results from the **fading** of a signal's strength as propagates through the wireless medium. Figure 5.40(b) illustrates the case where A and C are placed such that their signal strengths are not strong enough for them to detect each other's transmissions, and yet their transmissions are strong enough to have interfered with each other at station B.

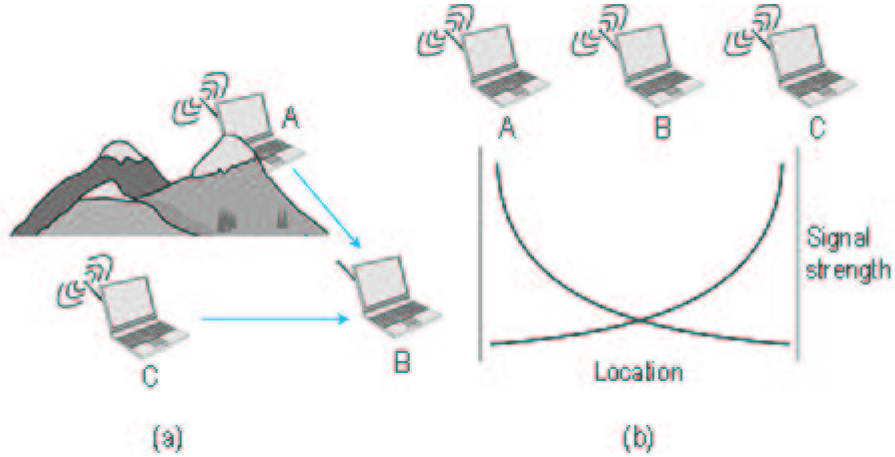


Figure 5.40: Hidden terminal problem (a) and fading (b)

Given these difficulties with detecting collisions at a wireless receiver, the designers of IEEE 802.11 developed an access protocol that aimed to avoid collisions (hence the name CSMA/CA), rather than detect and recover from collisions (CSMA/CD). First, the IEEE 802.11 frame contains a duration field in which the sending station explicitly indicates the length of time that its frame will be transmitting on the channel. This value allows other stations to determine the minimum amount of time (the so-called network allocation vector, NAV) for which they should defer their access, as shown in Figure 5.39.

The IEEE 802.11 protocol can also use a short Request To Send (RTS) control frame and a short Clear To Send (CTS) frame to *reserve* access to the channel. When a sender wants to send a frame, it can first send an RTS frame to the receiver, indicating the duration of the data packet and the ACK packet. A receiver that receives an RTS frame responds with a CTS frame, giving the sender explicit permission to send. All other stations hearing the RTS or CTS then know about the pending data transmission and can avoid interfering with those transmissions. The RTS, CTS, DATA, and ACK frames are shown in Figure 5.41. An IEEE 802.11 sender can operate either using the RTS/CTS control frames, as shown in Figure 5.41, or can simply send its data without first using the RTS control frame, as shown in Figure 5.39.

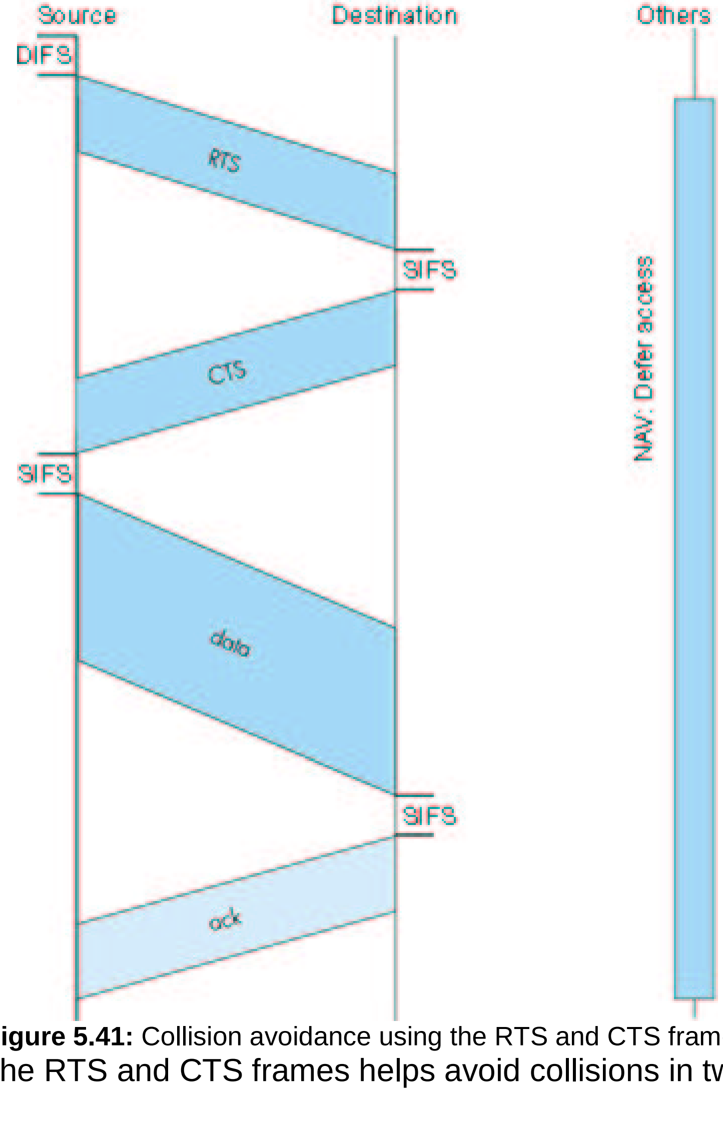


Figure 5.41: Collision avoidance using the RTS and CTS frames

The use of the RTS and CTS frames helps avoid collisions in two important ways:

- Because the receiver's transmitted CTS frame will be heard by all stations within the receiver's vicinity, the CTS frame helps avoid both the hidden station problem and the fading problem.

- Because the RTS and CTS frames are short, a collision involving an RTS or CTS frame will only last for the duration of the whole RTS or CTS frame. Note that when the RTS and CTS frames are correctly transmitted, there should be no collisions involving the subsequent DATA and ACK frames.

In our discussion above, we have only highlighted some of the key aspects of the 802.11 protocol. Additional protocol capabilities such as time synchronization, power management, joining and leaving a network (that is, support for roaming stations) are covered in the full IEEE 802.11 standard. See [Brenner 1997; Crow 1997; IEEE 802.11 1999] for details.

5.8: PPP: The Point-to-Point Protocol

Most of our discussion of data-link protocols thus far has focused on protocols for broadcast channels. In this section we cover a data-link protocol for point-to-point links--PPP, the point-to-point protocol. Because PPP is typically the protocol of choice for a dialup link from residential hosts, it is undoubtedly one of the most widely deployed data-link protocols today. The other important data-link protocol in use today is the HDLC (high-level data link control) protocol; see [Spragins 1991] for a discussion of HDLC. Our discussion here of the simpler PPP protocol will allow us to explore many of the most important features of a point-to-point data-link protocol.

As its name implies, the point-to-point protocol (PPP) [RFC 1661; RFC 2153] is a data-link layer protocol that operates over a **point-to-point link**--a link directly connecting two nodes, one on each end of the link. The point-to-point link over which PPP operates might be a serial dialup telephone line (for example, a 56K modem connection), a SONET/SDH link, an X.25 connection, or an ISDN circuit. As noted above, PPP has become the protocol of choice for connecting home users to their ISPs over a dialup connection.

Before diving into the details of PPP, it is instructive to examine the original requirements that the IETF placed on the design of PPP [RFC 1547]:

- *Packet framing.* The PPP protocol data-link layer sender must be able to take a network-level packet and encapsulate it within the PPP data-link layer frame such that the receiver will be able to identify the start and end of both the data link frame and the network-layer packet within the frame.

- *Transparency.* The PPP protocol must not place any constraints on data appearing on the network-layer packet (headers or data). Thus, for example, the PPP protocol cannot forbid the use of certain bit patterns in the network-layer packet. We'll return to this issue shortly in our discussion of byte stuffing.

- *Multiple network-layer protocols.* The PPP protocol must be able to support multiple network-layer protocols (for example, IP and DECnet) running over the *same* physical link at the *same* time. Just as the IP protocol is required to multiplex different transport level protocols (for example, TCP and UDP) over a single end-to-end connection, so too must PPP be able to multiplex different network layer protocols over a single point-to-point connection. This requirement means that at a minimum, PPP will likely require a "protocol type" field or some similar mechanism so the receiving-side PPP can demultiplex a received frame up to the appropriate network-layer protocol.

- *Multiple types of links.* In addition to being able to carry multiple higher-level protocols, PPP must also be able to operate over a wide variety of link types, including links that are either serial (transmitting a bit at a time in a given direction) or parallel (transmitting bits in parallel), synchronous (transmitting a clock signal along with the data bits) or asynchronous, low-speed or high-speed, electrical or optical.

- *Error detection.* A PPP receiver must be able to detect bit errors in the received frame.

- *Connection liveness.* PPP must be able to detect a failure at the link level (for example, the inability to transfer data from the sending side of the link to the receiving side of the link) and signal this error condition to the network layer.

- *Network-layer address negotiation.* PPP must provide a mechanism for the communicating network layers (for example, IP) to learn or configure each other's network layer address.

- *Simplicity.* PPP was required to meet a number of additional requirements beyond those listed above. On top of all of these requirements, first and foremost among all of the PPP requirements is that of "simplicity." RFC 1547 states "the watchword for a point-to-point protocol should be simplicity." A tall order indeed given all of the other requirements placed on the design of PPP! More than 50 RFCs now define the various aspects of this "simple" protocol.

While it may appear that many requirements were placed on the design of

PPP, the situation could actually have been much more difficult! The design specifications for PPP also explicitly note protocol functionality that PPP was *not* required to implement:

- Error correction. PPP is required to detect bit errors but is *not* required to correct them.

- *Flow control.* A PPP receiver is expected to be able to receive frames at the full rate of the underlying physical layer. If a higher layer cannot receive packets at this full rate, it is then up to the higher layer to drop packets or throttle the sender at the higher layer. That is, rather than having the PPP sender throttle its own transmission rate, it is the responsibility of a higher-level protocol to throttle the rate at which packets are delivered to PPP for sending.

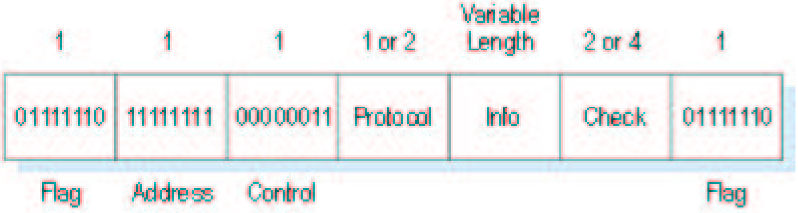
- *Sequencing.* PPP is *not* required to deliver frames to the link receiver in the same order in which they were sent by the link sender. It is interesting to note that while this flexibility is compatible with the IP service model (which allows IP packets to be delivered end-to-end in any order), other network layer protocols that operate over PPP do require sequenced end-to-end packet delivery.

- *Multipoint links.* PPP need only operate over links that have a single sender and a single receiver. Other link-layer protocols (for example, HDLC) can accommodate multiple receivers (for example, an Ethernet-like scenario) on a link.

Having now considered the design goals (and non-goals) for PPP, let us see how the design of PPP met these goals.

5.8.1: PPP Data Framing

Figure 5.42 shows a PPP data frame using HDLC-like framing [RFC 1662].



1	1	1	1 or 2	Variable Length	2 or 4	1
01111110	11111111	00000011	Protocol	Info	Check	01111110
Flag	Address	Control				Flag

Figure 5.42: PPP data frame format

The PPP frame contains the following fields:

- *Flag field.* Every PPP frame begins and ends with a one byte flag field with a value of 01111110.

- *Address field.* The only possible value for this field is 11111111.

- *Control field.* The only possible value of this field is 00000011. Because both the address and control fields can currently take only a fixed value, one wonders why the fields are even defined in the first place. The PPP specification [RFC 1662] states that other values "may be defined at a later time," although none have been defined to date. Because these fields take fixed values, PPP allows the sender to simply not send the address and control bytes, thus saving two bytes of overhead in the PPP frame.

- *Protocol.* The protocol field tells the PPP receiver the upper-layer protocol to which the received encapsulated data (that is, the contents of the PPP frame's info field) belongs. On receipt of a PPP frame, the PPP receiver will check the frame for correctness and then pass the encapsulated data on to the appropriate protocol. RFC 1700 defines the 16-bit protocol codes used by PPP. Of interest to us is the IP protocol (that is, the data encapsulated in the PPP frame is an IP datagram) which has a value of 21 hexadecimal, other network-layer protocols such as Appletalk (29) and DECnet (27), the PPP link control protocol (C021 hexadecimal) that we discuss in detail in the following section, and the IP Control Protocol (8021). This last protocol is called by PPP when a link is first activated in order to configure the IP-level connection between the IP-capable devices on each end of the link (see below).

- *Information.* This field contains the encapsulated packet (data) that is being sent by an upper-layer protocol (for example, IP) over the PPP link. The default maximum length of the information field is 1,500 bytes, although this can be changed when the link is first configured, as discussed below.

- *Checksum.* The checksum field is used to detect bit errors in a transmitted frame. It uses either a two or four byte HDLC-standard cyclic redundancy code.

Byte stuffing

Before closing our discussion of PPP framing, let us consider a problem that arises when any protocol uses a specific bit pattern in a flag field to delineate the beginning or end of the frame. What happens if the flag pattern itself occurs elsewhere in the packet? For example, what happens if the flag field value of 01111110 appears in the information field? Will the receiver incorrectly detect the end of the PPP frame?

One way to solve this problem would be for PPP to forbid the upper-layer protocol from sending data containing the flag field bit pattern. The PPP requirement of transparency discussed above obviates this possibility. An alternate solution, and the one taken in PPP and many other protocols, is to use a technique known as **byte stuffing.**

PPP defines a special control escape byte, 01111101. If the flag sequence, 01111110 appears anywhere in the frame, except in the flag field, PPP precedes that instance of the flag pattern with the control escape byte. That is, it "stuffs" (adds) a control escape byte into the transmitted data stream, before the 01111110, to indicate that the following 011111110 is *not* a flag value but is, in fact, actual data. A receiver that sees a 01111110 preceded by a 01111101 will, of course, remove the stuffed control escape to reconstruct the original data. Similarly, if the control escape byte bit pattern itself appears as actual data, it too must be preceded by a stuffed control escape byte. Thus, when the receiver sees a single control escape byte by itself in the data stream, it knows that the byte was stuffed into the data stream. A pair of control escape bytes occurring back to back means that one instance of the control escape byte appears in the original data being sent. Figure 5.43 illustrates PPP byte stuffing. (Actually, PPP also XORs the data byte being escaped with 20 hexadecimal, a detail we omit here for simplicity.)

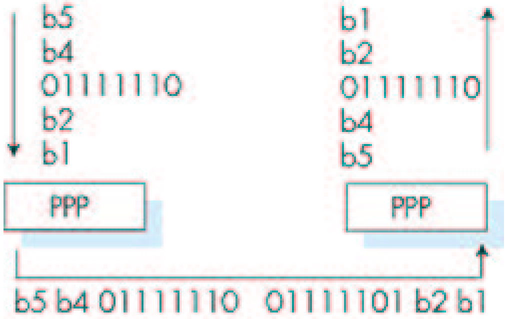


Figure 5.43: Byte stuffing

5.8.2: PPP Link Control Protocol (LCP) and Network Control Protocols

Thus far, we have seen how PPP frames the data being sent over the point-to-point link. But how does the link get initialized when a host or router on one end of the PPP link is first turned on? The initialization, maintenance, error reporting, and shutdown of a PPP link is accomplished using PPP’s link-control protocol (LCP) and family of PPP network-control protocols.

Before any data is exchanged over a PPP link, the two peers (one at each end of the PPP link) must first perform a considerable amount of work to configure the link, in much the same way that a TCP sender and receiver must perform a three-way handshake (see Section 3.5) to set the parameters of the TCP connection before TCP data segments are transmitted. Figure 5.44 illustrates the state transition diagram for the LCP protocol for configuring, maintaining, and terminating the PPP link.

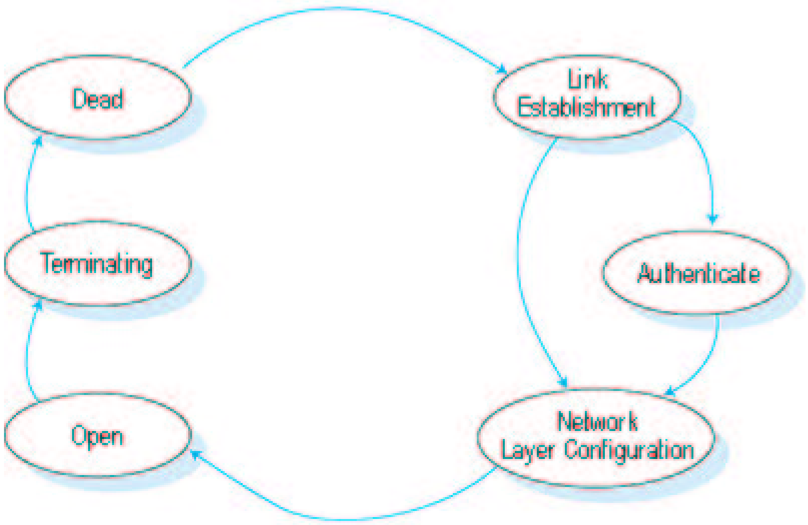


Figure 5.44: PPP link-control protocol

The PPP link always begins and ends in the dead state. When an event such as a carrier detection or network administrator intervention indicates that a physical layer is present and ready to be used, PPP enters the link establishment state. In this state, one end of the link sends its desired link configuration options using an LCP configure-request frame (a PPP frame with the protocol field set to LCP and the PPP information field containing the specific configuration request). The other side then responds with a `configure-ack` frame (all options acceptable), a `configure-nak` frame (all options understood but not acceptable) or a configure-reject frame (options not recognizable or not acceptable for negotiation). LCP configuration options include a maximum frame size for the link, the specification of an authentication protocol (if any) to be used, and an option to skip the use of the address and control fields in PPP frames.

Once the link has been established, link options negotiated, and the authentication (if any) performed, the two sides of the PPP link then exchange network-layer-specific network control packets with each other. If IP is running over the PPP link, the IP control protocol [RFC 1332] is used to configure the IP protocol modules at each end of the PPP link. IPCP data are carried within a PPP frame (with a protocol field value of 8021), just as LCP data are carried in a PPP frame. IPCP allows the two IP modules to exchange or configure their IP addresses and negotiate whether or not IP datagrams will be sent in compressed form. Similar network-control protocols are defined for other network-layer protocols, such as DECnet [RFC 1762] and AppleTalk [RFC 1378]. Once the network layer has been configured, PPP may then begin sending network-layer datagrams--the link is in the opened state and data has begun to flow across the PPP link. The LCP echo-request frame and echo-reply frame can be exchanged between the two PPP endpoints in order to check the status of the link.

The PPP link remains configured for communication until an LCP terminate-request packet is sent. If a terminate-request LCP frame is sent by one end of the PPP link and replied to with a terminate-ack LCP frame, the link then enters the dead state.

In summary, PPP is a data-link layer protocol by which two communicating link-level peers, one on each end of a point-to-point link, exchange PPP frames containing network layer datagrams. The principal components of PPP are:

- *Framing.* A method for encapsulating data in a PPP frame, identifying the beginning and end of the frame, and detecting errors in the frame.
- *Link-control protocol.* A protocol for initializing, maintaining, and taking down the PPP link.
- *Network-control protocols.* A family of protocols, one for each upper layer network protocol, that allows the network-layer modules to configure themselves before network-level datagrams begin flowing across the PPP link.

Online Book

5.9: Asynchronous Transfer Mode (ATM)

The standards for ATM were first developed in the mid 1980s. For those too young to remember, at this time there were predominately two types of networks: telephone networks, that were (and still are) primarily used to carry real-time voice, and data networks, that were primarily used to transfer text files, support remote login, and provide e-mail. There were also dedicated private networks available for video conferencing. The Internet existed at this time, but few people were thinking about using it to transport phone calls, and the World Wide Web was as yet unheard of. It was therefore natural to design a networking technology that would be appropriate for transporting real-time audio and video as well as text, e-mail, and image files. Asynchronous transfer mode (ATM) achieved this goal. Two standards committees, the ATM Forum [ATM Forum 2000] and the International Telecommunications Union [ITU 2000] developed standards for broadband digital services networks.

The ATM standards call for packet switching with virtual circuits (called virtual channels in ATM jargon). The standards define how applications directly interface with ATM, so that ATM provides a complete networking solution for distributed applications. Paralleling the development of the ATM standards, major companies throughout the world made significant investments in ATM research and development. These investments have led to a myriad of high-performing ATM technologies, including ATM switches that can switch terabits per second. In recent years, ATM technology has been deployed very aggressively within both telephone networks and the Internet backbones.

Although ATM has been deployed within networks, it has been less successful in extending itself all the way to desktop PCs and workstations. And it is now questionable whether ATM will ever have a significant presence at the desktop. Indeed, while ATM was brewing in the standards committees and research labs in the late 1980s and early 1990s, the Internet and its TCP/IP protocols were already operational and making significant headway:

- The TCP/IP protocol suite was integrated into all of the most popular operating systems.
- Companies began to transact commerce (e-commerce) over the Internet.
- Residential Internet access became inexpensive.
- Many wonderful desktop applications were developed for TCP/IP networks, including the World Wide Web, Internet phone, and interactive streaming video. Thousands of companies are currently developing new applications and services for the Internet.

Today, we live in a world in which most networking applications interface only with TCP/IP. Nevertheless, ATM switches can forward data at very high rates and, consequently, they have been deployed in Internet backbone networks, where the need to transport traffic at high rates is most acute. When ATM is in the Internet backbone, TCP/IP runs on top of ATM and views an entire ATM network, which might span a continent, as one large link-layer network. In other words, although ATM has not caught hold as a process-to-process solution (or even a desktop-to-desktop solution), it has found a home at the link level within parts of the Internet backbone; this is referred to as IP-over-ATM--a topic we'll cover in Section 5.9.5. For these reasons, we have included our discussion of ATM in this chapter on the link layer, rather than in the previous chapter on the network layer.

5.9.1: Principle Characteristics of ATM

The principle characteristics of ATM are as follows:

- The ATM standard defines a full suite of communication protocols, from an application-level API all the way down through the physical layer.
- The ATM service models include constant bit rate (CBR) service, variable bit rate (VBR) service, available bit rate (ABR) service, and unspecified bit rate (UBR) service. We've already considered each of the service models in detail in Section 4.1.1.
- ATM uses packet switching with fixed-length packets of 53 bytes. In

ATM jargon, these packets are called **cells.** Each cell has 5 bytes of header and 48 bytes of "payload." The fixed-length cells and simple headers have facilitated high-speed switching.

- ATM uses virtual circuits. In ATM jargon, virtual circuits are called **virtual channels.** The ATM header includes a field for the virtual channel number, which is called the **virtual channel identifier (VCI)** in ATM jargon. As discussed in Section 1.3, packet switches use the VCI to route cells toward their destinations.
- ATM provides no retransmissions on a link-by-link basis. If a switch detects an error in an ATM cell header, it attempts to correct the error using error-correcting codes. If it cannot correct the error, it drops the cell rather than request a retransmission from the preceding switch.
- ATM provides congestion control only within the ATM ABR service class (see Table 4.1). We covered ATM ABR congestion control in Section 3.6.3, where we saw that it belongs to the general class of network-assisted congestion control approaches. ATM switches themselves do provide feedback to a sending end system to help it regulate its transmission rate in times of network congestion.
- ATM can run over just about any physical layer. It often runs over fiber optics using the SONET standard at speeds of 155.52 Mbps, 622 Mbps, and higher.

As shown in Figure 5.45, the ATM protocol stack consists of three layers: the ATM physical layer, the ATM layer, and the ATM adaptation layer (AAL):

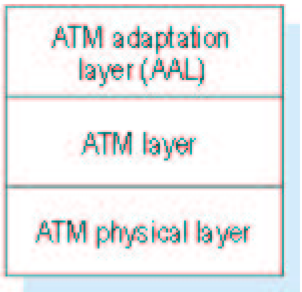


Figure 5.45: The three ATM layers

- The **ATM physical layer** deals with voltages, bit timings, and framing on the physical medium.
- The **ATM layer** is the core of the ATM standard. It defines the structure of the ATM cell.
- The **ATM adaptation layer** (AAL) is roughly analogous to the transport layer in the Internet protocol stack. ATM includes several different types of AALs to support different types of services.

Currently, ATM is most commonly used as a link-layer technology within localized regions of the Internet. A special AAL type, AAL5, has been developed to allow TCP/IP to interface with ATM. At the IP-to-ATM interface, AAL5 prepares IP datagrams for ATM transport; at the ATM-to-IP interface, AAL5 reassembles ATM cells into IP datagrams. Figure 5.46 shows the protocol stack for the regions of the Internet that use ATM. Note that in this configuration, the three ATM layers have been squeezed into the lower two layers of the Internet protocol stack. In particular, the Internet's network layer views ATM as a link-layer protocol. A nice tutorial on ATM, reflecting its original goals, is given in [LeBoudec 1992].

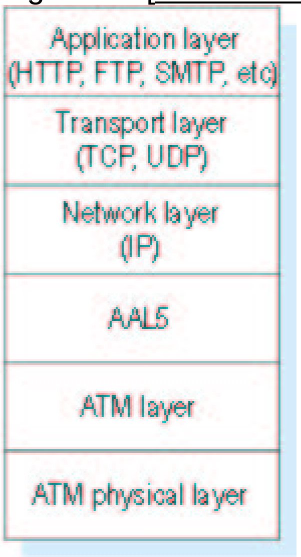


Figure 5.46: Internet-over-ATM protocol stack

5.9.2: ATM Physical Layer

The physical layer is concerned with sending an ATM cell over a single physical link. As shown in Table 5.2, the physical layer has two sublayers: the physical medium dependent (PMD) sublayer and the transmission convergence (TC) sublayer.

Table 5.2: The two sublayers of the physical layer and their responsibilities

Sublayer	**Responsibilities**
Transmission Convergence (TC) Sublayer	Idle cell insertion Cell delineation Transmission frame adaptation
Physical Medium Dependent (PMD) Sublayer	Physical medium Bit voltages and timings Frame structure

The physical medium dependent (PMD) sublayer

The PMD sublayer is at the very bottom of the ATM protocol stack. As the name implies, the PMD sublayer depends on the physical medium of the link; in particular, the sublayer is specified differently for different physical media (fiber, copper, and so on). It is also responsible for generating and delineating bits. There are two classes of PMD sublayers: PMD sublayers that have a transmission frame structure (for example, T1, T3, SONET, or SDH) and PMD sublayers that do not. If the PMD has a frame structure, then it is responsible for generating and delineating frames. (The terminology "frames" in this section is not to be confused with link-layer frames used in the earlier sections of this chapter. The transmission frame is a physical-layer TDM-like mechanism for organizing the bits sent on a link.) The PMD sublayer does not recognize cells. Some possible PMD sublayers include:

1. SONET/SDH (synchronous optical network/synchronous digital hierarchy) over single-mode fiber. Like T1 and T3, SONET and SDH have frame structures that establish bit synchronization between the transmitter and receiver at the two ends of the link. There are several standardized rates, including:

   OC-1: 51.84 Mbps
   OC-3: 155.52 Mbps
   OC-12: 622.08 Mbps

2. T1/T3 frames over fiber, microwave, and copper.

3. Cell-based with no frames. In this case, the clock at receiver is derived from a transmitted signal.

Transmission convergence (TC) sublayer

The ATM layer is specified independently of the physical layer; it has no concept of SONET, T1, or physical media. A sublayer is therefore needed (1) at the sending side of the link to accept ATM cells from the ATM layer and prepare them for transmission on the physical medium, and (2) at the receiving side of the link to group bits arriving from the physical medium into cells and pass the cells to the ATM layer. These are the jobs of the TC sublayer, which sits on top of the PMD sublayer and just below the ATM layer. We note that the TC sublayer is also physical-medium-dependent--if we change the physical medium or the underlying frame structure, then we must also change the TC sublayer.

On the transmit side, the TC sublayer places ATM cells into the bit and transmission frame structure of the PMD sublayer. On the receive side, it extracts ATM cells from the bit and transmission frame structure of the PMD sublayer. It also performs header error correction (HEC). More specifically, the TC sublayer has the following tasks:

- At the transmit side, the TC sublayer generates the HEC byte for each ATM cell that is to be transmitted. At the receive side, the TC sublayer uses the HEC byte to correct all one-bit errors in the header and some multiple-bit errors in the header, reducing the possibility of incorrect routing of cells. The HEC is computed over the first 32-bits in the cell header, using an eight-bit polynomial-coding technique, as described in Section 5.2.3.

- At the receive side, the TC sublayer delineates cells. If the PMD sublayer is cell-based with no frames, then this delineation is typically done by running the HEC on all contiguous sets of 40 bits (that is, 5 bytes). When a match occurs, a cell is delineated. Upon matching four consecutive cells, cell synchronization is declared and subsequent cells are passed to the ATM layer.
- If the PMD sublayer is cell based with no frames, the sublayer sends an idle cell when ATM layer has not provided a cell, thereby generating a continuous stream of cells. The receiving TC sublayer does not pass idle cells to the ATM layer. Idle cells are marked in the PT field in the ATM header.

5.9.3: ATM Layer

When IP runs over ATM, the ATM cell plays the role of the link-layer frame. The ATM layer defines the structure of the ATM cell and the meaning of the fields within this structure. The first five bytes of the cell constitute the ATM header; the remaining 48 bytes constitute the ATM payload. Figure 5.47 shows the structure of the ATM header.

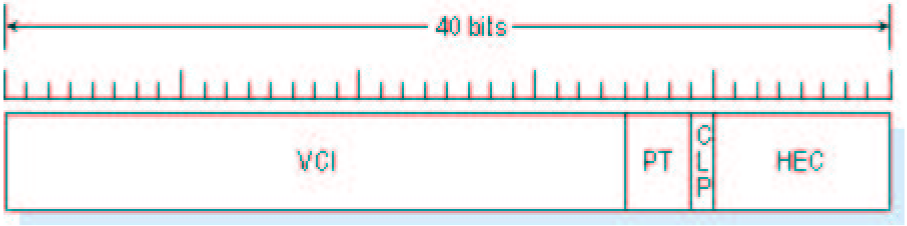


Figure 5.47: The format of the ATM cell header

The fields in the ATM cell are as follows:

- *Virtual channel identifier (VCI).* Indicates the VC to which the cell belongs. As with most network technologies that use virtual circuits, a cell’s VCI is translated from link to link (see Section 1.4).
- *Payload type (PT).* Indicates the type of payload the cell contains. There are several data payload types, several maintenance payload types, and an idle cell payload type. (Recall that idle cells are sometimes needed by the physical layer for synchronization.)
- *Cell-loss priority (CLP) bit.* Can be set by the source to differentiate between high-priority traffic and low-priority traffic. If congestion occurs and an ATM switch must discard cells, the switch can use this bit to first discard low-priority traffic.
- *Header error checksum* (HEC) *byte.* Error detection and correction bits that protect the cell header, as described above.

Virtual channels

Before a source can begin to send cells to a destination, the ATM network must first establish a virtual channel (VC) from source to destination. A virtual channel is nothing more than a virtual circuit, as described in Section 1.4. Each VC is a path consisting of a sequence of links between source and destination. On each of the links the VC has a virtual circuit identifier (VCI). Whenever a VC is established or torn-down, VC translation tables must be updated (see Section 1.4). As noted earlier, ATM backbones in the Internet often use permanent VCs; they obviate the need for dynamic VC establishment and tear-down.

5.9.4: ATM Adaptation Layer

The purpose of the AAL is to allow existing protocols (for example, IP) and applications (for example, constant-bit-rate video) to run on top of ATM. As shown in Figure 5.48, AAL is implemented only at the endpoints of an ATM network. Such an endpoint could be a host system (if ATM provides end-host-to-end-host data transfer) or an IP router (if ATM is being used to connect two IP routers). In this respect, the AAL layer is analogous to the transport layer in the Internet protocol stack.

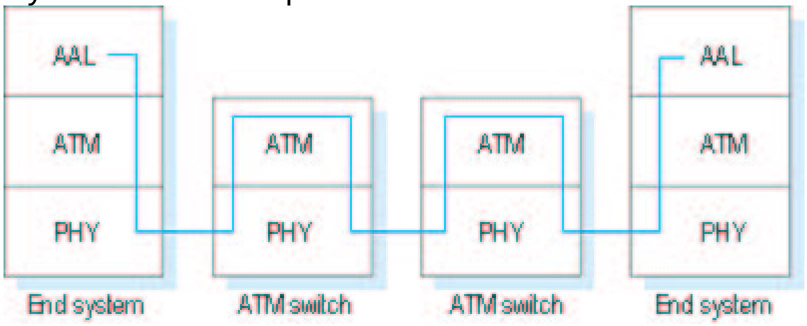


Figure 5.48: The AAL layer is present only at the edges of the ATM network

The AAL sublayer has its own header fields. As shown in Figure 5.49 these fields occupy a small portion of the payload in the ATM cell.

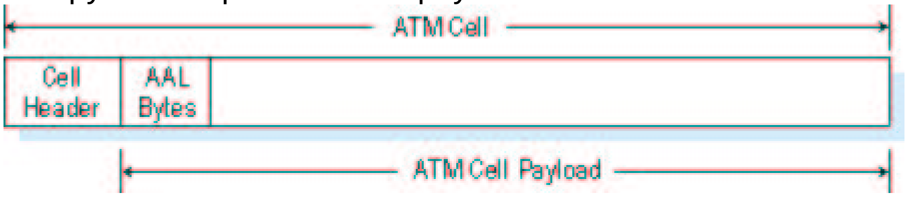


Figure 5.49: The AAL fields within the ATM payload

The ITU and the ATM Forum have standardized several AALs. Some of the most important AALs and the ATM service classes (see Section 4.1.3) they typically support include:

AAL 1: For constant-bit-rate (CBR) services and circuit emulation.
AAL 2: For variable-bit-rate (VBR) services.
AAL 5: For data (for example, IP datagrams)

AAL structure

AAL has two sublayers: the segmentation and reassembly (SAR) sublayer and the convergence sublayer (CS). As shown in Figure 5.50, the SAR sits just above the ATM layer; the CS sublayer sits between the user application and the SAR sublayer. Higher-layer data (for example, an IP datagram) are first encapsulated in a common part convergence sublayer (CPCS) PDU in the Convergence sublayer. This PDU can have a CPCS header and CPCS trailer. Typically, the CPCS-PDU is much too large to fit into the payload of an ATM cell; thus the CPCS-PDU has to be segmented at the ATM source and reassembled at the ATM destination. The SAR sublayer segments the CPCS-PDU and adds AAL header and trailer bits to form the payloads of the ATM cells. Depending on the AAL types, the AAL and CPCS header and trailers could be empty.

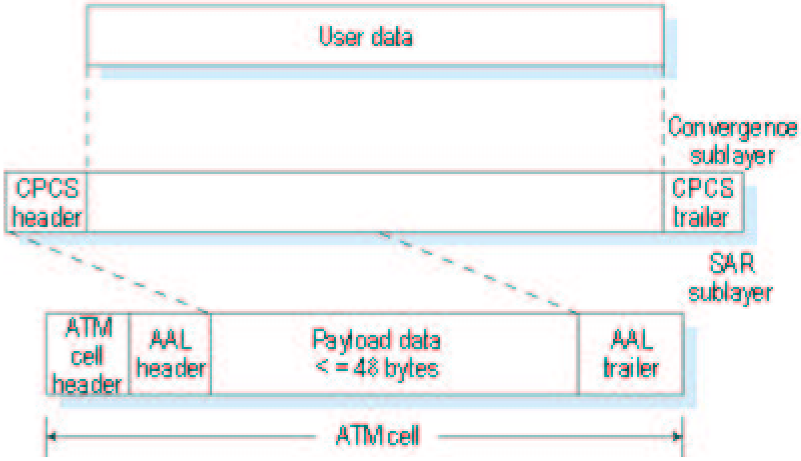
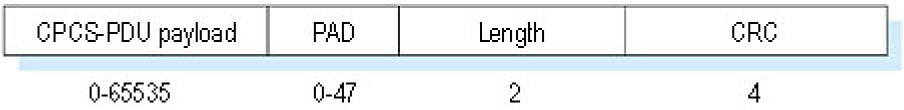


Figure 5.50: The sublayers of the AAL

AAL 5 (Simple and Efficient Adaptation Layer--SEAL)

AAL5 is a low-overhead AAL that is used to transport IP datagrams over ATM networks. With AAL5, the AAL header and trailer are empty; thus, all 48 bytes of the ATM payload are used to carry segments of the CPCS-PDU. An IP datagram occupies the CPCS-PDU payload, which can be from 1 to 65,535 bytes. The AAL5 CPCS-PDU is shown in Figure 5.51.



CPCS-PDU payload	PAD	Length	CRC
0-65535	0-47	2	4

Figure 5.51: CPCS-PDU for AALS

The PAD ensures that the CPCS-PDU is an integer multiple of 48 bytes. The length field identifies the size of the CPCS-PDU payload, so that the PAD can be removed at the receiver. The CRC is the same one that is used by Ethernet, Token Ring, and FDDI. At the ATM source, the AAL5 SAR chops the CPCS-PDU into 48-byte segments. As shown in Figure 5.52, a bit in the PT field of the ATM cell header, which is normally 0, is set to 1 for the last cell of the CPCS-PDU. At the ATM destination, the ATM layer directs cells with a specific VCI to an SAR-sublayer buffer. The ATM cell headers are removed, and the AAL_indicate bit is used to delineate the CPCS-PDUs. Once the CPCS-PDU is delineated, it is passed to the AAL convergence sublayer. At the convergence sublayer, the length field is used to extract the CPCS-PDU payload (for example, an IP datagram), which is passed to the higher layer.

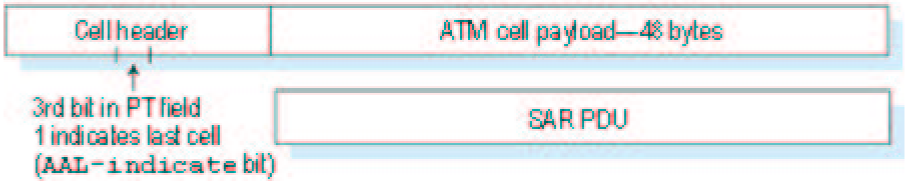


Figure 5.52: The AAL_indicate bit is used to reassemble IP datagrams from ATM cells

5.9.5: IP over ATM

Figure 5.53 shows an ATM backbone with four entry/exit points for Internet IP traffic. Note that each entry/exit point is a router. An ATM backbone can span an entire continent and may have tens or even hundreds of ATM switches. Most ATM backbones have a permanent virtual channel (VC) between each pair of entry/exit points. By using permanent VCs, ATM cells are routed from entry point to exit point without having to dynamically establish and tear down VCs. Permanent VCs, however, are only feasible when the number of entry/exit points is relatively small. For *n* entry points, *n*(*n* - 1) permanent VCs are needed to directly connect *n* entry/exit points.

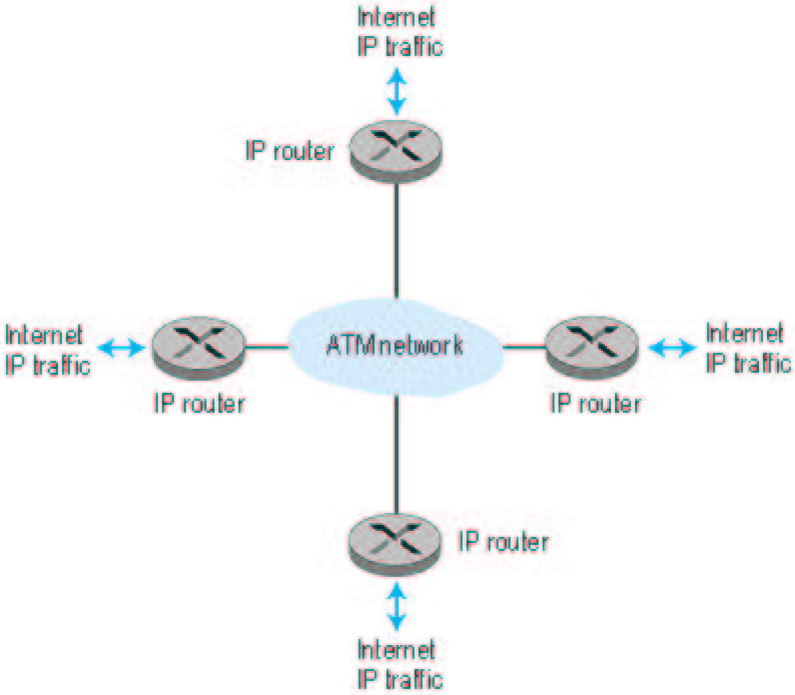


Figure 5.53: ATM network in the core of an Internet backbone

Each router interface that connects to the ATM network will have two addresses. The router interface will have an IP address, as usual, and the router will have an ATM address, which is essentially a LAN address (see Section 5.4).

Consider now an IP datagram that is to be moved across the ATM backbone in Figure 5.53. Note that to the four IP routers, the backbone appears as a single logical link--ATM interconnects these four routers just as Ethernet can be used to connect four routers. Let us refer to the router at which the datagram enters the ATM network as the "entry router" and the router at which the datagram leaves the network as the "exit router." The entry router does the following:

1. Examines the destination address of the datagram.
2. Indexes its routing table and determines the IP address of the exit router (that is, the next router in the datagram's route).
3. To get the datagram to the exit router, the entry router views ATM as just another link-layer protocol. To move the datagram to the next router, the physical address of the next-hop router must be determined. Recall from our discussion in Section 5.4, that this is done using ARP. In particular, the entry router indexes an ATM ARP table with the IP address of the exit router and determines the ATM address of the exit router.
4. IP in the entry router then passes down to the link layer (that is, ATM) the datagram along with the ATM address of the exit router.

After these four steps have been completed, the job of moving the datagram to the exit router is out of the hands of IP and in the hands of ATM. ATM must now move the datagram to the ATM destination address obtained in Step 3 above. This task has two subtasks:

- Determine the VCI for the VC that leads to the ATM destination address.
- Segment the datagram into cells at the sending side of the VC (that is, at the entry router), and reassemble the cells into the original datagram at the receiving side of the VC (that is, at the exit router).

The first subtask listed above is straightforward. The interface at the sending side maintains a table that maps ATM addresses to VCIs. Because we are assuming that the VCs are permanent, this table is up-to-date and static. (If the VCs were not permanent, then an ATM signaling protocol would be needed to dynamically establish and tear down the VCs.) The second task merits careful consideration. One approach is to use IP fragmentation, as discussed in Section 4.4.4. With IP fragmentation, the sending router would first fragment the original datagram into fragments, with each fragment being no more than 48 bytes, so that the fragment could fit into the payload of the ATM cell. But this fragmentation approach has a big problem--each IP fragment typically has 20 bytes of header, so that an ATM cell carrying a fragment would have 25 bytes of "overhead" and only 28 bytes of useful information. ATM uses AAL5 to provide a more efficient way to segment and reassemble a datagram.

Recall that IP in the entry router passes the datagram down to ATM along with the ATM address of the exit router. ATM in the entry router indexes an ATM table to determine the VCI for the VC that leads to the ATM destination address. AAL5 then creates ATM cells out of the IP datagram:

- The datagram is encapsulated in a CPCS-PDU using the format in

Figure 5.52.

- The CPCS-PDU is chopped up into 48-byte chunks. Each chunk is placed in the payload field of an ATM cell.
- All of the cells except for the last cell have the third bit of the PT field set to 0. The last cell has the bit set to 1.

AAL5 then passes the cells to the ATM layer. ATM sets the VCI and CLP fields and passes each cell to the TC sublayer. For each cell, the TC sublayer calculates the HEC and inserts it in the HEC field. The TC sublayer then inserts the bits of the cells into the PMD sublayer.
The ATM network then moves each cell across the network to the ATM destination address. At each ATM switch between ATM source and ATM destination, the ATM cell is processed by the ATM physical and ATM layers, but not by the AAL layer. At each switch the VCI is typically translated (see Section 1.4) and the HEC is recalculated. When the cells arrive at the ATM destination address, they are directed to an AAL buffer that has been put aside for the particular VC. The CPCS-PDU is reconstructed using the AAL_indicate bit to determine which cell is the last cell of the CPCS-PDU. Finally, the IP datagram is extracted out of the CPCS-PDU and is passed up the protocol stack to the IP layer.

5.9.6: ARP and ATM

Consider once again the problem of moving a datagram from entry IP router to exit router across the ATM network in Figure 5.53. Recall that ARP has the important role of translating the exit router's address to an ATM destination address. This translation is straightforward if the ARP table is complete and accurate. But as with Ethernet, ATM ARP tables are auto-configured and may not be complete. As with Ethernet, if the desired mapping is not in the table, an ARP protocol must contact the exit router and obtain the mapping. However, there is a fundamental difference here between Ethernet and ATM--Ethernet is a broadcast technology and ATM is a switched technology. What this means is that ATM cannot simply send an ARP request message within a broadcast packet. ATM must work harder to get the mapping. There are two generic approaches that can be used: (1) broadcast ARP request messages and (2) ARP server.

- *Broadcast ARP request messages:* In this approach, the entry router constructs an ARP request message, converts the message to cells, and sends the cells into the ATM network. These cells are sent by the source along a special VC reserved for ARP request messages. The switches broadcast all cells received on this special VC. The exit router receives the ARP request message and sends the entry router an ARP response message (which is not broadcasted). The entry router then updates its ARP table. This approach can place a significant amount of overhead ARP broadcast traffic into the network.

- *ARP server:* In this approach, an ARP server is attached directly to one of the ATM switches in the network, with permanent VCs set up between each router and the ARP server. All of these permanent VCs use the same VCI on all links from the routers to the ARP server. There are also permanent VCs from the ARP server to each router. The ARP server contains an up-to-date ARP table that maps IP addresses to ATM addresses. Using some registration protocol, all routers must register themselves with the ARP server. This approach eliminates the broadcast ARP traffic. However it requires an ARP server, which can be swamped with ARP request messages.

An important reference for running ARP over ATM is RFC 1577, which discusses IP and ARP over ATM. RFC 1932 also provides a good overview of IP over ATM.

Online Book

5.10: X.25 and Frame Relay

In this section we discuss two end-to-end wide-area-networking (WAN) technologies, namely X.25 and Frame Relay. Introduced in the early 1980s and popular in Europe up through the mid 1990s, X.25 is arguably the first public packet-switching technology. Frame Relay, a successor to X.25, is another public packet-switching technology that has been popular in North America throughout the 1990s.

Given that X.25 and Frame Relay are end-to-end WAN technologies, you may be wondering why we are discussing them in a chapter that is devoted to the data-link layer? We have chosen to discuss these technologies in this chapter for the same reason we chose to discuss ATM in this chapter--all of these technologies are often employed today to carry IP datagrams from one IP router to another. Thus, from the perspective of IP (which is also an end-to-end WAN technology), X.25, Frame Relay, and ATM are link-layer technologies. Because IP is one of the protocols being highlighted in this book, we have put X.25, Frame Relay, and ATM where IP (and most Internet zealots) believe these technologies belong, namely, in the link layer.

Although X.25 still exists throughout Europe and in certain niche markets in North America, the X.25 networks are on the verge of extinction throughout the world. They were designed almost twenty years ago for a technological context that is very different from today’s. Frame Relay had great appeal to corporate customers throughout the 1990s, but it is increasingly fighting fierce competition from the public Internet. In fact, due to this competition, Frame Relay may become a minor player in the mid-2000s. Even though X.25 is on its way out and Frame Relay may disappear as well a few years down the road, we have chosen to cover these technologies in this book because of their immense historical importance.

5.10.1: A Few Words About X.25

The X.25 protocol suite was designed in the late 1970s. To understand the motivation behind the design, we need to understand the technological context of that ancient era. Although the Apple II personal computer was making a big hit at this time [Nerds 1996], PCs and workstations were not widespread and didn't have much networking support. Instead, most people were using inexpensive "dumb terminals" to access distant mainframes over computer networks. These dumb terminals had minimal intelligence and storage (no disks); what appeared on their screens was completely controlled by the mainframe at the other end of the network. In order to widely support dumb terminals, the designers of X.25 decided to "put the intelligence in the network." This philosophy, as we now know, is diametrically opposed to the Internet philosophy, which places much of the complexity in the end systems and makes minimal assumptions about network-layer services.

One way the designers put intelligence in the X.25 network was by employing virtual circuits. Recall from Chapter 1 that virtual-circuit networks require the packet switches to maintain state information. In particular, the switch must maintain a table that maps inbound interface/VC-number to outbound interface/VC-number. Moreover, complex signaling protocols are needed to establish VCs and tear them down. As we learned in Chapter 4, the IP protocol is connectionless and, thus, does not use VCs. When a node wants to send an IP datagram into the network, it just stamps the datagram with a destination address and injects it into the network; it does not first request the network to establish a virtual circuit between itself and the destination.

Another important part of the technological context of the late 1970s and early 1980s concerns the physical links. In those days, almost all of the wired links were noisy, error-prone copper links. Fiber-optic links were still in the research laboratories at that time. Bit error rates over long-haul copper links were *many* orders of magnitude higher than they are now over fiber links. Because of the high error rates, it made sense to design the X.25 protocol with error recovery on a hop-by-hop basis. In particular, whenever an X.25 switch sends a packet, it keeps a copy of the packet until the next switch (in the packet's route) returns an acknowledgment. Thus each switch, when receiving a packet, performs error checking, and if the packet is error-free, it sends an acknowledgment to the previous switch. Hop-by-hop error recovery significantly reduces link transmission rates, and was consistent with the technological context of the era--high link error rates and dumb terminals. The X.25 design also calls for flow-control on a hop-by-hop basis. By contrast, the TCP performs error recovery and flow control on an end-to-end basis, and thus does not require the links to perform these tasks.

5.10.2: Frame Relay

Frame Relay, designed in the late 1980s and widely deployed in the 1990s, is in many ways a second-generation X.25. Like X.25, it uses virtual circuits. However, because the fiber-based systems of the 1990s had much lower bit error rates than the copper-based systems of the 1980s, Frame Relay was naturally designed for much lower error rates. The essence of Frame Relay is a VC-based packet-switching service with no error recovery and no flow control. Whenever a Frame Relay switch detects an error in a packet, its only possible course of action is to discard the data. This results in a network with lower processing overheads and higher transmission rates than X.25, but requires intelligent end systems for data integrity. In most cases today, the Frame Relay network is owned by a public network service provider (for example, AT&T, Sprint, or Bell Atlantic) and its use is contracted on a multiyear basis to corporate customers. Frame Relay is extensively used today to allow LANs on different corporate campuses to send data to each other at reasonably high speeds. As shown in Figure 5.54, Frame Relay often interconnects these LANs through IP routers, with each IP router in a different corporate campus. Frame Relay offers a corporation an alternative to sending its intercampus IP traffic over the public Internet, for which the corporation may have reliability and security concerns.

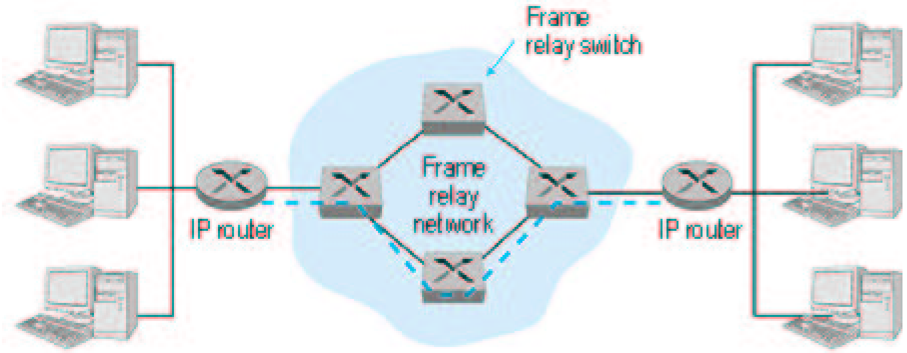


Figure 5.54: Public Frame Relay network interconnecting two Ethernets through routers located on the Ethernets. The dotted line represents a virtual circuit.

Frame Relay networks can use either switched VCs (SVCs) or permanent virtual circuits (PVCs). For router interconnection, a PVC is often permanently established between each pair of routers. $N(N - 1)/2$ PVCs are necessary to interconnect N routers. Throughout our discussion we shall assume that the frame relay network uses PVCs (which is the more common case).

Sending an IP datagram from Ethernet to Frame Relay to Ethernet

Consider the transmission of an IP datagram between two end systems on two Ethernets interconnected by a Frame Relay network. Let's walk through the steps in the context of Figure 5.54. When an Ethernet frame arrives to the source router, the router's Ethernet card strips off the Ethernet fields and passes the IP datagram to the network layer. The network layer passes the IP datagram to the Frame Relay interface card. This card encapsulates the IP datagram in the Frame Relay frame, as shown in Figure 5.55. It also calculates the CRC (2 bytes) and inserts the resulting value in the CRC field. The link-layer field (2 bytes) includes a 10-bit virtual-circuit number field. The interface card obtains the VC number from a table that associates IP network numbers to VC numbers. The interface card then transmits the packet.

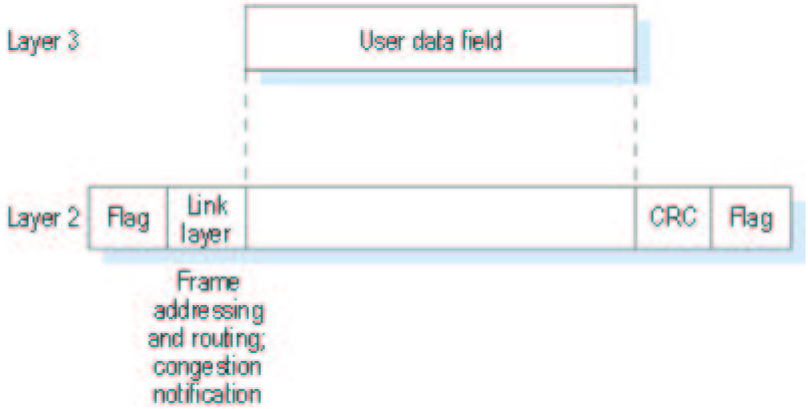


Figure 5.55: Encapsulating user data (for example, an IP datagram) into a Frame Relay frame

The interface card transmits the Frame Relay packet to a nearby Frame Relay switch, owned by the Frame Relay service provider. The switch examines the CRC field. If the frame has an error, the switch discards the frame; unlike X.25, frame relay does not bother to retransmit packets on a hop-by-hop basis. If there is no error in the frame, the switch uses the frame's VC number to route the frame to the next switch (or to the destination router). The destination router removes the frame relay fields and then delivers the datagram over Ethernet to the destination host. If TCP segments are lost or arrive out of sequence, then TCP in the communicating hosts corrects the problem.

Committed Information Rate (CIR)

Frame Relay makes use of an innovative mechanism referred to as the **committed information rate (CIR).** Every frame relay VC has a committed information rate. We will define the CIR rigorously below, but roughly, the CIR is a *commitment* on the part of the Frame Relay network to dedicate to the VC a specified transmission rate determined by the CIR. The CIR service, introduced by Frame Relay in the early 1990s, is in many ways a forerunner to the Internet's differentiated service (see Chapter 6). As we shall shortly see, Frame Relay provides the CIR service by marking packets.

In Frame Relay networks, Frame Relay packets can belong to one of two priority levels--either high priority or low priority. Packets are assigned priorities by *marking* a special bit in the packet header--the so-called discard eligibility (DE) bit--to either 0 for high priority and 1 for low priority. If a frame is a high-priority frame, then the Frame Relay network should deliver the packet to the destination under all but the most desperate network conditions, including periods of congestion and backbone link failures. However, for low-priority packets, the Frame Relay network is permitted to discard the frame under congested conditions. Under particularly draconian conditions, the network can even discard high-priority packets. Congestion is typically measured by the state of output buffers in Frame Relay switches. When an output buffer in a Frame Relay switch is about to overflow, the switch will first discard the low-priority packets, that is, the packets in the buffer with the DE bit set to 1.

The actions that a Frame Relay switch takes on marked packets should be clear, but we haven’t said anything about how packets get marked. This is where the CIR comes in. To explain this, we need to introduce a little frame-relay jargon, which we do in the context of Figure 5.54. The **access rate** is the rate of the access link, that is, the rate of the link from the source router to the "edge" Frame Relay switch. This rate is often 64 Kbps, but integer multiples of 64 Kbps up to 1.544 Mbps are also common. Denote R for the access rate. As we learned in Chapter 1, each packet sent over the link of rate R is transmitted at rate R bps. The edge switch is responsible for marking packets that arrive from the source router. To perform the marking, the edge switch examines the arrival times of packets from the source router over short, fixed intervals of time, called the **measurement interval,** denoted by T_c. Most frame-relay service providers use a T_c value that falls somewhere between 100 msec and 1 sec.

Now we can precisely describe the CIR. Each VC that emanates from the source router (there may be many, possibly destined to different LANs) is assigned a **committed information rate (CIR),** which is in units of bits/sec. The CIR is never greater than R, the access rate. Customers pay for a specific CIR; the higher the CIR, the more the customer pays to the Frame Relay service provider. If the VC generates packets at a rate that is less than the CIR, then all of the VC’s packets will be marked as high-priority packets (DE = 0). However, if the rate at which the VC generates packets exceeds the CIR, then the fraction of the VC’s packets that exceed the rate will be marked as low-priority packets. More specifically, over each measurement interval T_c, for the first $CIR \bullet T_c$ bits the VC sends, the edge switch marks the corresponding packets as high-priority packets (DE = 0). The edge switch marks all additional packets sent over this interval as low-priority packets (DE = 1).

To get a feel for what is going on here, let us look at an example. Let us suppose that the Frame Relay service provider uses a measurement interval of T_c = 500 msec. Suppose that the access link is R = 64 Kbps and that the CIR assigned to a particular VC is 32 Kbps. Also suppose, for simplicity, that each Frame Relay packet consists of exactly L = 4,000 bits. This means that every 500 msec the VC can send CIR • T_c/L = 4 packets as high-priority packets. All additional packets sent within the 500 msec interval are marked as low-priority packets. Note that up to four low-priority packets can be sent over each 500 msec interval (in addition to four high-priority packets). Because the goal of the frame relay network is to deliver all high-priority packets to the destination frame-relay node, the VC is essentially guaranteed of a throughput of at least 32 Kbps. Frame Relay does not, however, make any guarantees about the end-to-end delays of either the high- or low-priority packets.

Increasing the measurement interval T_c increases the potential burstiness of the high-priority packets emitted from the source router. In the previous example, if T_c = 0.5 sec, up to four high-priority packets can be emitted back-to-back; for T_c = 1 sec, up to eight high-priority packets can be emitted back-to-back. When the frame relay network uses a smaller value of T_c, it forces the stream of high-priority packets to be smoother (less bursty); but a large value of T_c gives the VC more flexibility. In any case, for every choice of T_c, the long-run average rate of bits emitted as high-priority bits never exceeds the CIR of the VC.

We must keep in mind that many PVCs may emanate from the source router and travel over the access link. It is interesting to note that the sum of the CIRs for all these VCs is permitted to exceed the access rate, *R*. This is referred to as **overbooking.** Because overbooking is permitted, an access link may transmit high-priority packets at a corresponding bit rate that exceeds the CIR (even though each individual VC sends priority packets at a rate that does not exceed the CIR).

We conclude this section by mentioning that the Frame Relay Forum [FRForum 2000] maintains a number or relevant specifications. An excellent introductory course for Frame Relay is made available on the Hill Associates Web site [Hill 2000]. Walter Goralski has also written a readable yet in-depth book about Frame Relay [Goralski 1999].

Online Book

5.11: Summary

In this chapter, we've examined the data-link layer--its services, the principles underlying its operation, and a number of important specific protocols that use these principles in implementing data-link services.

We saw that the basic service of the data-link layer is to move a network-layer datagram from one node (router or host) to an adjacent node. We saw that all data link protocols operate by encapsulating a network-layer datagram within a link-layer frame before transmitting the frame over the "link" to the adjacent node. Beyond this common framing function, however, we learned that different data link protocols provide very different link access, delivery (reliability, error detection/correction), flow control, and transmission (for example, full-duplex versus half-duplex) services. These differences are due in part to the wide variety of link types over which data-link protocols must operate. A simple point-to-point link has a single sender and receiver communicating over a single "wire." A multiple access link is shared among many senders and receivers; consequently, the data link protocol for a multiple access channel has a protocol (its multiple access protocol) for coordinating link access. In the cases of ATM, X.25, and frame relay, we saw that the "link" connecting two adjacent nodes (for example, two IP routers that are adjacent in an IP sense--that they are next-hop IP routers toward some destination), may actually be a *network* in and of itself. In one sense, the idea of a network being considered as a "link" should not seem odd. A telephone "link" connecting a home modem/computer to a remote modem/router, for example, is actually a path through a sophisticated and complex telephone *network.*

Among the principles underlying data-link communication, we examined error-detection and correction techniques, multiple access protocols, link-layer addressing, and the construction of extended local area networks via hubs, bridges, and switches. In the case of error detection/correction, we examined how it is possible to add additional bits to a frame’s header in order to detect, and in some cases correct, bit-flip errors that might occur when the frame is transmitted over the link. We covered simple parity and checksumming schemes, as well as the more robust cyclic redundancy check. We then moved on to the topic of multiple access protocols. We identified and studied three broad approaches for coordinating access to a broadcast channel: channel partitioning approaches (TDM, FDM, CDMA), random access approaches (the ALOHA protocols, and CSMA protocols), and taking-turns approaches (polling and token passing). We saw that a consequence of having multiple nodes share a single broadcast channel was the need to provide node addresses at the data-link level. We learned that physical addresses were quite different from network-layer addresses, and that in the case of the Internet, a special protocol (ARP--the address-resolution protocol) is used to translate between these two forms of addressing. We then examined how nodes sharing a broadcast channel form a local area network (LAN), and how multiple LANs can be connected together to form larger LANs--all *without* the intervention of network-layer routing to interconnect these local nodes. Finally, we covered a number of specific data-link layer protocols in detail--Ethernet, the wireless IEEE 802.11 protocol, and the point-to-point protocol, PPP. As discussed in Sections 5.9 and 5.10, ATM, X.25, and Frame Relay can also be used to connect two network-layer routers. For example, in the IP-over-ATM scenario, two adjacent IP routers can be connected to each other by a virtual circuit through an ATM network. In such circumstances, a network that is based on one network architecture (for example, ATM, or Frame Relay) can serve as a single logical link between two neighboring nodes (for example, IP routers) in another network architecture.

Having covered the data link layer, *our journey down the protocol stack is now over*! Certainly, the physical layer lies below the data-link layer, but the details of the physical layer are probably best left for another course (for example, in communication theory, rather than computer networking). We have, however, touched upon several aspects of the physical layer in this chapter (for example, our brief discussions of Manchester encoding in Section 5.5 and of signal fading in Section 5.7) and in Chapter 1 (our discussion of physical media in Section 1.5).

Although our journey down the protocol stack is over, our study of computer networking is not yet at an end. In the following three chapters we cover multimedia networking, network security, and network management. These three topics do not fit conveniently into any one layer; indeed, each topic crosscuts many layers. Understanding these topics (sometimes billed as "advanced topics" in some networking texts) thus requires a firm foundation in all layers of the protocol stack--a foundation that our study of the data link layer has now completed!