CHAPTER 9

Introduction to Data-Link Layer

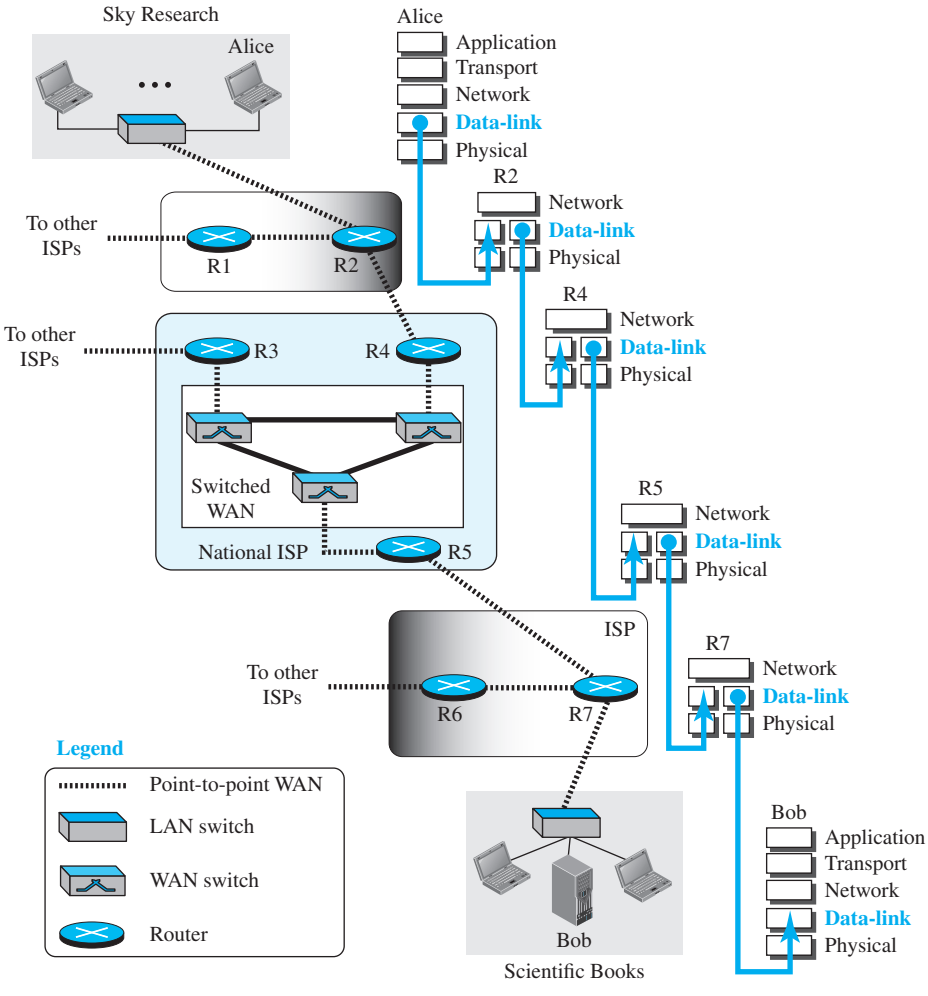
The TCP/IP protocol suite does not define any protocol in the data-link layer or physical layer. These two layers are territories of networks that when connected make up the Internet. These networks, wired or wireless, provide services to the upper three layers of the TCP/IP suite. This may give us a clue that there are several standard protocols in the market today. For this reason, we discuss the data-link layer in several chapters. This chapter is an introduction that gives the general idea and common issues in the data-link layer that relate to all networks.

- The first section introduces the data-link layer. It starts with defining the concept of links and nodes. The section then lists and briefly describes the services provided by the data-link layer. It next defines two categories of links: point-to-point and broadcast links. The section finally defines two sublayers at the data-link layer that will be elaborated on in the next few chapters.
- The second section discusses link-layer addressing. It first explains the rationale behind the existence of an addressing mechanism at the data-link layer. It then describes three types of link-layer addresses to be found in some link-layer protocols. The section discusses the Address Resolution Protocol (ARP), which maps the addresses at the network layer to addresses at the data-link layer. This protocol helps a packet at the network layer find the link-layer address of the next node for delivery of the frame that encapsulates the packet. To show how the network layer helps us to find the data-link-layer addresses, a long example is included in this section that shows what happens at each node when a packet is travelling through the Internet.

9.1 INTRODUCTION

The Internet is a combination of networks glued together by connecting devices (routers or switches). If a packet is to travel from a host to another host, it needs to pass through these networks. Figure 9.1 shows the same scenario we discussed in Chapter 3, but we are now interested in communication at the data-link layer. Communication at the data-link layer is made up of five separate logical connections between the data-link layers in the path.

Figure 9.1 *Communication at the data-link layer*



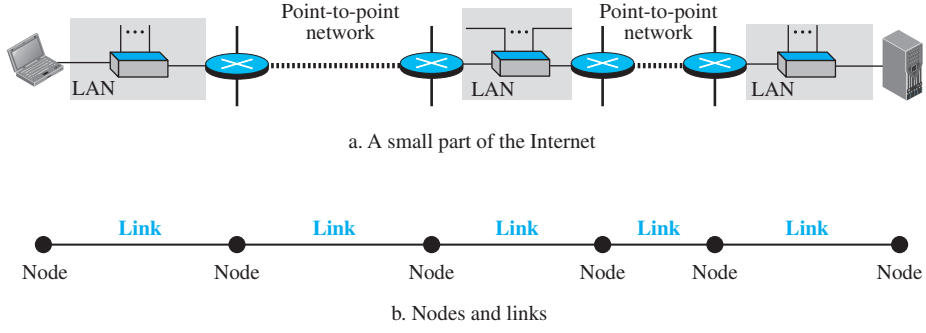
The data-link layer at Alice's computer communicates with the data-link layer at router
he data-link layer at router R2 communicates with the data-link layer at router R4,

and so on. Finally, the data-link layer at router R7 communicates with the data-link layer at Bob's computer. Only one data-link layer is involved at the source or the destination, but two data-link layers are involved at each router. The reason is that Alice's and Bob's computers are each connected to a single network, but each router takes input from one network and sends output to another network. Note that although switches are also involved in the data-link-layer communication, for simplicity we have not shown them in the figure.

9.1.1 Nodes and Links

Communication at the data-link layer is node-to-node. A data unit from one point in the Internet needs to pass through many networks (LANs and WANs) to reach another point. Theses LANs and WANs are connected by routers. It is customary to refer to the two end hosts and the routers as ***nodes*** and the networks in between as ***links*.** Figure 9.2 is a simple representation of links and nodes when the path of the data unit is only six nodes.

Figure 9.2 *Nodes and Links*



The first node is the source host; the last node is the destination host. The other four nodes are four routers. The first, the third, and the fifth links represent the three LANs; the second and the fourth links represent the two WANs.

9.1.2 Services

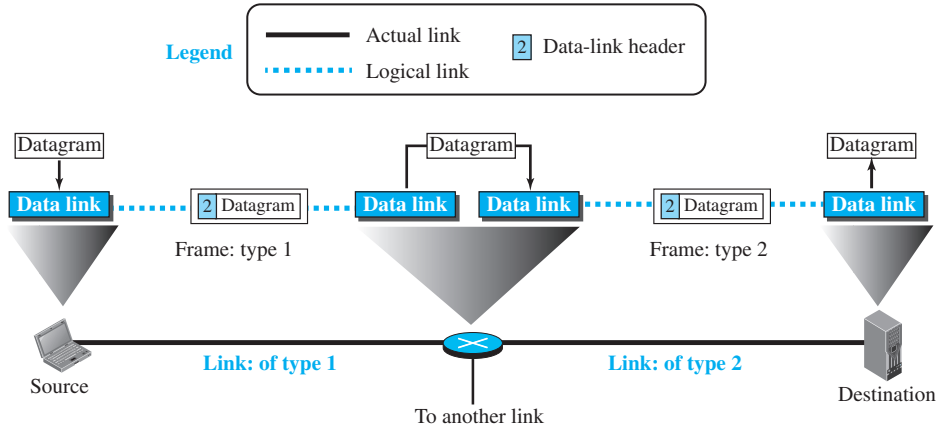
The data-link layer is located between the physical and the network layers. The data-link layer provides services to the network layer; it receives services from the physical layer. Let us discuss services provided by the data-link layer.

The duty scope of the data-link layer is node-to-node. When a packet is travelling in the Internet, the data-link layer of a node (host or router) is responsible for delivering a datagram to the next node in the path. For this purpose, the data-link layer of the sending node needs to encapsulate the datagram received from the network in a frame, and the data-link layer of the receiving node needs to decapsulate the datagram from the frame. In other words, the data-link layer of the source host needs only to

encapsulate, the data-link layer of the destination host needs to decapsulate, but each intermediate node needs to both encapsulate and decapsulate. One may ask why we need encapsulation and decapsulation at each intermediate node. The reason is that each link may be using a different protocol with a different frame format. Even if one link and the next are using the same protocol, encapsulation and decapsulation are needed because the link-layer addresses are normally different. An analogy may help in this case. Assume a person needs to travel from her home to her friend's home in another city. The traveller can use three transportation tools. She can take a taxi to go to the train station in her own city, then travel on the train from her own city to the city where her friend lives, and finally reach her friend's home using another taxi. Here we have a source node, a destination node, and two intermediate nodes. The traveller needs to get into the taxi at the source node, get out of the taxi and get into the train at the first intermediate node (train station in the city where she lives), get out of the train and get into another taxi at the second intermediate node (train station in the city where her friend lives), and finally get out of the taxi when she arrives at her destination. A kind of encapsulation occurs at the source node, encapsulation and decapsulation occur at the intermediate nodes, and decapsulation occurs at the destination node. Our traveller is the same, but she uses three transporting tools to reach the destination.

Figure 9.3 shows the encapsulation and decapsulation at the data-link layer. For simplicity, we have assumed that we have only one router between the source and destination. The datagram received by the data-link layer of the source host is encapsulated in a frame. The frame is logically transported from the source host to the router. The frame is decapsulated at the data-link layer of the router and encapsulated at another frame. The new frame is logically transported from the router to the destination host. Note that, although we have shown only two data-link layers at the router, the router actually has three data-link layers because it is connected to three physical links.

Figure 9.3 *A communication with only three nodes*



With the contents of the above figure in mind, we can list the services provided by a data-link layer as shown below.

Framing

Definitely, the first service provided by the data-link layer is **framing**. The data-link layer at each node needs to encapsulate the datagram (packet received from the network layer) in a **frame** before sending it to the next node. The node also needs to decapsulate the datagram from the frame received on the logical channel. Although we have shown only a header for a frame, we will see in future chapters that a frame may have both a header and a trailer. Different data-link layers have different formats for framing.

A packet at the data-link layer is normally called a *frame*.

Flow Control

Whenever we have a producer and a consumer, we need to think about flow control. If the producer produces items that cannot be consumed, accumulation of items occurs. The sending data-link layer at the end of a link is a producer of frames; the receiving data-link layer at the other end of a link is a consumer. If the rate of produced frames is higher than the rate of consumed frames, frames at the receiving end need to be buffered while waiting to be consumed (processed). Definitely, we cannot have an unlimited buffer size at the receiving side. We have two choices. The first choice is to let the receiving data-link layer drop the frames if its buffer is full. The second choice is to let the receiving data-link layer send a feedback to the sending data-link layer to ask it to stop or slow down. Different data-link-layer protocols use different strategies for flow control. Since flow control also occurs at the transport layer, with a higher degree of importance, we discuss this issue in Chapter 23 when we talk about the transport layer.

Error Control

At the sending node, a frame in a data-link layer needs to be changed to bits, transformed to electromagnetic signals, and transmitted through the transmission media. At the receiving node, electromagnetic signals are received, transformed to bits, and put together to create a frame. Since electromagnetic signals are susceptible to error, a frame is susceptible to error. The error needs first to be detected. After detection, it needs to be either corrected at the receiver node or discarded and retransmitted by the sending node. Since error detection and correction is an issue in every layer (node-to-node or host-to-host), we have dedicated all of Chapter 10 to this issue.

Congestion Control

Although a link may be congested with frames, which may result in frame loss, most data-link-layer protocols do not directly use a congestion control to alleviate congestion, although some wide-area networks do. In general, congestion control is considered an issue in the network layer or the transport layer because of its end-to-end nature. We will discuss congestion control in the network layer and the transport layer in later chapters.

9.1.3 Two Categories of Links

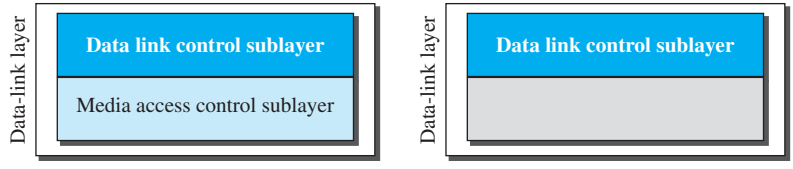
Although two nodes are physically connected by a transmission medium such as cable or air, we need to remember that the data-link layer controls how the medium is used. We can have a data-link layer that uses the whole capacity of the medium; we can also

have a data-link layer that uses only part of the capacity of the link. In other words, we can have a *point-to-point link* or a *broadcast link*. In a point-to-point link, the link is dedicated to the two devices; in a broadcast link, the link is shared between several pairs of devices. For example, when two friends use the traditional home phones to chat, they are using a point-to-point link; when the same two friends use their cellular phones, they are using a broadcast link (the air is shared among many cell phone users).

9.1.4 Two Sublayers

To better understand the functionality of and the services provided by the link layer, we can divide the data-link layer into two sublayers: **data link control (DLC)** and **media access control (MAC).** This is not unusual because, as we will see in later chapters, LAN protocols actually use the same strategy. The data link control sublayer deals with all issues common to both point-to-point and broadcast links; the media access control sublayer deals only with issues specific to broadcast links. In other words, we separate these two types of links at the data-link layer, as shown in Figure 9.4.

Figure 9.4 *Dividing the data-link layer into two sublayers*



a. Data-link layer of a broadcast link

b. Data-link layer of a point-to-point link

We discuss the DLC and MAC sublayers later, each in a separate chapter. In addition, we discuss the issue of error detection and correction, a duty of the data-link and other layers, also in a separate chapter.

9.2 LINK-LAYER ADDRESSING

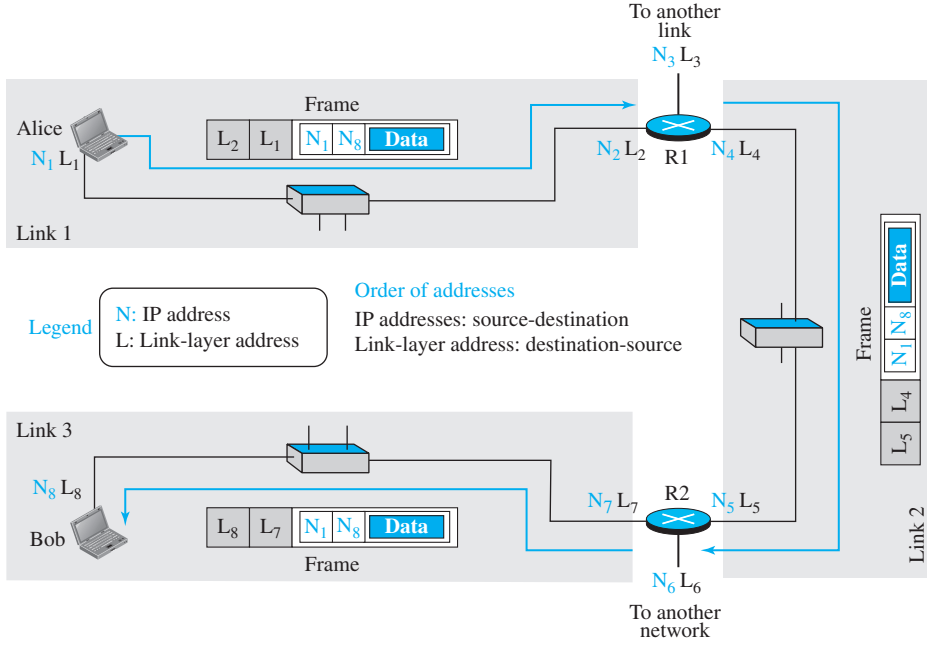
The next issue we need to discuss about the data-link layer is the link-layer addresses. In Chapter 18, we will discuss IP addresses as the identifiers at the network layer that define the exact points in the Internet where the source and destination hosts are connected. However, in a connectionless internetwork such as the Internet we cannot make a datagram reach its destination using only IP addresses. The reason is that each datagram in the Internet, from the same source host to the same destination host, may take a different path. The source and destination IP addresses define the two ends but cannot define which links the datagram should pass through.

We need to remember that the IP addresses in a datagram should not be changed. If the destination IP address in a datagram changes, the packet never reaches its destination; if the source IP address in a datagram changes, the destination host or a router can never communicate with the source if a response needs to be sent back or an error needs to be reported back to the source (see ICMP in Chapter 19).

The above discussion shows that we need another addressing mechanism in a connectionless internetwork: the link-layer addresses of the two nodes. A *link-layer address* is sometimes called a *link address*, sometimes a *physical address*, and sometimes a *MAC address*. We use these terms interchangeably in this book.

Since a link is controlled at the data-link layer, the addresses need to belong to the data-link layer. When a datagram passes from the network layer to the data-link layer, the datagram will be encapsulated in a frame and two data-link addresses are added to the frame header. These two addresses are changed every time the frame moves from one link to another. Figure 9.5 demonstrates the concept in a small internet.

Figure 9.5 *IP addresses and link-layer addresses in a small internet*



In the internet in Figure 9.5, we have three links and two routers. We also have shown only two hosts: Alice (source) and Bob (destination). For each host, we have shown two addresses, the IP addresses (N) and the link-layer addresses (L). Note that a router has as many pairs of addresses as the number of links the router is connected to. We have shown three frames, one in each link. Each frame carries the same datagram with the same source and destination addresses (**N1** and **N8**), but the link-layer addresses of the frame change from link to link. In link 1, the link-layer addresses are L_1 and L_2. In link 2, they are L_4 and L_5. In link 3, they are L_7 and L_8. Note that the IP addresses and the link-layer addresses are not in the same order. For IP addresses, the source address comes before the destination address; for link-layer addresses, the destination address comes before the source. The datagrams and

frames are designed in this way, and we follow the design. We may raise several questions:

- If the IP address of a router does not appear in any datagram sent from a source to a destination, why do we need to assign IP addresses to routers? The answer is that in some protocols a router may act as a sender or receiver of a datagram. For example, in routing protocols we will discuss in Chapters 20 and 21, a router is a sender or a receiver of a message. The communications in these protocols are between routers.
- Why do we need more than one IP address in a router, one for each interface? The answer is that an interface is a connection of a router to a link. We will see that an IP address defines a point in the Internet at which a device is connected. A router with *n* interfaces is connected to the Internet at *n* points. This is the situation of a house at the corner of a street with two gates; each gate has the address related to the corresponding street.
- How are the source and destination IP addresses in a packet determined? The answer is that the host should know its own IP address, which becomes the source IP address in the packet. As we will discuss in Chapter 26, the application layer uses the services of DNS to find the destination address of the packet and passes it to the network layer to be inserted in the packet.
- How are the source and destination link-layer addresses determined for each link? Again, each hop (router or host) should know its own link-layer address, as we discuss later in the chapter. The destination link-layer address is determined by using the Address Resolution Protocol, which we discuss shortly.
- What is the size of link-layer addresses? The answer is that it depends on the protocol used by the link. Although we have only one IP protocol for the whole Internet, we may be using different data-link protocols in different links. This means that we can define the size of the address when we discuss different link-layer protocols.

9.2.1 Three Types of addresses

Some link-layer protocols define three types of addresses: unicast, multicast, and broadcast.

Unicast Address

Each host or each interface of a router is assigned a unicast address. Unicasting means one-to-one communication. A frame with a unicast address destination is destined only for one entity in the link.

Example 9.1

As we will see in Chapter 13, the unicast link-layer addresses in the most common LAN, Ethernet, are 48 bits (six bytes) that are presented as 12 hexadecimal digits separated by colons; for example, the following is a link-layer address of a computer.

A3:34:45:11:92:F1

Multicast Address

Some link-layer protocols define multicast addresses. Multicasting means one-to-many communication. However, the jurisdiction is local (inside the link).

Example 9.2

As we will see in Chapter 13, the multicast link-layer addresses in the most common LAN, Ethernet, are 48 bits (six bytes) that are presented as 12 hexadecimal digits separated by colons. The second digit, however, needs to be an even number in hexadecimal. The following shows a multicast address:

A2:34:45:11:92:F1

Broadcast Address

Some link-layer protocols define a broadcast address. Broadcasting means one-to-all communication. A frame with a destination broadcast address is sent to all entities in the link.

Example 9.3

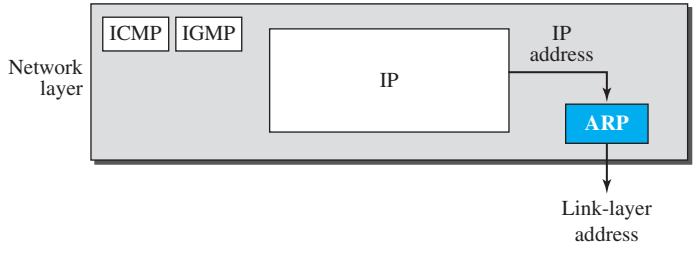
As we will see in Chapter 13, the broadcast link-layer addresses in the most common LAN, Ethernet, are 48 bits, all 1s, that are presented as 12 hexadecimal digits separated by colons. The following shows a broadcast address:

FF:FF:FF:FF:FF:FF

9.2.2 Address Resolution Protocol (ARP)

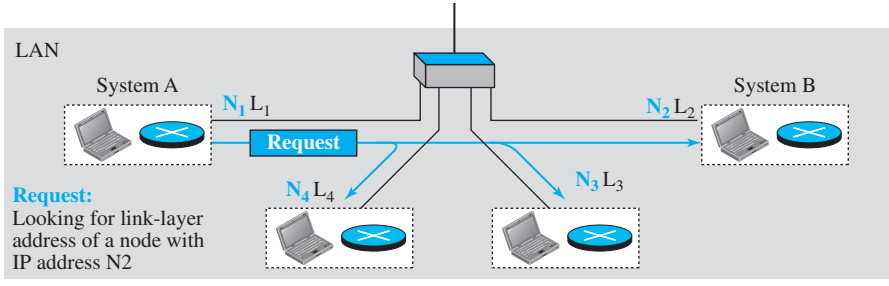
Anytime a node has an IP datagram to send to another node in a link, it has the IP address of the receiving node. The source host knows the IP address of the default router. Each router except the last one in the path gets the IP address of the next router by using its forwarding table. The last router knows the IP address of the destination host. However, the IP address of the next node is not helpful in moving a frame through a link; we need the link-layer address of the next node. This is the time when the **Address Resolution Protocol (ARP)** becomes helpful. The ARP protocol is one of the auxiliary protocols defined in the network layer, as shown in Figure 9.6. It belongs to the network layer, but we discuss it in this chapter because it maps an IP address to a logical-link address. ARP accepts an IP address from the IP protocol, maps the address to the corresponding link-layer address, and passes it to the data-link layer.

Figure 9.6 *Position of ARP in TCP/IP protocol suite*

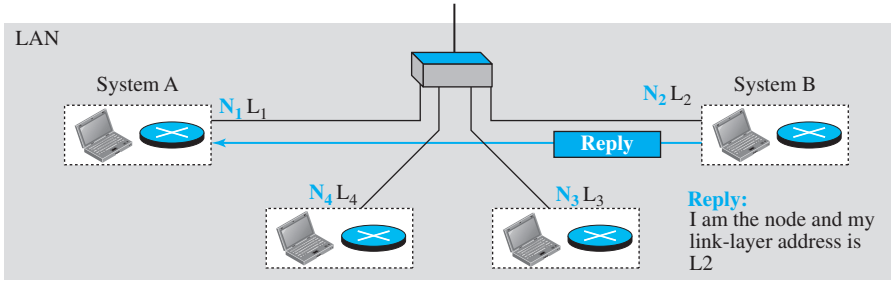


Anytime a host or a router needs to find the link-layer address of another host or router in its network, it sends an ARP request packet. The packet includes the link-layer and IP addresses of the sender and the IP address of the receiver. Because the sender does not know the link-layer address of the receiver, the query is broadcast over the link using the link-layer broadcast address, which we discuss for each protocol later (see Figure 9.7).

Figure 9.7 *ARP operation*



a. ARP request is broadcast



b. ARP reply is unicast

Every host or router on the network receives and processes the ARP request packet, but only the intended recipient recognizes its IP address and sends back an ARP response packet. The response packet contains the recipient's IP and link-layer addresses. The packet is unicast directly to the node that sent the request packet.

In Figure 9.7a, the system on the left (A) has a packet that needs to be delivered to another system (B) with IP address **N2**. System A needs to pass the packet to its data-link layer for the actual delivery, but it does not know the physical address of the recipient. It uses the services of ARP by asking the ARP protocol to send a broadcast ARP request packet to ask for the physical address of a system with an IP address of **N2**.

This packet is received by every system on the physical network, but only system B will answer it, as shown in Figure 9.7b. System B sends an ARP reply packet that includes its physical address. Now system A can send all the packets it has for this destination using the physical address it received.

Caching

A question that is often asked is this: If system A can broadcast a frame to find the link-layer address of system B, why can't system A send the datagram for system B using a broadcast frame? In other words, instead of sending one broadcast frame (ARP request), one unicast frame (ARP response), and another unicast frame (for sending the datagram), system A can encapsulate the datagram and send it to the network. System B receives it and keep it; other systems discard it.

To answer the question, we need to think about the efficiency. It is probable that system A has more than one datagram to send to system B in a short period of time. For example, if system B is supposed to receive a long e-mail or a long file, the data do not fit in one datagram.

Let us assume that there are 20 systems connected to the network (link): system A, system B, and 18 other systems. We also assume that system A has 10 datagrams to send to system B in one second.

a. Without using ARP, system A needs to send 10 broadcast frames. Each of the 18 other systems need to receive the frames, decapsulate the frames, remove the datagram and pass it to their network-layer to find out the datagrams do not belong to them.This means processing and discarding 180 broadcast frames.

b. Using ARP, system A needs to send only one broadcast frame. Each of the 18 other systems need to receive the frames, decapsulate the frames, remove the ARP message and pass the message to their ARP protocol to find that the frame must be discarded. This means processing and discarding only 18 (instead of 180) broadcast frames. After system B responds with its own data-link address, system A can store the link-layer address in its cache memory. The rest of the nine frames are only unicast. Since processing broadcast frames is expensive (time consuming), the first method is preferable.

Packet Format

Figure 9.8 shows the format of an ARP packet. The names of the fields are self-explanatory. The *hardware type* field defines the type of the link-layer protocol; Ethernet is given the type 1. The *protocol type* field defines the network-layer protocol: IPv4 protocol is $(0800)_{16}$. The source hardware and source protocol addresses are variable-length fields defining the link-layer and network-layer addresses of the sender. The destination hardware address and destination protocol address fields define the receiver link-layer and network-layer addresses. An ARP packet is encapsulated directly into a data-link frame. The frame needs to have a field to show that the payload belongs to the ARP and not to the network-layer datagram.

Example 9.4

A host with IP address **N1** and MAC address **L1** has a packet to send to another host with IP address **N2** and physical address **L2** (which is unknown to the first host). The two hosts are on the same network. Figure 9.9 shows the ARP request and response messages.

Figure 9.8 *ARP packet*

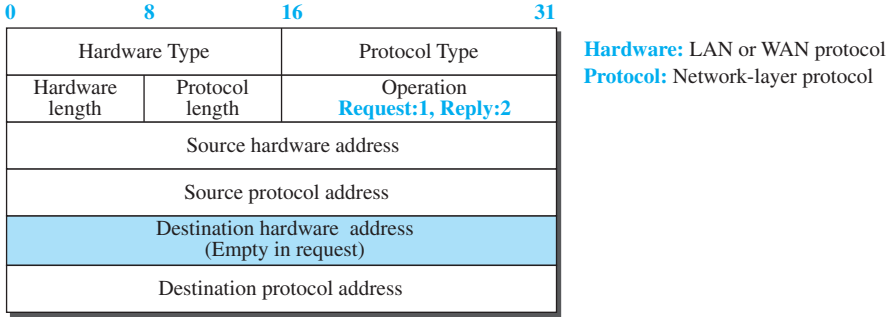
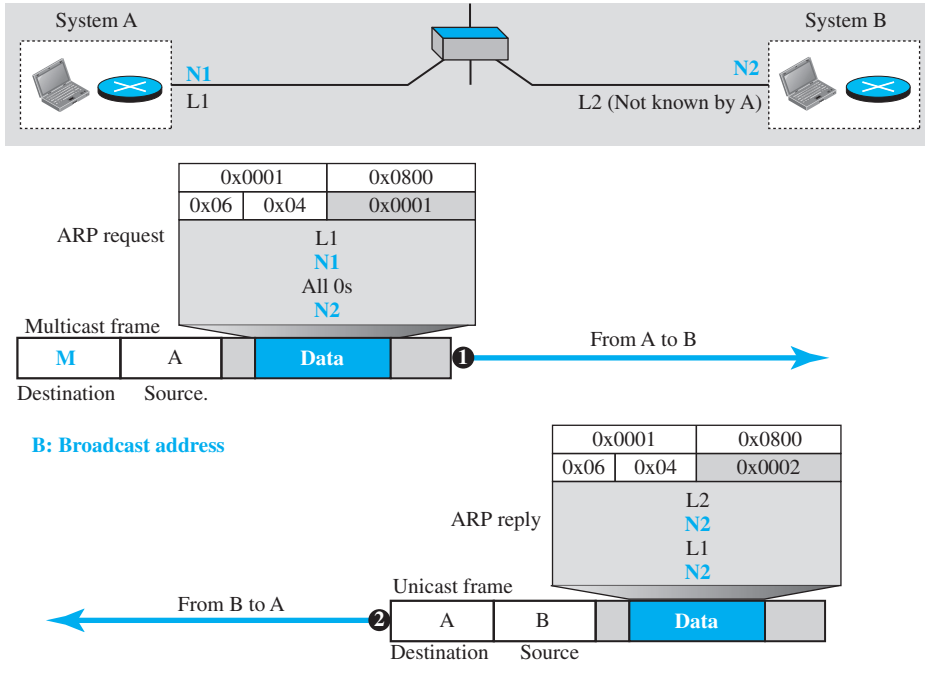


Figure 9.9 *Example 9.4*

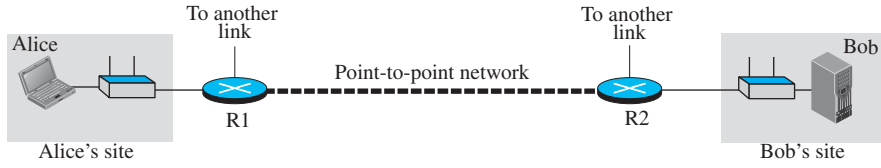


9.2.3 An Example of Communication

To show how communication is done at the data-link layer and how link-layer addresses are found, let us go through a simple example. Assume Alice needs to send a datagram to Bob, who is three nodes away in the Internet. How Alice finds the network-layer address of Bob is what we discover in Chapter 26 when we discuss DNS. For the moment, assume that Alice knows the network-layer (IP) address of Bob. In other words, Alice's host is given the data to be sent, the IP address of Bob, and the

IP address of Alice's host (each host needs to know its IP address). Figure 9.10 shows the part of the internet for our example.

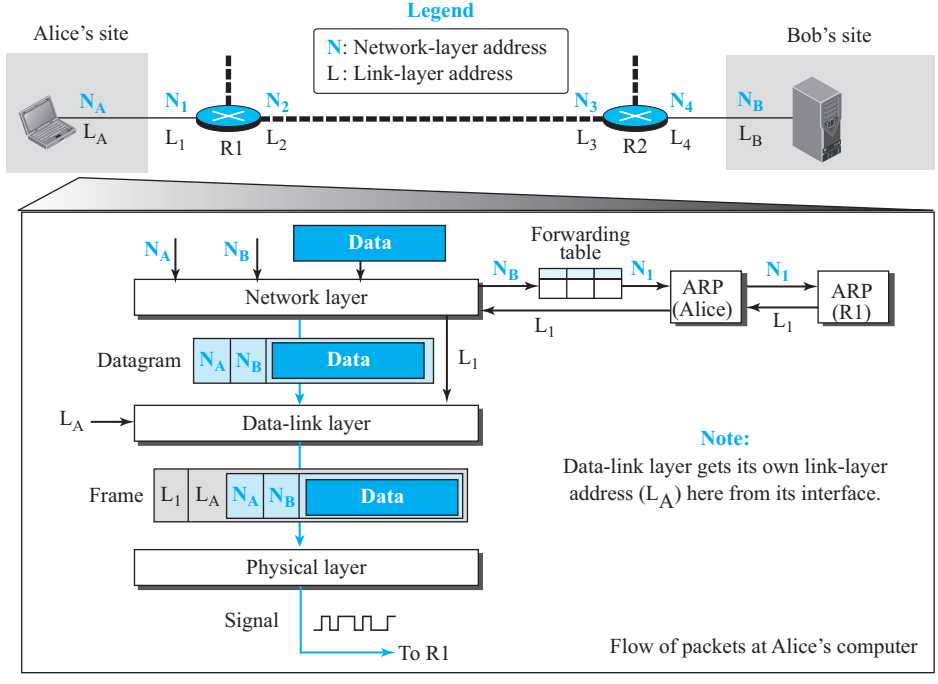
Figure 9.10 *The internet for our example*



Activities at Alice's Site

We will use symbolic addresses to make the figures more readable. Figure 9.11 shows what happens at Alice's site.

Figure 9.11 *Flow of packets at Alice's computer*



The network layer knows it's given N_A, N_B, and the packet, but it needs to find the link-layer address of the next node. The network layer consults its routing table and tries to find which router is next (the default router in this case) for the destination N_B. As we will discuss in Chapter 18, the routing table gives N_1, but the network layer

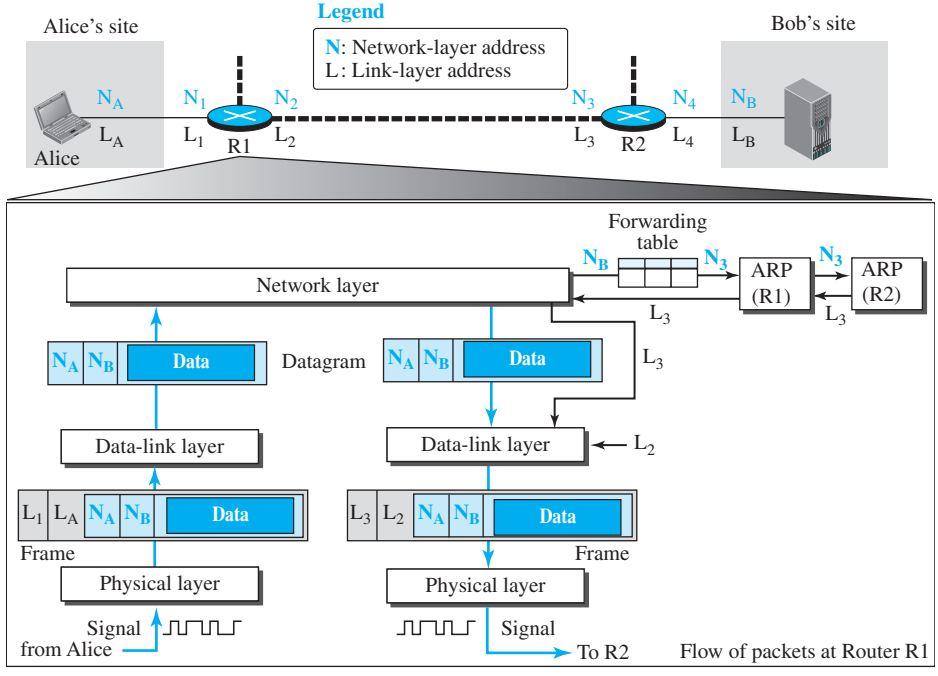
needs to find the link-layer address of router R1. It uses its ARP to find the link-layer address $\mathbf{L_1}$. The network layer can now pass the datagram with the link-layer address to the data-link layer.

The data-link layer knows its own link-layer address, $\mathbf{L_A}$. It creates the frame and passes it to the physical layer, where the address is converted to signals and sent through the media.

Activities at Router R1

Now let us see what happens at Router R1. Router R1, as we know, has only three lower layers. The packet received needs to go up through these three layers and come down. Figure 9.12 shows the activities.

Figure 9.12 *Flow of activities at router R1*



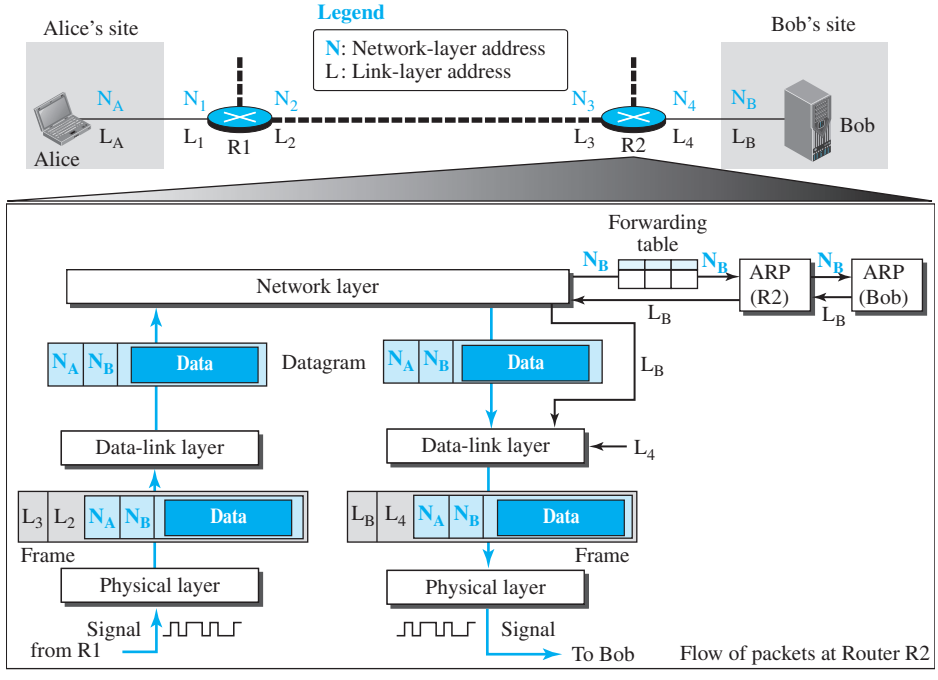
At arrival, the physical layer of the left link creates the frame and passes it to the data-link layer. The data-link layer decapsulates the datagram and passes it to the network layer. The network layer examines the network-layer address of the datagram and finds that the datagram needs to be delivered to the device with IP address $\mathbf{N_B}$. The network layer consults its routing table to find out which is the next node (router) in the path to $\mathbf{N_B}$. The forwarding table returns $\mathbf{N_3}$. The IP address of router R2 is in the same link with R1. The network layer now uses the ARP to find the link-layer address of this router, which comes up as $\mathbf{L_3}$. The network layer passes the datagram and $\mathbf{L_3}$ to the data-link layer belonging to the link at the right side. The link layer

encapsulates the datagram, adds **L3** and **L2** (its own link-layer address), and passes the frame to the physical layer. The physical layer encodes the bits to signals and sends them through the medium to R2.

Activities at Router R2

Activities at router R2 are almost the same as in R1, as shown in Figure 9.13.

Figure 9.13 *Activities at router R2.*



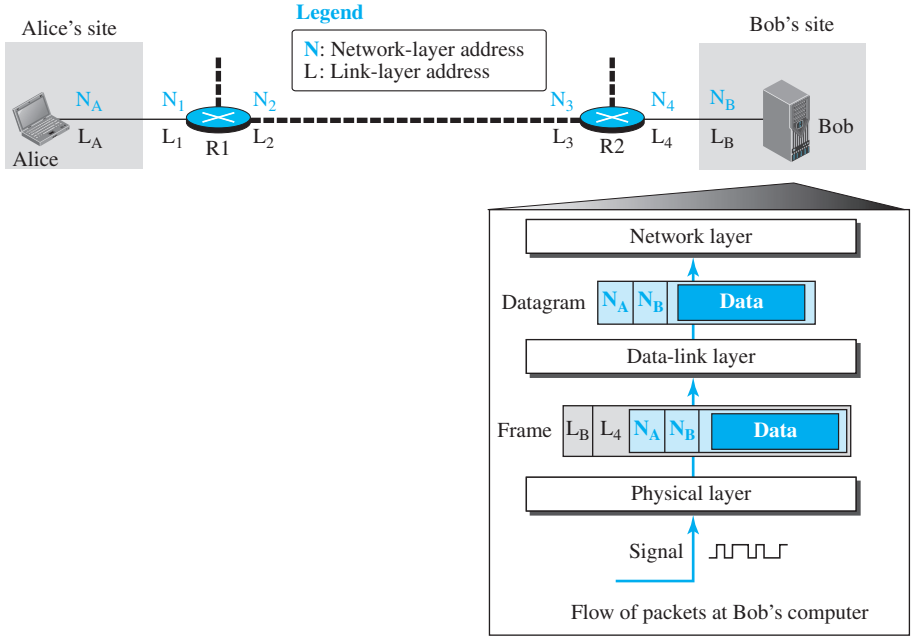
Activities at Bob's Site

Now let us see what happens at Bob's site. Figure 9.14 shows how the signals at Bob's site are changed to a message. At Bob's site there are no more addresses or mapping needed. The signal received from the link is changed to a frame. The frame is passed to the data-link layer, which decapsulates the datagram and passes it to the network layer. The network layer decapsulates the message and passes it to the transport layer.

Changes in Addresses

This example shows that the source and destination network-layer addresses, NA and NB, have not been changed during the whole journey. However, all four network-layer addresses of routers R1 and R2 (N1, N2, N3, and N4) are needed to transfer a datagram from Alice's computer to Bob's computer.

Figure 9.14 *Activities at Bob's site*



9.3 END-CHAPTER MATERIALS

9.3.1 Recommended Reading

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

Books

Several books discuss link-layer issues. Among them we recommend [Ham 80], [Zar 02], [Ror 96], [Tan 03], [GW 04], [For 03], [KMK 04], [Sta 04], [Kes 02], [PD 03], [Kei 02], [Spu 00], [KCK 98], [Sau 98], [Izz 00], [Per 00], and [WV 00].

9.3.2 Key Terms

Address Resolution Protocol (ARP)
data link control (DLC)
frame
framing
links
media access control (MAC)
nodes

9.3.3 Summary

The Internet is made of many hosts, networks, and connecting devices such as routers. The hosts and connecting devices are referred to as *nodes;* the networks are referred to

as *links*. A path in the Internet from a source host to a destination host is a set of nodes and links through which a packet should travel.

The data-link layer is responsible for the creation and delivery of a frame to another node, along the link. It is responsible for packetizing (framing), flow control, error control, and congestion control along the link. Two data-link layers at the two ends of a link coordinate to deliver a frame from one node to the next.

As with any delivery between a source and destination in which there are many paths, we need two types of addressing. The end-to-end addressing defines the source and destination; the link-layer addressing defines the addresses of the nodes that the packet should pass through. To avoid including the link-layer addresses of all of these nodes in the frame, the Address Resolution Protocol (ARP) was devised to map an IP address to its corresponding link-layer address. When a packet is at one node ready to be sent to the next, the forwarding table finds the IP address of the next node and ARP find its link-layer address.

9.4 PRACTICE SET

9.4.1 Quizzes

A set of interactive quizzes for this chapter can be found on the book website. It is strongly recommended that the student take the quizzes to check his/her understanding of the materials before continuing with the practice set.

9.4.2 Questions

Q9-1. Distinguish between communication at the network layer and communication at the data-link layer.

Q9-2. Distinguish between a point-to-point link and a broadcast link.

Q9-3. Can two hosts in two different networks have the same link-layer address? Explain.

Q9-4. Is the size of the ARP packet fixed? Explain.

Q9-5. What is the size of an ARP packet when the protocol is IPv4 and the hardware is Ethernet?

Q9-6. Assume we have an isolated link (not connected to any other link) such as a private network in a company. Do we still need addresses in both the network layer and the data-link layer? Explain.

Q9-7. In Figure 9.9, why is the destination hardware address all 0s in the ARP request message?

Q9-8. In Figure 9.9, why is the destination hardware address of the frame from A to B a broadcast address?

Q9-9. In Figure 9.9, how does system A know what the link-layer address of system B is when it receives the ARP reply?

Q9-10. When we talkabout the broadcast address in a link, do we mean sending a message to all hosts and routers in the link or to all hosts and routers in the Internet? In other words, does a broadcast address have a local jurisdiction or a universal jurisdiction? Explain.

Q9-11. Why does a host or a router need to run the ARP program all of the time in the background?

Q9-12. Why does a router normally have more than one interface?

Q9-13. Why is it better not to change an end-to-end address from the source to the destination?

Q9-14. How many IP addresses and how many link-layer addresses should a router have when it is connected to five links?

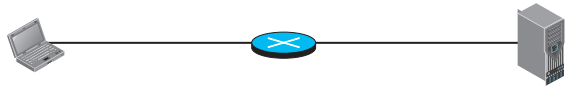
9.4.3 Problems

P9-1. Assume we have an internet (a private small internet) in which all hosts are connected in a mesh topology. Do we need routers in this internet? Explain.

P9-2. In the previous problem, do we need both network and data-link layers?

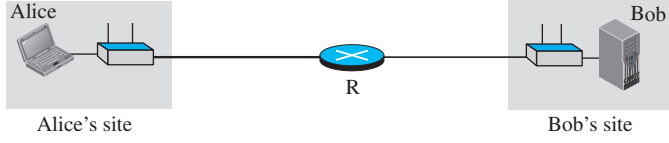
P9-3. Explain why we do not need the router in Figure 9.15.

Figure 9.15 *Problem 9-3*



P9-4. Explain why we may need a router in Figure 9.16.

Figure 9.16 *Problem 9-4*



P9-5. Is the current Internet using circuit-switching or packet-switching at the data-link layer? Explain.

P9-6. Assume Alice is travelling from 2020 Main Street in Los Angeles to 1432 American Boulevard in Chicago. If she is travelling by air from Los Angeles Airport to Chicago Airport,

a. find the end-to-end addresses in this scenario.

b. find the link-layer addresses in this scenario.

P9-7. In the previous problem, assume Alice cannot find a direct flight from the Los Angeles to the Chicago. If she needs to change flights in Denver,

a. find the end-to-end addresses in this scenario.

b. find the link-layer addresses in this scenario.

P9-8. When we send a letter using the services provided by the post office, do we use an end-to-end address? Does the post office necessarily use an end-to-end address to deliver the mail? Explain.

P9-9. In Figure 9.5, assume Link 2 is broken. How can Alice communicate with Bob?

P9-10. In Figure 9.5, show the process of frame change in routers R1 and R2.

P9-11. In Figure 9.7, assume system B is not running the ARP program. What would happen?

P9-12. In Figure 9.7, do you think that system A should first check its cache for mapping from N2 to L2 before even broadcasting the ARP request?

P9-13. Assume the network in Figure 9.7 does not support broadcasting. What do you suggest for sending the ARP request in this network?

P9-14. In Figures 9.11 to 9.13, both the forwarding table and ARP are doing a kind of mapping. Show the difference between them by listing the input and output of mapping for a forwarding table and ARP.

P9-15. Figure 9.7 shows a system as either a host or a router. What would be the actual entity (host or router) of system A and B in each of the following cases:

- **a.** If the link is the first one in the path?
- **b.** If the link is the middle one in the path?
- **c.** If the link is the last one in the path?
- **d.** If there is only one link in the path (local communication)?

CHAPTER 10

Error Detection and Correction

Networks must be able to transfer data from one device to another with acceptable accuracy. For most applications, a system must guarantee that the data received are identical to the data transmitted. Any time data are transmitted from one node to the next, they can become corrupted in passage. Many factors can alter one or more bits of a message. Some applications require a mechanism for detecting and correcting **errors.**

Some applications can tolerate a small level of error. For example, random errors in audio or video transmissions may be tolerable, but when we transfer text, we expect a very high level of accuracy.

At the data-link layer, if a frame is corrupted between the two nodes, it needs to be corrected before it continues its journey to other nodes. However, most link-layer protocols simply discard the frame and let the upper-layer protocols handle the retransmission of the frame. Some multimedia applications, however, try to correct the corrupted frame.

This chapter is divided into five sections.

- The first section introduces types of errors, the concept of redundancy, and distinguishes between error detection and correction.
- The second section discusses block coding. It shows how error can be detected using block coding and also introduces the concept of Hamming distance.
- The third section discusses cyclic codes. It discusses a subset of cyclic code, CRC, that is very common in the data-link layer. The section shows how CRC can be easily implemented in hardware and represented by polynomials.
- The fourth section discusses checksums. It shows how a checksum is calculated for a set of data words. It also gives some other approaches to traditional checksum.
- The fifth section discusses forward error correction. It shows how Hamming distance can also be used for this purpose. The section also describes cheaper methods to achieve the same goal, such as XORing of packets, interleaving chunks, or compounding high and low resolutions packets.

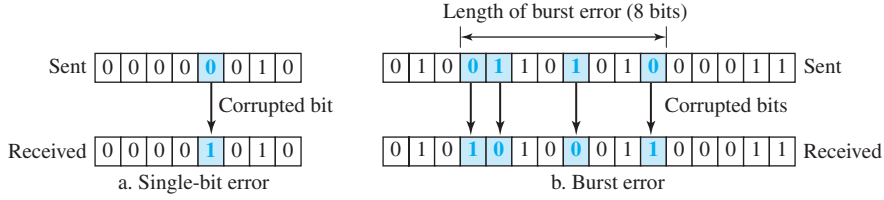
10.1 INTRODUCTION

Let us first discuss some issues related, directly or indirectly, to error detection and correction.

10.1.1 Types of Errors

Whenever bits flow from one point to another, they are subject to unpredictable changes because of **interference.** This interference can change the shape of the signal. The term ***single-bit error*** means that only 1 bit of a given data unit (such as a byte, character, or packet) is changed from 1 to 0 or from 0 to 1. The term ***burst error*** means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1. Figure 10.1 shows the effect of a single-bit and a burst error on a data unit.

Figure 10.1 *Single-bit and burst error*



A burst error is more likely to occur than a single-bit error because the duration of the noise signal is normally longer than the duration of 1 bit, which means that when noise affects data, it affects a set of bits. The number of bits affected depends on the data rate and duration of noise. For example, if we are sending data at 1 kbps, a noise of 1/100 second can affect 10 bits; if we are sending data at 1 Mbps, the same noise can affect 10,000 bits.

10.1.2 Redundancy

The central concept in detecting or correcting errors is **redundancy**. To be able to detect or correct errors, we need to send some extra bits with our data. These redundant bits are added by the sender and removed by the receiver. Their presence allows the receiver to detect or correct corrupted bits.

10.1.3 Detection versus Correction

The correction of errors is more difficult than the detection. In **error detection**, we are only looking to see if any error has occurred. The answer is a simple yes or no. We are not even interested in the number of corrupted bits. A single-bit error is the same for us as a burst error. In **error correction**, we need to know the exact number of bits that are corrupted and, more importantly, their location in the message. The number of errors and the size of the message are important factors. If we need to correct a single error in an 8-bit data unit, we need to consider eight possible error locations; if we need to correct two

errors in a data unit of the same size, we need to consider 28 (permutation of 8 by 2) possibilities. You can imagine the receiver's difficulty in finding 10 errors in a data unit of 1000 bits.

10.1.4 Coding

Redundancy is achieved through various coding schemes. The sender adds redundant bits through a process that creates a relationship between the redundant bits and the actual data bits. The receiver checks the relationships between the two sets of bits to detect errors. The ratio of redundant bits to data bits and the robustness of the process are important factors in any coding scheme.

We can divide coding schemes into two broad categories: **block coding** and **convolution coding**. In this book, we concentrate on block coding; convolution coding is more complex and beyond the scope of this book.

10.2 BLOCK CODING

In block coding, we divide our message into blocks, each of k bits, called ***datawords***. We add r redundant bits to each block to make the length $n = k + r$. The resulting n-bit blocks are called ***codewords***. How the extra r bits are chosen or calculated is something we will discuss later. For the moment, it is important to know that we have a set of datawords, each of size k, and a set of codewords, each of size of n. With k bits, we can create a combination of 2^k datawords; with n bits, we can create a combination of 2^n codewords. Since $n > k$, the number of possible codewords is larger than the number of possible datawords. The block coding process is one-to-one; the same dataword is always encoded as the same codeword. This means that we have $2^n - 2^k$ codewords that are not used. We call these codewords invalid or illegal. The trick in error detection is the existence of these invalid codes, as we discuss next. If the receiver receives an invalid codeword, this indicates that the data was corrupted during transmission.

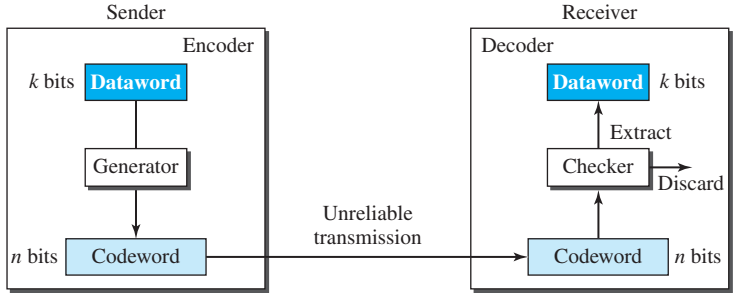
10.2.1 Error Detection

How can errors be detected by using block coding? If the following two conditions are met, the receiver can detect a change in the original codeword.

1. The receiver has (or can find) a list of valid codewords.
2. The original codeword has changed to an invalid one.

Figure 10.2 shows the role of block coding in error detection. The sender creates codewords out of datawords by using a generator that applies the rules and procedures of encoding (discussed later). Each codeword sent to the receiver may change during transmission. If the received codeword is the same as one of the valid codewords, the word is accepted; the corresponding dataword is extracted for use. If the received codeword is not valid, it is discarded. However, if the codeword is corrupted during transmission but the received word still matches a valid codeword, the error remains undetected.

Figure 10.2 *Process of error detection in block coding*



Example 10.1

Let us assume that $k = 2$ and $n = 3$. Table 10.1 shows the list of datawords and codewords. Later, we will see how to derive a codeword from a dataword.

Table 10.1 *A code for error detection in Example 10.1*

Dataword	*Codeword*	*Dataword*	*Codeword*
00	000	10	101
01	011	11	110

Assume the sender encodes the dataword 01 as 011 and sends it to the receiver. Consider the following cases:

1. The receiver receives 011. It is a valid codeword. The receiver extracts the dataword 01 from it.
2. The codeword is corrupted during transmission, and 111 is received (the leftmost bit is corrupted). This is not a valid codeword and is discarded.
3. The codeword is corrupted during transmission, and 000 is received (the right two bits are corrupted). This is a valid codeword. The receiver incorrectly extracts the dataword 00. Two corrupted bits have made the error undetectable.

An error-detecting code can detect only the types of errors for which it is designed; other types of errors may remain undetected.

Hamming Distance

One of the central concepts in coding for error control is the idea of the Hamming distance. The **Hamming distance** between two words (of the same size) is the number of differences between the corresponding bits. We show the Hamming distance between two words x and y as $\boldsymbol{d}(x, y)$. We may wonder why Hamming distance is important for error detection. The reason is that the Hamming distance between the received codeword and the sent codeword is the number of bits that are corrupted during transmission. For example, if the codeword 00000 is sent and 01101 is received, 3 bits are in error and the Hamming distance between the two is $\boldsymbol{d}(00000, 01101) = 3$. In other words, if the Hamming

distance between the sent and the received codeword is not zero, the codeword has been corrupted during transmission.

The Hamming distance can easily be found if we apply the XOR operation ($\oplus$) on the two words and count the number of 1s in the result. Note that the Hamming distance is a value greater than or equal to zero.

The Hamming distance between two words is the number of differences between corresponding bits.

Example 10.2

Let us find the Hamming distance between two pairs of words.

1. The Hamming distance $\boldsymbol{d}(000, 011)$ is 2 because $(000 \oplus 011)$ is 011 (two 1s).
2. The Hamming distance $\boldsymbol{d}(10101, 11110)$ is 3 because $(10101 \oplus 11110)$ is 01011 (three 1s).

Minimum Hamming Distance for Error Detection

In a set of codewords, the **minimum Hamming distance** is the smallest Hamming distance between all possible pairs of codewords. Now let us find the minimum Hamming distance in a code if we want to be able to detect up to s errors. If s errors occur during transmission, the Hamming distance between the sent codeword and received codeword is s. If our system is to detect up to s errors, the minimum distance between the valid codes must be $(s + 1)$, so that the received codeword does not match a valid codeword. In other words, if the minimum distance between all valid codewords is $(s + 1)$, the received codeword cannot be erroneously mistaken for another codeword. The error will be detected. We need to clarify a point here: Although a code with $d_{min} = s + 1$ may be able to detect more than s errors in some special cases, only s or fewer errors are guaranteed to be detected.

To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = s + 1$.

We can look at this criteria geometrically. Let us assume that the sent codeword x is at the center of a circle with radius s. All received codewords that are created by 0 to s errors are points inside the circle or on the perimeter of the circle. All other valid codewords must be outside the circle, as shown in Figure 10.3. This means that d_{min} must be an integer greater than s or $d_{min} = s + 1$.

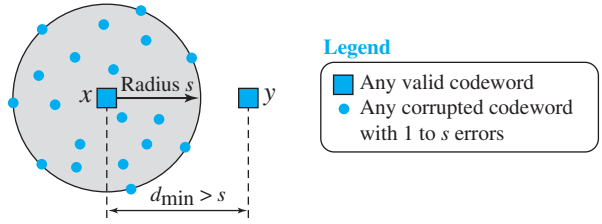
Example 10.3

The minimum Hamming distance for our first code scheme (Table 10.1) is 2. This code guarantees detection of only a single error. For example, if the third codeword (101) is sent and one error occurs, the received codeword does not match any valid codeword. If two errors occur, however, the received codeword may match a valid codeword and the errors are not detected.

Example 10.4

A code scheme has a Hamming distance $d_{min} = 4$. This code guarantees the detection of up to three errors ($d = s + 1$ or $s = 3$).

Figure 10.3 *Geometric concept explaining* d_{min} *in error detection*



Linear Block Codes

Almost all block codes used today belong to a subset of block codes called ***linear block codes***. The use of nonlinear block codes for error detection and correction is not as widespread because their structure makes theoretical analysis and implementation difficult. We therefore concentrate on linear block codes. The formal definition of linear block codes requires the knowledge of abstract algebra (particularly Galois fields), which is beyond the scope of this book. We therefore give an informal definition. For our purposes, a linear block code is a code in which the exclusive OR (addition modulo-2) of two valid codewords creates another valid codeword.

Example 10.5

The code in Table 10.1 is a linear block code because the result of XORing any codeword with any other codeword is a valid codeword. For example, the XORing of the second and third codewords creates the fourth one.

Minimum Distance for Linear Block Codes

It is simple to find the minimum Hamming distance for a linear block code. The minimum Hamming distance is the number of 1s in the nonzero valid codeword with the smallest number of 1s.

Example 10.6

In our first code (Table 10.1), the numbers of 1s in the nonzero codewords are 2, 2, and 2. So the minimum Hamming distance is $d_{\text{min}} = 2$.

Parity-Check Code

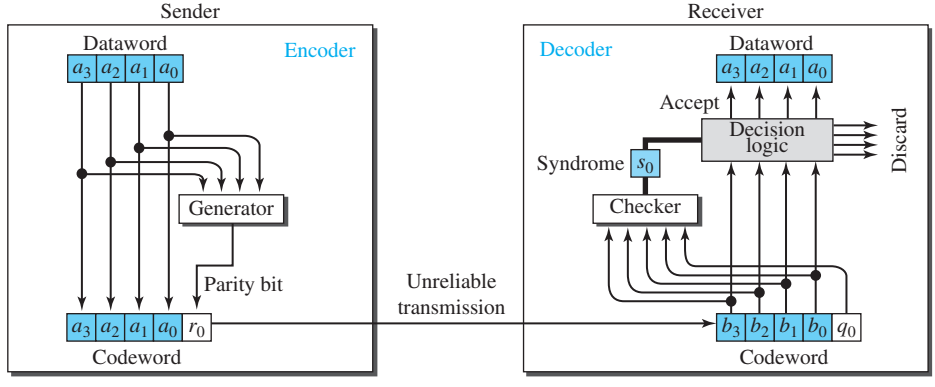
Perhaps the most familiar error-detecting code is the **parity-check code.** This code is a linear block code. In this code, a k-bit dataword is changed to an n-bit codeword where $n = k + 1$. The extra bit, called the *parity bit,* is selected to make the total number of 1s in the codeword even. Although some implementations specify an odd number of 1s, we discuss the even case. The minimum Hamming distance for this category is $d_{\text{min}} = 2$, which means that the code is a single-bit error-detecting code. Our first code (Table 10.1) is a parity-check code ($k = 2$ and $n = 3$). The code in Table 10.2 is also a parity-check code with $k = 4$ and $n = 5$.

Table 10.2 *Simple parity-check code C(5, 4)*

Dataword	*Codeword*	*Dataword*	*Codeword*
0000	**00000**	1000	**10001**
0001	**00011**	1001	**10010**
0010	**00101**	1010	**10100**
0011	**00110**	1011	**10111**
0100	**01001**	1100	**11000**
0101	**01010**	1101	**11011**
0110	**01100**	1110	**11101**
0111	**01111**	1111	**11110**

Figure 10.4 shows a possible structure of an encoder (at the sender) and a decoder (at the receiver).

Figure 10.4 *Encoder and decoder for simple parity-check code*



The calculation is done in **modular arithmetic** (see Appendix E). The encoder uses a generator that takes a copy of a 4-bit dataword (a_0, a_1, a_2, and a_3) and generates a parity bit r_0. The dataword bits and the parity bit create the 5-bit codeword. The parity bit that is added makes the number of 1s in the codeword even. This is normally done by adding the 4 bits of the dataword (modulo-2); the result is the parity bit. In other words,

$$r_0 = a_3 + a_2 + a_1 + a_0 \qquad \textbf{(modulo-2)}$$

If the number of 1s is even, the result is 0; if the number of 1s is odd, the result is 1. In both cases, the total number of 1s in the codeword is even.

The sender sends the codeword, which may be corrupted during transmission. The receiver receives a 5-bit word. The checker at the receiver does the same thing as the generator in the sender with one exception: The addition is done over all 5 bits. The result,

which is called the ***syndrome***, is just 1 bit. The syndrome is 0 when the number of 1s in the received codeword is even; otherwise, it is 1.

$$s_0 = b_3 + b_2 + b_1 + b_0 + q_0 \quad \textbf{(modulo-2)}$$

The syndrome is passed to the decision logic analyzer. If the syndrome is 0, there is no detectable error in the received codeword; the data portion of the received codeword is accepted as the dataword; if the syndrome is 1, the data portion of the received codeword is discarded. The dataword is not created.

Example 10.7

Let us look at some transmission scenarios. Assume the sender sends the dataword 1011. The codeword created from this dataword is 10111, which is sent to the receiver. We examine five cases:

1. No error occurs; the received codeword is 10111. The syndrome is 0. The dataword 1011 is created.
2. One single-bit error changes a_1. The received codeword is 10**0**11. The syndrome is 1. No dataword is created.
3. One single-bit error changes r_0. The received codeword is 1011**0**. The syndrome is 1. No dataword is created. Note that although none of the dataword bits are corrupted, no dataword is created because the code is not sophisticated enough to show the position of the corrupted bit.
4. An error changes r_0 and a second error changes a_3. The received codeword is **0**0110. The syndrome is 0. The dataword 0011 is created at the receiver. Note that here the dataword is wrongly created due to the syndrome value. The simple parity-check decoder cannot detect an even number of errors. The errors cancel each other out and give the syndrome a value of 0.
5. Three bits—a_3, a_2, and a_1—are changed by errors. The received codeword is **010**11. The syndrome is 1. The dataword is not created. This shows that the simple parity check, guaranteed to detect one single error, can also find any odd number of errors.

A parity-check code can detect an odd number of errors.

10.3 CYCLIC CODES

Cyclic codes are special linear block codes with one extra property. In a **cyclic code,** if a codeword is cyclically shifted (rotated), the result is another codeword. For example, if 1011000 is a codeword and we cyclically left-shift, then 0110001 is also a codeword. In this case, if we call the bits in the first word a_0 to a_6, and the bits in the second word b_0 to b_6, we can shift the bits by using the following:

$$b_1 = a_0 \qquad b_2 = a_1 \qquad b_3 = a_2 \qquad b_4 = a_3 \qquad b_5 = a_4 \qquad b_6 = a_5 \qquad b_0 = a_6$$

In the rightmost equation, the last bit of the first word is wrapped around and becomes the first bit of the second word.

10.3.1 Cyclic Redundancy Check

We can create cyclic codes to correct errors. However, the theoretical background required is beyond the scope of this book. In this section, we simply discuss a subset of

cyclic codes called the **cyclic redundancy check (CRC)**, which is used in networks such as LANs and WANs.

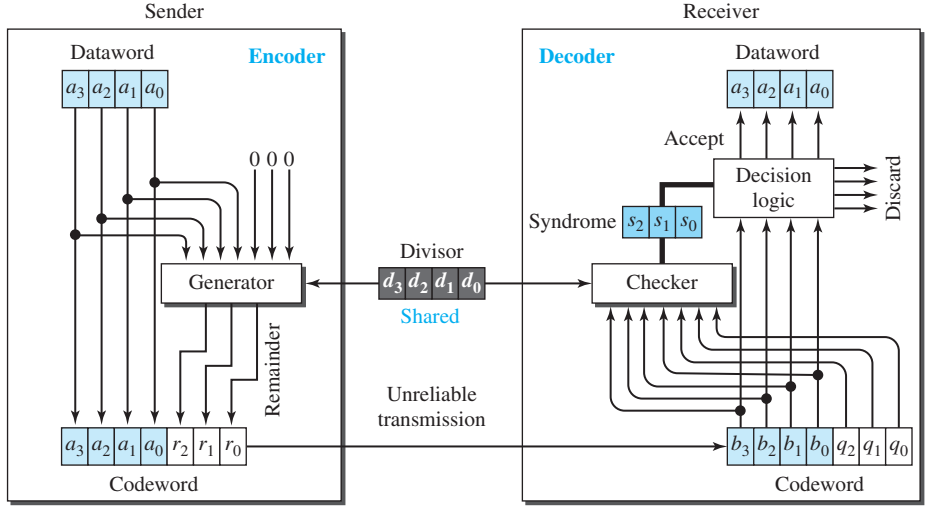
Table 10.3 shows an example of a CRC code. We can see both the linear and cyclic properties of this code.

Table 10.3 *A CRC code with C(7, 4)*

Dataword	*Codeword*	*Dataword*	*Codeword*
0000	0000000	1000	1000101
0001	0001011	1001	1001110
0010	0010110	1010	1010011
0011	0011101	1011	1011000
0100	0100111	1100	1100010
0101	0101100	1101	1101001
0110	0110001	1110	1110100
0111	0111010	1111	1111111

Figure 10.5 shows one possible design for the encoder and decoder.

Figure 10.5 *CRC encoder and decoder*



In the encoder, the dataword has k bits (4 here); the codeword has n bits (7 here). The size of the dataword is augmented by adding $n - k$ (3 here) 0s to the right-hand side of the word. The n-bit result is fed into the generator. The generator uses a divisor of size $n - k + 1$ (4 here), predefined and agreed upon. The generator divides the augmented dataword by the divisor (modulo-2 division). The quotient of the division is discarded; the remainder ($r_2 r_1 r_0$) is appended to the dataword to create the codeword.

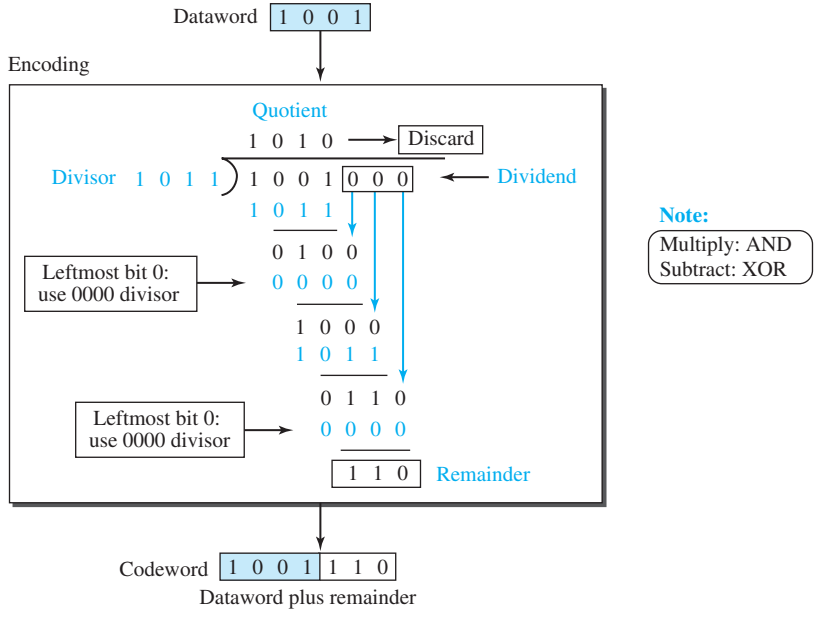
The decoder receives the codeword (possibly corrupted in transition). A copy of all n bits is fed to the checker, which is a replica of the generator. The remainder produced

by the checker is a syndrome of $n - k$ (3 here) bits, which is fed to the decision logic analyzer. The analyzer has a simple function. If the syndrome bits are all 0s, the 4 leftmost bits of the codeword are accepted as the dataword (interpreted as no error); otherwise, the 4 bits are discarded (error).

Encoder

Let us take a closer look at the encoder. The encoder takes a dataword and augments it with $n - k$ number of 0s. It then divides the augmented dataword by the divisor, as shown in Figure 10.6.

Figure 10.6 *Division in CRC encoder*



The process of modulo-2 binary division is the same as the familiar division process we use for decimal numbers. However, addition and subtraction in this case are the same; we use the XOR operation to do both.

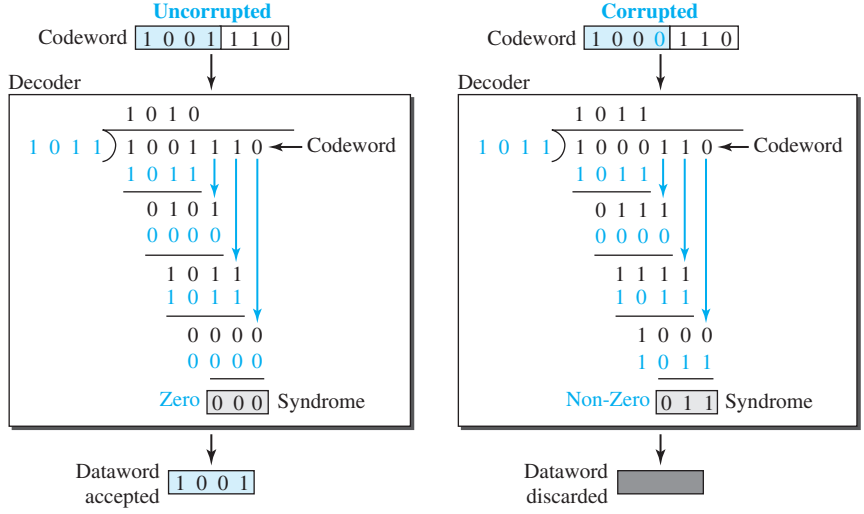
As in decimal division, the process is done step by step. In each step, a copy of the divisor is XORed with the 4 bits of the dividend. The result of the XOR operation (remainder) is 3 bits (in this case), which is used for the next step after 1 extra bit is pulled down to make it 4 bits long. There is one important point we need to remember in this type of division. If the leftmost bit of the dividend (or the part used in each step) is 0, the step cannot use the regular divisor; we need to use an all-0s divisor.

When there are no bits left to pull down, we have a result. The 3-bit remainder forms the **check bits** (r_2, r_1, and r_0). They are appended to the dataword to create the codeword.

Decoder

The codeword can change during transmission. The decoder does the same division process as the encoder. The remainder of the division is the syndrome. If the syndrome is all 0s, there is no error with a high probability; the dataword is separated from the received codeword and accepted. Otherwise, everything is discarded. Figure 10.7 shows two cases: The left-hand figure shows the value of the syndrome when no error has occurred; the syndrome is 000. The right-hand part of the figure shows the case in which there is a single error. The syndrome is not all 0s (it is 011).

Figure 10.7 *Division in the CRC decoder for two cases*



Divisor

We may be wondering how the divisor 1011 is chosen. This depends on the expectation we have from the code. We will show some standard divisors later in the chapter (Table 10.4) after we discuss polynomials.

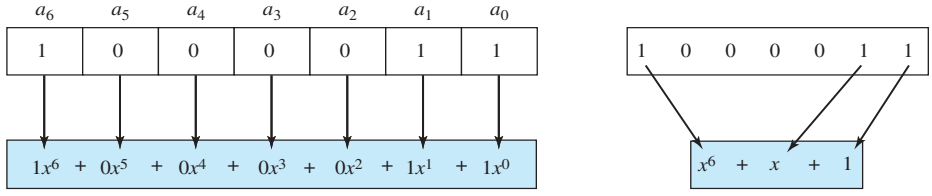
10.3.2 Polynomials

A better way to understand cyclic codes and how they can be analyzed is to represent them as polynomials. Again, this section is optional.

A pattern of 0s and 1s can be represented as a **polynomial** with coefficients of 0 and 1. The power of each term shows the position of the bit; the coefficient shows the value of the bit. Figure 10.8 shows a binary pattern and its polynomial representation. In Figure 10.8a we show how to translate a binary pattern into a polynomial; in Figure 10.8b we show how the polynomial can be shortened by removing all terms with zero coefficients and replacing x^1 by x and x^0 by 1.

Figure 10.8 shows one immediate benefit; a 7-bit pattern can be replaced by three terms. The benefit is even more conspicuous when we have a polynomial such as

Figure 10.8 *A polynomial to represent a binary word*



a. Binary pattern and polynomial

b. Short form

$x^{23} + x^3 + 1$. Here the bit pattern is 24 bits in length (three 1s and twenty-one 0s) while the polynomial is just three terms.

Degree of a Polynomial

The degree of a polynomial is the highest power in the polynomial. For example, the degree of the polynomial $x^6 + x + 1$ is 6. Note that the degree of a polynomial is 1 less than the number of bits in the pattern. The bit pattern in this case has 7 bits.

Adding and Subtracting Polynomials

Adding and subtracting polynomials in mathematics are done by adding or subtracting the coefficients of terms with the same power. In our case, the coefficients are only 0 and 1, and adding is in modulo-2. This has two consequences. First, addition and subtraction are the same. Second, adding or subtracting is done by combining terms and deleting pairs of identical terms. For example, adding $x^5 + x^4 + x^2$ and $x^6 + x^4 + x^2$ gives just $x^6 + x^5$. The terms x^4 and x^2 are deleted. However, note that if we add, for example, three polynomials and we get x^2 three times, we delete a pair of them and keep the third.

Multiplying or Dividing Terms

In this arithmetic, multiplying a term by another term is very simple; we just add the powers. For example, $x^3 \times x^4$ is x^7. For dividing, we just subtract the power of the second term from the power of the first. For example, x^5/x^2 is x^3.

Multiplying Two Polynomials

Multiplying a polynomial by another is done term by term. Each term of the first polynomial must be multiplied by all terms of the second. The result, of course, is then simplified, and pairs of equal terms are deleted. The following is an example:

$$(x^5 + x^3 + x^2 + x)(x^2 + x + 1) = x^7 + x^6 + x^5 + x^5 + x^4 + x^3 + x^4 + x^3 + x^2 + x^3 + x^2 + x$$
$$= x^7 + x^6 + x^3 + x$$

Dividing One Polynomial by Another

Division of polynomials is conceptually the same as the binary division we discussed for an encoder. We divide the first term of the dividend by the first term of the divisor to get the first term of the quotient. We multiply the term in the quotient by the divisor and

subtract the result from the dividend. We repeat the process until the dividend degree is less than the divisor degree. We will show an example of division later in this chapter.

Shifting

A binary pattern is often shifted a number of bits to the right or left. Shifting to the left means adding extra 0s as rightmost bits; shifting to the right means deleting some rightmost bits. Shifting to the left is accomplished by multiplying each term of the polynomial by x^m, where m is the number of shifted bits; shifting to the right is accomplished by dividing each term of the polynomial by x^m. The following shows shifting to the left and to the right. Note that we do not have negative powers in the polynomial representation.

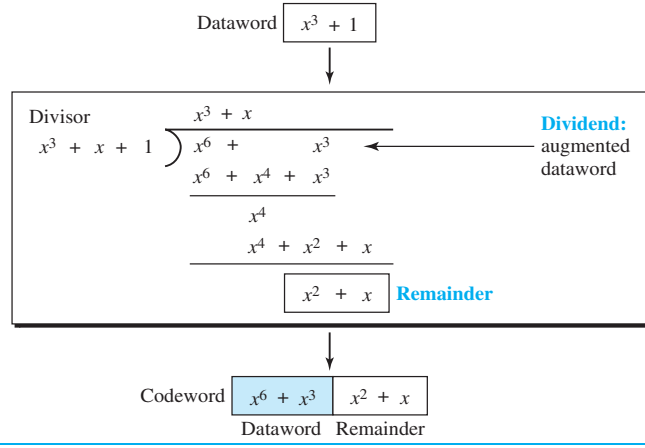
Shifting left 3 bits: 10011 becomes 10011000	$x^4 + x + 1$ becomes $x^7 + x^4 + x^3$
Shifting right 3 bits: 10011 becomes 10	$x^4 + x + 1$ becomes x

When we augmented the dataword in the encoder of Figure 10.6, we actually shifted the bits to the left. Also note that when we concatenate two bit patterns, we shift the first polynomial to the left and then add the second polynomial.

10.3.3 Cyclic Code Encoder Using Polynomials

Now that we have discussed operations on polynomials, we show the creation of a codeword from a dataword. Figure 10.9 is the polynomial version of Figure 10.6. We can see that the process is shorter. The dataword 1001 is represented as $x^3 + 1$. The divisor 1011 is represented as $x^3 + x + 1$. To find the augmented dataword, we have left-shifted the dataword 3 bits (multiplying by x^3). The result is $x^6 + x^3$. Division is straightforward. We divide the first term of the dividend, x^6, by the first term of the divisor, x^3. The first term of the quotient is then x^6/x^3, or x^3. Then we multiply x^3 by the divisor and subtract (according to our previous definition of subtraction) the result from the dividend. The result is x^4, with a degree greater than the divisor's degree; we continue to divide until the degree of the remainder is less than the degree of the divisor.

Figure 10.9 *CRC division using polynomials*



It can be seen that the polynomial representation can easily simplify the operation of division in this case, because the two steps involving all-0s divisors are not needed here. (Of course, one could argue that the all-0s divisor step can also be eliminated in binary division.) In a polynomial representation, the divisor is normally referred to as the ***generator polynomial*** $t(x)$.

The divisor in a cyclic code is normally called the *generator polynomial* or simply the *generator.*

10.3.4 Cyclic Code Analysis

We can analyze a cyclic code to find its capabilities by using polynomials. We define the following, where $f(x)$ is a polynomial with binary coefficients.

Dataword: $d(x)$ **Codeword: $c(x)$** **Generator: $g(x)$** **Syndrome: $s(x)$** **Error: $e(x)$**

If $s(x)$ is not zero, then one or more bits is corrupted. However, if $s(x)$ is zero, either no bit is corrupted or the decoder failed to detect any errors. (Note that ¦ means divide).

In a cyclic code,

1. If $s(x) \nmid 0$, one or more bits is corrupted.
2. If $s(x) = 0$, either
   a. No bit is corrupted, or
   b. Some bits are corrupted, but the decoder failed to detect them.

In our analysis we want to find the criteria that must be imposed on the generator, $g(x)$ to detect the type of error we especially want to be detected. Let us first find the relationship among the sent codeword, error, received codeword, and the generator. We can say

$$\textbf{Received codeword} = c(x) + e(x)$$

In other words, the received codeword is the sum of the sent codeword and the error. The receiver divides the received codeword by $g(x)$ to get the syndrome. We can write this as

$$\frac{\textbf{Received codeword}}{g(x)} = \frac{c(x)}{g(x)} + \frac{e(x)}{g(x)}$$

The first term at the right-hand side of the equality has a remainder of zero (according to the definition of codeword). So the syndrome is actually the remainder of the second term on the right-hand side. If this term does not have a remainder (syndrome = 0), either $e(x)$ is 0 or $e(x)$ is divisible by $g(x)$. We do not have to worry about the first case (there is no error); the second case is very important. Those errors that are divisible by $g(x)$ are not caught.

In a cyclic code, those $e(x)$ errors that are divisible by $g(x)$ are not caught.

Let us show some specific errors and see how they can be caught by a well-designed $g(x)$.

Single-Bit Error

What should the structure of $g(x)$ be to guarantee the detection of a single-bit error? A single-bit error is $e(x) = x^i$, where i is the position of the bit. If a single-bit error is caught, then x^i is not divisible by $g(x)$. (Note that when we say *not divisible,* we mean that there is a remainder.) If $g(x)$ has at least two terms (which is normally the case) and the coefficient of x^0 is not zero (the rightmost bit is 1), then $e(x)$ cannot be divided by $g(x)$.

If the generator has more than one term and the coefficient of x^0 is 1, all single-bit errors can be caught.

Example 10.8

Which of the following $g(x)$ values guarantees that a single-bit error is caught? For each case, what is the error that cannot be caught?

a. $x + 1$
b. x^3
c. 1

Solution

a. No x^i can be divisible by $x + 1$. In other words, $x^i/(x + 1)$ always has a remainder. So the syndrome is nonzero. Any single-bit error can be caught.

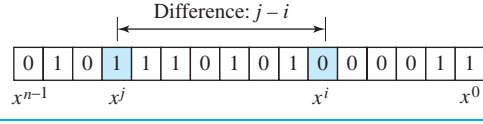
b. If i is equal to or greater than 3, x^i is divisible by $g(x)$. The remainder of x^i/x^3 is zero, and the receiver is fooled into believing that there is no error, although there might be one. Note that in this case, the corrupted bit must be in position 4 or above. All single-bit errors in positions 1 to 3 are caught.

c. All values of i make x^i divisible by $g(x)$. No single-bit error can be caught. In addition, this $g(x)$ is useless because it means the codeword is just the dataword augmented with $n - k$ zeros.

Two Isolated Single-Bit Errors

Now imagine there are two single-bit isolated errors. Under what conditions can this type of error be caught? We can show this type of error as $e(x) = x^j + x^i$. The values of i and j define the positions of the errors, and the difference $j - i$ defines the distance between the two errors, as shown in Figure 10.10.

Figure 10.10 *Representation of two isolated single-bit errors using polynomials*



We can write $e(x) = x^i(x^{j-i} + 1)$. If $g(x)$ has more than one term and one term is x^0, it cannot divide x^i, as we saw in the previous section. So if $g(x)$ is to divide $e(x)$, it must divide $x^{j-i} + 1$. In other words, $g(x)$ must not divide $x^t + 1$, where t is between 0 and $n - 1$. However, $t = 0$ is meaningless and $t = 1$ is needed, as we will see later. This means t should be between 2 and $n - 1$.

If a generator cannot divide $x^t + 1$ (t between 0 and $n - 1$), then all isolated double errors can be detected.

Example 10.9

Find the status of the following generators related to two isolated, single-bit errors.

a. $x + 1$
b. $x^4 + 1$
c. $x^7 + x^6 + 1$
d. $x^{15} + x^{14} + 1$

Solution

a. This is a very poor choice for a generator. Any two errors next to each other cannot be detected.
b. This generator cannot detect two errors that are four positions apart. The two errors can be anywhere, but if their distance is 4, they remain undetected.
c. This is a good choice for this purpose.
d. This polynomial cannot divide any error of type $x^t + 1$ if t is less than 32,768. This means that a codeword with two isolated errors that are next to each other or up to 32,768 bits apart can be detected by this generator.

Odd Numbers of Errors

A generator with a factor of $x + 1$ can catch all odd numbers of errors. This means that we need to make $x + 1$ a factor of any generator. Note that we are not saying that the generator itself should be $x + 1$; we are saying that it should have a factor of $x + 1$. If it is only $x + 1$, it cannot catch the two adjacent isolated errors (see the previous section). For example, $x^4 + x^2 + x + 1$ can catch all odd-numbered errors since it can be written as a product of the two polynomials $x + 1$ and $x^3 + x^2 + 1$.

A generator that contains a factor of $x + 1$ can detect all odd-numbered errors.

Burst Errors

Now let us extend our analysis to the burst error, which is the most important of all. A burst error is of the form $e(x) = (x^j + \cdots + x^i)$. Note the difference between a burst error and two isolated single-bit errors. The first can have two terms or more; the second can only have two terms. We can factor out x^i and write the error as $x^i(x^{j-i} + \cdots + 1)$. If our generator can detect a single error (minimum condition for a generator), then it cannot divide x^i. What we should worry about are those generators that divide $x^{j-i} + \cdots + 1$. In other words, the remainder of $(x^{j-i} + \cdots + 1)/(x^r + \cdots + 1)$ must not be zero. Note that the denominator is the generator polynomial. We can have three cases:

1. If $j - i < r$, the remainder can never be zero. We can write $j - i = L - 1$, where L is the length of the error. So $L - 1 < r$ or $L < r + 1$ or L ð r. This means all burst errors with length smaller than or equal to the number of check bits r will be detected.
2. In some rare cases, if $j - i = r$, or $L = r + 1$, the syndrome is 0 and the error is undetected. It can be proved that in these cases, the probability of undetected burst error of length $r + 1$ is $(1/2)^{r-1}$. For example, if our generator is $x^{14} + x^3 + 1$, in which $r = 14$, a burst error of length $L = 15$ can slip by undetected with the probability of $(1/2)^{14-1}$ or almost 1 in 10,000.
3. In some rare cases, if $j - i > r$, or $L > r + 1$, the syndrome is 0 and the error is undetected. It can be proved that in these cases, the probability of undetected burst error of length greater than $r + 1$ is $(1/2)^r$. For example, if our generator is $x^{14} + x^3 + 1$, in which $r = 14$, a burst error of length greater than 15 can slip by undetected with the probability of $(1/2)^{14}$ or almost 1 in 16,000 cases.

- **All burst errors with $L \leq r$ will be detected.**
- **All burst errors with $L = r + 1$ will be detected with probability $1 - (1/2)^{r-1}$.**
- **All burst errors with $L > r + 1$ will be detected with probability $1 - (1/2)^r$.**

Example 10.10

Find the suitability of the following generators in relation to burst errors of different lengths.

a. $x^6 + 1$
b. $x^{18} + x^7 + x + 1$
c. $x^{32} + x^{23} + x^7 + 1$

Solution

a. This generator can detect all burst errors with a length less than or equal to 6 bits; 3 out of 100 burst errors with length 7 will slip by; 16 out of 1000 burst errors of length 8 or more will slip by.
b. This generator can detect all burst errors with a length less than or equal to 18 bits; 8 out of 1 million burst errors with length 19 will slip by; 4 out of 1 million burst errors of length 20 or more will slip by.
c. This generator can detect all burst errors with a length less than or equal to 32 bits; 5 out of 10 billion burst errors with length 33 will slip by; 3 out of 10 billion burst errors of length 34 or more will slip by.

Summary

We can summarize the criteria for a good polynomial generator:

A good polynomial generator needs to have the following characteristics:

1. **It should have at least two terms.**
2. **The coefficient of the term x^0 should be 1.**
3. **It should not divide $x^t + 1$, for t between 2 and $n - 1$.**
4. **It should have the factor $x + 1$.**

Standard Polynomials

Some standard polynomials used by popular protocols for CRC generation are shown in Table 10.4 along with the corresponding bit pattern.

Table 10.4 *Standard polynomials*

Name	*Polynomial*	*Used in*
CRC-8	$x^8 + x^2 + x + 1$ **100000111**	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$ **11000110101**	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$ **10001000000100001**	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ **10000010011000001000111011011 0111**	LANs

10.3.5 Advantages of Cyclic Codes

We have seen that cyclic codes have a very good performance in detecting single-bit errors, double errors, an odd number of errors, and burst errors. They can easily be implemented in hardware and software. They are especially fast when implemented in hardware. This has made cyclic codes a good candidate for many networks.

10.3.6 Other Cyclic Codes

The cyclic codes we have discussed in this section are very simple. The check bits and syndromes can be calculated by simple algebra. There are, however, more powerful polynomials that are based on abstract algebra involving Galois fields. These are beyond the scope of this book. One of the most interesting of these codes is the **Reed-Solomon code** used today for both detection and correction.

10.3.7 Hardware Implementation

One of the advantages of a cyclic code is that the encoder and decoder can easily and cheaply be implemented in hardware by using a handful of electronic devices. Also, a hardware implementation increases the rate of check bit and syndrome bit calculation. In this section, we try to show, step by step, the process. The section, however, is optional and does not affect the understanding of the rest of the chapter.

Divisor

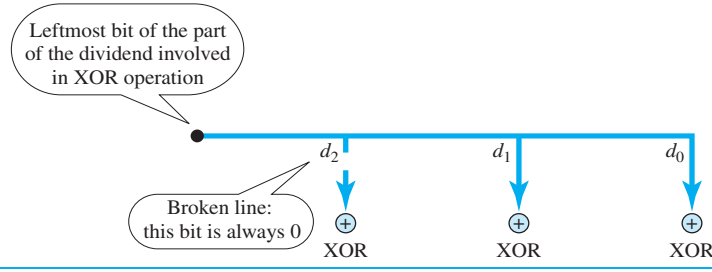
Let us first consider the divisor. We need to note the following points:

1. The divisor is repeatedly XORed with part of the dividend.
2. The divisor has $n - k + 1$ bits which either are predefined or are all 0s. In other words, the bits do not change from one dataword to another. In our previous example, the divisor bits were either 1011 or 0000. The choice was based on the leftmost bit of the part of the augmented data bits that are active in the XOR operation.

3. A close look shows that only $n - k$ bits of the divisor are needed in the XOR operation. The leftmost bit is not needed because the result of the operation is always 0, no matter what the value of this bit. The reason is that the inputs to this XOR operation are either both 0s or both 1s. In our previous example, only 3 bits, not 4, are actually used in the XOR operation.

Using these points, we can make a fixed (hardwired) divisor that can be used for a cyclic code if we know the divisor pattern. Figure 10.11 shows such a design for our previous example. We have also shown the XOR devices used for the operation.

Figure 10.11 *Hardwired design of the divisor in CRC*



Note that if the leftmost bit of the part of the dividend to be used in this step is 1, the divisor bits ($d_2d_1d_0$) are 011; if the leftmost bit is 0, the divisor bits are 000. The design provides the right choice based on the leftmost bit.

Augmented Dataword

In our paper-and-pencil division process in Figure 10.6, we show the augmented dataword as fixed in position with the divisor bits shifting to the right, 1 bit in each step. The divisor bits are aligned with the appropriate part of the augmented dataword. Now that our divisor is fixed, we need instead to shift the bits of the augmented dataword to the left (opposite direction) to align the divisor bits with the appropriate part. There is no need to store the augmented dataword bits.

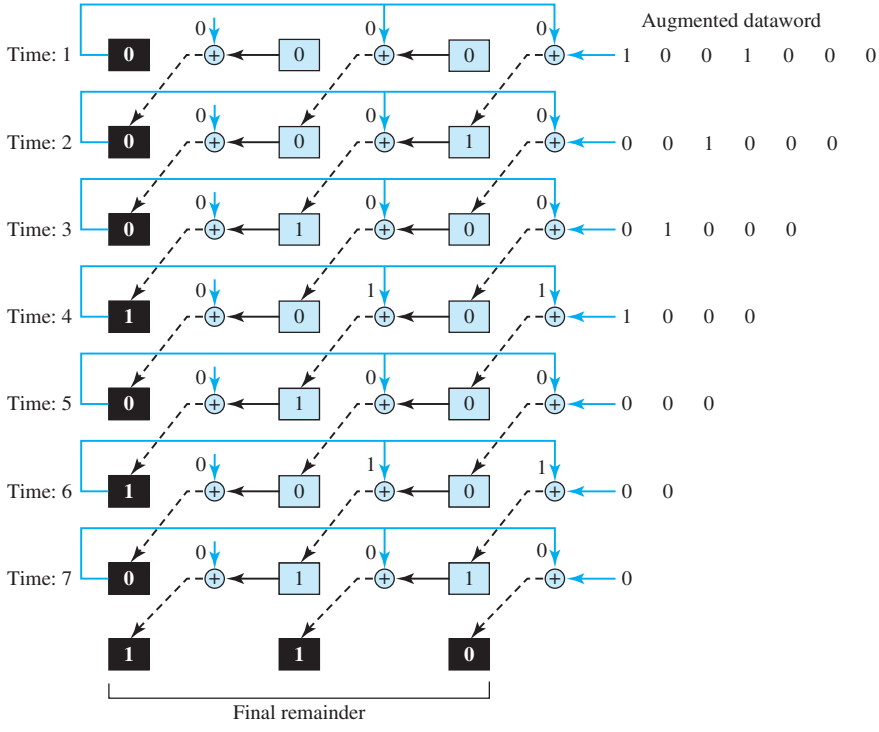
Remainder

In our previous example, the remainder is 3 bits ($n - k$ bits in general) in length. We can use three **registers** (single-bit storage devices) to hold these bits. To find the final remainder of the division, we need to modify our division process. The following is the step-by-step process that can be used to simulate the division process in hardware (or even in software).

1. We assume that the remainder is originally all 0s (000 in our example).
2. At each time click (arrival of 1 bit from an augmented dataword), we repeat the following two actions:
   a. We use the leftmost bit to make a decision about the divisor (011 or 000).
   b. The other 2 bits of the remainder and the next bit from the augmented dataword (total of 3 bits) are XORed with the 3-bit divisor to create the next remainder.

Figure 10.12 shows this simulator, but note that this is not the final design; there will be more improvements.

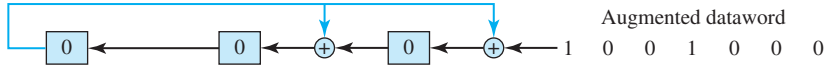
Figure 10.12 *Simulation of division in CRC encoder*



At each clock tick, shown as different times, one of the bits from the augmented dataword is used in the XOR process. If we look carefully at the design, we have seven steps here, while in the paper-and-pencil method we had only four steps. The first three steps have been added here to make each step equal and to make the design for each step the same. Steps 1, 2, and 3 push the first 3 bits to the remainder registers; steps 4, 5, 6, and 7 match the paper-and-pencil design. Note that the values in the remainder register in steps 4 to 7 exactly match the values in the paper-and-pencil design. The final remainder is also the same.

The above design is for demonstration purposes only. It needs simplification to be practical. First, we do not need to keep the intermediate values of the remainder bits; we need only the final bits. We therefore need only 3 registers instead of 24. After the XOR operations, we do not need the bit values of the previous remainder. Also, we do not need 21 XOR devices; two are enough because the output of an XOR operation in which one of the bits is 0 is simply the value of the other bit. This other bit can be used as the output. With these two modifications, the design becomes tremendously simpler and less expensive, as shown in Figure 10.13.

Figure 10.13 *The CRC encoder design using shift registers*

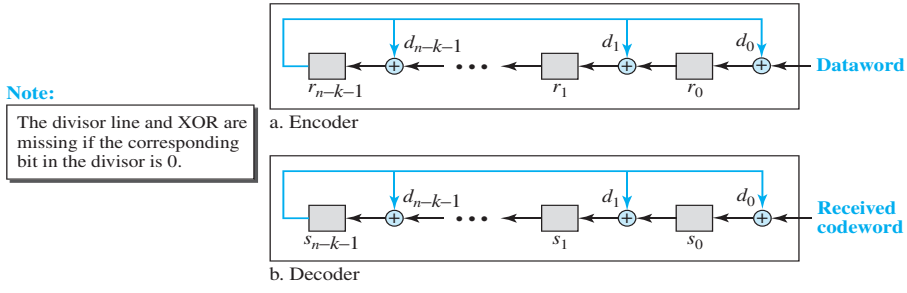


We need, however, to make the registers shift registers. A 1-bit shift register holds a bit for a duration of one clock time. At a time click, the shift register accepts the bit at its input port, stores the new bit, and displays it on the output port. The content and the output remain the same until the next input arrives. When we connect several 1-bit shift registers together, it looks as if the contents of the register are shifting.

General Design

A general design for the encoder and decoder is shown in Figure 10.14.

Figure 10.14 *General design of encoder and decoder of a CRC code*



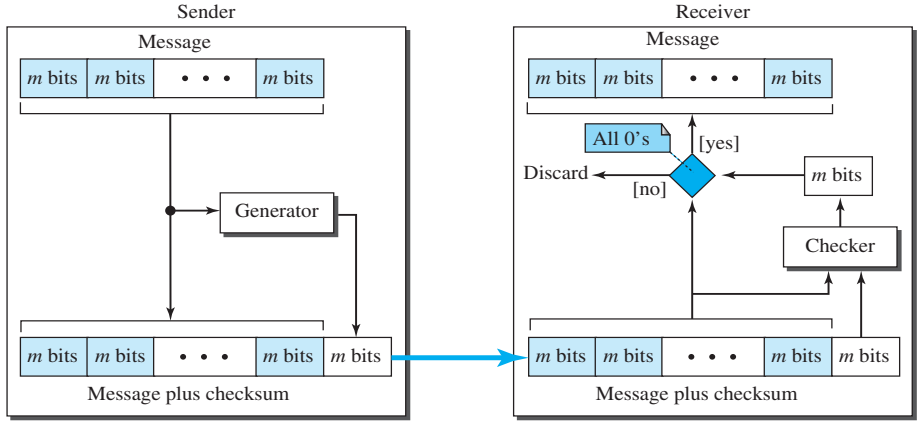
Note that we have $n - k$ 1-bit shift registers in both the encoder and decoder. We have up to $n - k$ XOR devices, but the divisors normally have several 0s in their pattern, which reduces the number of devices. Also note that, instead of augmented datawords, we show the dataword itself as the input because after the bits in the dataword are all fed into the encoder, the extra bits, which all are 0s, do not have any effect on the rightmost XOR. Of course, the process needs to be continued for another $n - k$ steps before the check bits are ready. This fact is one of the criticisms of this design. Better schemes have been designed to eliminate this waiting time (the check bits are ready after k steps), but we leave this as a research topic for the reader. In the decoder, however, the entire codeword must be fed to the decoder before the syndrome is ready.

10.4 CHECKSUM

Checksum is an error-detecting technique that can be applied to a message of any length. In the Internet, the checksum technique is mostly used at the network and transport layer rather than the data-link layer. However, to make our discussion of error-detecting techniques complete, we discuss the checksum in this chapter.

At the source, the message is first divided into *m*-bit units. The generator then creates an extra *m*-bit unit called the ***checksum,*** which is sent with the message. At the destination, the checker creates a new checksum from the combination of the message and sent checksum. If the new checksum is all 0s, the message is accepted; otherwise, the message is discarded (Figure 10.15). Note that in the real implementation, the checksum unit is not necessarily added at the end of the message; it can be inserted in the middle of the message.

Figure 10.15 *Checksum*



10.4.1 Concept

The idea of the traditional checksum is simple. We show this using a simple example.

Example 10.11

Suppose the message is a list of five 4-bit numbers that we want to send to a destination. In addition to sending these numbers, we send the sum of the numbers. For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12, 0, 6, **36**), where 36 is the sum of the original numbers. The receiver adds the five numbers and compares the result with the sum. If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum. Otherwise, there is an error somewhere and the message is not accepted.

One's Complement Addition

The previous example has one major drawback. Each number can be written as a 4-bit word (each is less than 15) except for the sum. One solution is to use **one's complement** arithmetic. In this arithmetic, we can represent unsigned numbers between 0 and $2^m - 1$ using only *m* bits. If the number has more than *m* bits, the extra leftmost bits need to be added to the *m* rightmost bits (wrapping).

Example 10.12

In the previous example, the decimal number 36 in binary is $(100100)_2$. To change it to a 4-bit number we add the extra leftmost bit to the right four bits as shown below.

$$(10)_2 + (0100)_2 = (0110)_2 \rightarrow (6)_{10}$$

Instead of sending 36 as the sum, we can send 6 as the sum (7, 11, 12, 0, 6, **6**). The receiver can add the first five numbers in one's complement arithmetic. If the result is 6, the numbers are accepted; otherwise, they are rejected.

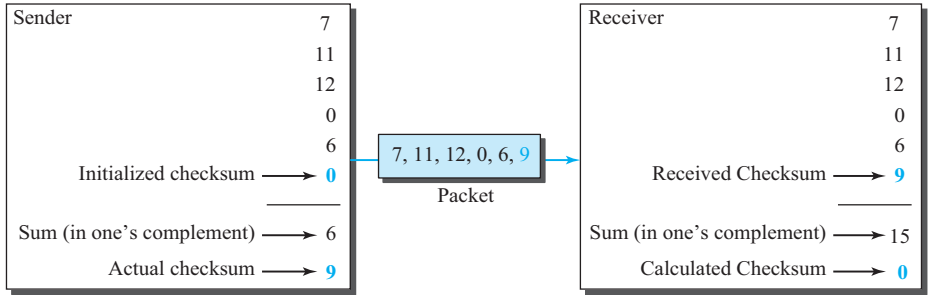
Checksum

We can make the job of the receiver easier if we send the complement of the sum, the checksum. In one's complement arithmetic, the complement of a number is found by completing all bits (changing all 1s to 0s and all 0s to 1s). This is the same as subtracting the number from $2^m - 1$. In one's complement arithmetic, we have two 0s: one positive and one negative, which are complements of each other. The positive zero has all m bits set to 0; the negative zero has all bits set to 1 (it is $2^m - 1$). If we add a number with its complement, we get a negative zero (a number with all bits set to 1). When the receiver adds all five numbers (including the checksum), it gets a negative zero. The receiver can complement the result again to get a positive zero.

Example 10.13

Let us use the idea of the checksum in Example 10.12. The sender adds all five numbers in one's complement to get the sum = 6. The sender then complements the result to get the checksum = 9, which is 15 − 6. Note that $6 = (0110)_2$ and $9 = (1001)_2$; they are complements of each other. The sender sends the five data numbers and the checksum (7, 11, 12, 0, 6, **9**). If there is no corruption in transmission, the receiver receives (7, 11, 12, 0, 6, **9**) and adds them in one's complement to get 15. The sender complements 15 to get 0. This shows that data have not been corrupted. Figure 10.16 shows the process.

Figure 10.16 *Example 10.13*



Internet Checksum

Traditionally, the Internet has used a 16-bit checksum. The sender and the receiver follow the steps depicted in Table 10.5. The sender or the receiver uses five steps.

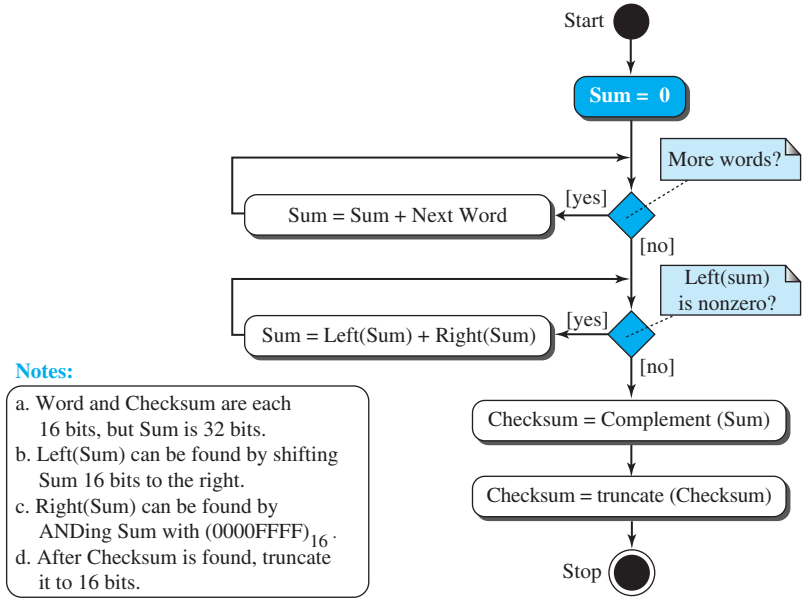
Table 10.5 *Procedure to calculate the traditional checksum*

Sender	*Receiver*
1. The message is divided into 16-bit words.	1. The message and the checksum are received.
2. The value of the checksum word is initially set to zero.	2. The message is divided into 16-bit words.
3. All words including the checksum are added using one's complement addition.	3. All words are added using one's complement addition.
4. The sum is complemented and becomes the checksum.	4. The sum is complemented and becomes the new checksum.
5. The checksum is sent with the data.	5. If the value of the checksum is 0, the message is accepted; otherwise, it is rejected.

Algorithm

We can use the flow diagram of Figure 10.17 to show the algorithm for calculation of the checksum. A program in any language can easily be written based on the algorithm. Note that the first loop just calculates the sum of the data units in two's complement; the second loop wraps the extra bits created from the two's complement calculation to simulate the calculations in one's complement. This is needed because almost all computers today do calculation in two's complement.

Figure 10.17 *Algorithm to calculate a traditional checksum*



Performance

The traditional checksum uses a small number of bits (16) to detect errors in a message of any size (sometimes thousands of bits). However, it is not as strong as the CRC in error-checking capability. For example, if the value of one word is incremented and the value of another word is decremented by the same amount, the two errors cannot be detected because the sum and checksum remain the same. Also, if the values of several words are incremented but the sum and the checksum do not change, the errors are not detected. Fletcher and Adler have proposed some weighted checksums that eliminate the first problem. However, the tendency in the Internet, particularly in designing new protocols, is to replace the checksum with a CRC.

10.4.2 Other Approaches to the Checksum

As mentioned before, there is one major problem with the traditional checksum calculation. If two 16-bit items are transposed in transmission, the checksum cannot catch this error. The reason is that the traditional checksum is not weighted: it treats each data item equally. In other words, the order of data items is immaterial to the calculation. Several approaches have been used to prevent this problem. We mention two of them here: Fletcher and Adler.

Fletcher Checksum

The Fletcher checksum was devised to weight each data item according to its position. Fletcher has proposed two algorithms: 8-bit and 16-bit. The first, 8-bit Fletcher, calculates on 8-bit data items and creates a 16-bit checksum. The second, 16-bit Fletcher, calculates on 16-bit data items and creates a 32-bit checksum.

The 8-bit Fletcher is calculated over data octets (bytes) and creates a 16-bit checksum. The calculation is done modulo 256 (2^8), which means the intermediate results are divided by 256 and the remainder is kept. The algorithm uses two accumulators, L and R. The first simply adds data items together; the second adds a weight to the calculation. There are many variations of the 8-bit Fletcher algorithm; we show a simple one in Figure 10.18.

The 16-bit Fletcher checksum is similar to the 8-bit Fletcher checksum, but it is calculated over 16-bit data items and creates a 32-bit checksum. The calculation is done modulo 65,536.

Adler Checksum

The Adler checksum is a 32-bit checksum. Figure 10.19 shows a simple algorithm in flowchart form. It is similar to the 16-bit Fletcher with three differences. First, calculation is done on single bytes instead of 2 bytes at a time. Second, the modulus is a prime number (65,521) instead of 65,536. Third, L is initialized to 1 instead of 0. It has been proved that a prime modulo has a better detecting capability in some combinations of data.

Figure 10.18 *Algorithm to calculate an 8-bit Fletcher checksum*

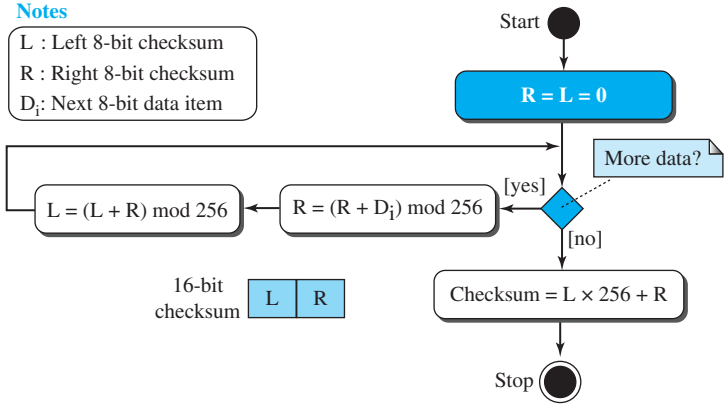
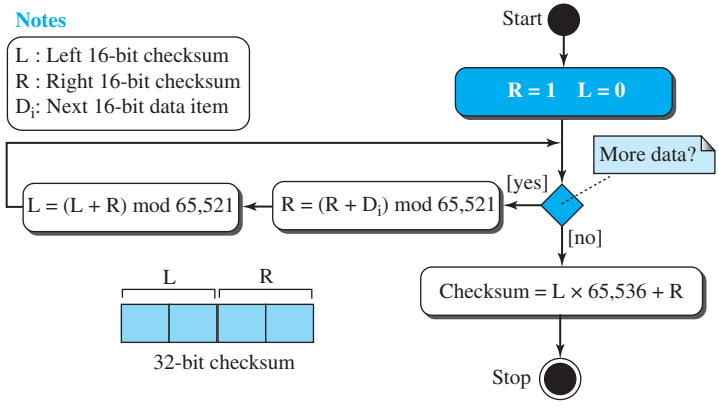


Figure 10.19 *Algorithm to calculate an Adler checksum*



To see the behavior of the different checksum algorithms, check some of the applets for this chapter at the book website.

10.5 FORWARD ERROR CORRECTION

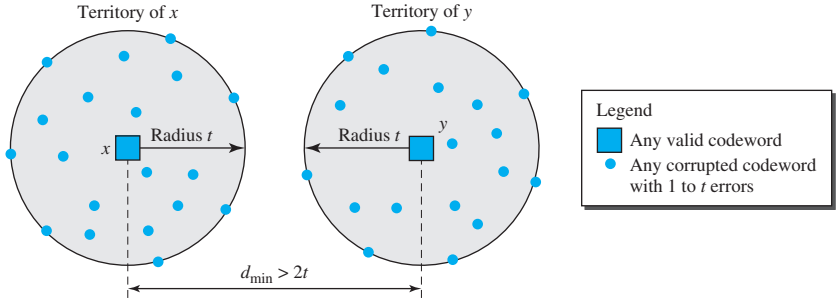
We discussed error detection and retransmission in the previous sections. However, retransmission of corrupted and lost packets is not useful for real-time multimedia transmission because it creates an unacceptable delay in reproducing: we need to wait until the lost or corrupted packet is resent. We need to correct the error or reproduce the

packet immediately. Several schemes have been designed and used in this case that are collectively referred to as **forward error correction** (**FEC**) techniques. We briefly discuss some of the common techniques here.

10.5.1 Using Hamming Distance

We earlier discussed the Hamming distance for error detection. We said that to detect s errors, the minimum Hamming distance should be $d_{min} = s + 1$. For error detection, we definitely need more distance. It can be shown that to detect t errors, we need to have $d_{min} = 2t + 1$. In other words, if we want to correct 10 bits in a packet, we need to make the minimum hamming distance 21 bits, which means a lot of redundant bits need to be sent with the data. To give an example, consider the famous BCH code. In this code, if data is 99 bits, we need to send 255 bits (extra 156 bits) to correct just 23 possible bit errors. Most of the time we cannot afford such a redundancy. We give some examples of how to calculate the required bits in the practice set. Figure 10.20 shows the geometrical representation of this concept.

Figure 10.20 *Hamming distance for error correction*



10.5.2 Using XOR

Another recommendation is to use the property of the exclusive OR operation as shown below.

$$\mathbf{R} = \mathbf{P_1} \oplus \mathbf{P_2} \oplus \ldots \oplus \mathbf{P}_i \oplus \ldots \oplus \mathbf{P}_N \quad \rightarrow \quad \mathbf{P}_i = \mathbf{P_1} \oplus \mathbf{P_2} \oplus \ldots \oplus \mathbf{R} \oplus \ldots \oplus \mathbf{P}_N$$

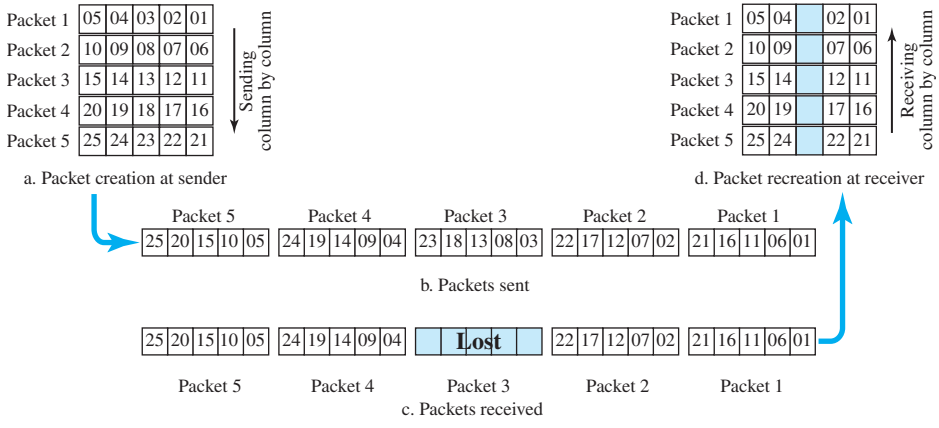
In other words, if we apply the exclusive OR operation on N data items (P_1 to P_N), we can recreate any of the data items by exclusive-ORing all of the items, replacing the one to be created by the result of the previous operation (R). This means that we can divide a packet into N chunks, create the exclusive OR of all the chunks and send $N + 1$ chunks. If any chunk is lost or corrupted, it can be created at the receiver site. Now the question is what should the value of N be. If $N = 4$, it means that we need to send 25 percent extra data and be able to correct the data if only one out of four chunks is lost.

10.5.3 Chunk Interleaving

Another way to achieve FEC in multimedia is to allow some small chunks to be missing at the receiver. We cannot afford to let all the chunks belonging to the same

packet be missing; however, we can afford to let one chunk be missing in each packet. Figure 10.21 shows that we can divide each packet into 5 chunks (normally the number is much larger). We can then create data chunk by chunk (horizontally), but combine the chunks into packets vertically. In this case, each packet sent carries a chunk from several original packets. If the packet is lost, we miss only one chunk in each packet, which is normally acceptable in multimedia communication.

Figure 10.21 *Interleaving*



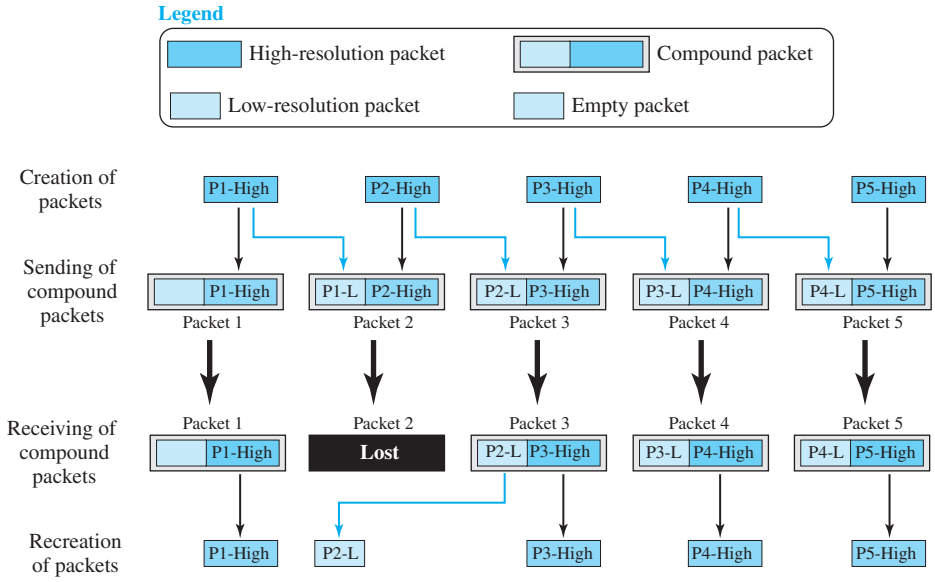
10.5.4 Combining Hamming Distance and Interleaving

Hamming distance and interleaving can be combined. We can first create n-bit packets that can correct t-bit errors. Then we interleave m rows and send the bits column by column. In this way, we can automatically correct burst errors up to $m \times t$-bit errors.

10.5.5 Compounding High- and Low-Resolution Packets

Still another solution is to create a duplicate of each packet with a low-resolution redundancy and combine the redundant version with the next packet. For example, we can create four low-resolution packets out of five high-resolution packets and send them as shown in Figure 10.22. If a packet is lost, we can use the low-resolution version from the next packet. Note that the low-resolution section in the first packet is empty. In this method, if the last packet is lost, it cannot be recovered, but we use the low-resolution version of a packet if the lost packet is not the last one. The audio and video reproduction does not have the same quality, but the lack of quality is not recognized most of the time.

Figure 10.22 *Compounding high- and low-resolution packets*



10.6 END-CHAPTER MATERIALS

10.6.1 Recommended Reading

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

Books

Several excellent books discuss link-layer issues. Among them we recommend [Ham 80], [Zar 02], [Ror 96], [Tan 03], [GW 04], [For 03], [KMK 04], [Sta 04], [Kes 02], [PD 03], [Kei 02], [Spu 00], [KCK 98], [Sau 98], [Izz 00], [Per 00], and [WV 00].

RFCs

A discussion of the use of the checksum in the Internet can be found in RFC 1141.

10.6.2 Key Terms

block coding
burst error
check bit
checksum
codeword
convolution coding
cyclic code
cyclic redundancy check (CRC)
dataword
error
error correction
error detection
forward error correction (FEC)
generator polynomial
Hamming distance
interference
linear block code
minimum Hamming distance
modular arithmetic
one's complement
parity-check code
polynomial
redundancy
register
Reed-Solomon code
single-bit error
syndrome

10.6.3 Summary

Data can be corrupted during transmission. Some applications require that errors be detected and corrected. In a single-bit error, only one bit in the data unit has changed. A burst error means that two or more bits in the data unit have changed. To detect or correct errors, we need to send extra (redundant) bits with data. There are two main methods of error correction: forward error correction and correction by retransmission.

We can divide coding schemes into two broad categories: block coding and convolution coding. In coding, we need to use modulo-2 arithmetic. Operations in this arithmetic are very simple; addition and subtraction give the same results. We use the XOR (exclusive OR) operation for both addition and subtraction. In block coding, we divide our message into blocks, each of k bits, called *datawords*. We add r redundant bits to each block to make the length $n = k + r$. The resulting n-bit blocks are called *codewords*.

In block coding, errors be detected by using the following two conditions:

a. The receiver has (or can find) a list of valid codewords.

b. The original codeword has changed to an invalid one.

The Hamming distance between two words is the number of differences between corresponding bits. The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of words. To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = s + 1$. To guarantee correction of up to t errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = 2t + 1$.

In a linear block code, the exclusive OR (XOR) of any two valid codewords creates another valid codeword.

A simple parity-check code is a single-bit error-detecting code in which $n = k + 1$ with $d_{min} = 2$. A simple parity-check code can detect an odd number of errors.

All Hamming codes discussed in this book have $d_{min} = 3$. The relationship between m and n in these codes is $n = 2m - 1$.

Cyclic codes are special linear block codes with one extra property. In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword. A category

of cyclic codes called the cyclic redundancy check (CRC) is used in networks such as LANs and WANs.

A pattern of 0s and 1s can be represented as a polynomial with coefficients of 0 and 1. Traditionally, the Internet has been using a 16-bit checksum, which uses one's complement arithmetic. In this arithmetic, we can represent unsigned numbers between 0 and $2^n - 1$ using only n bits.

10.7 PRACTICE SET

10.7.1 Quizzes

A set of interactive quizzes for this chapter can be found on the book website. It is strongly recommended that the student take the quizzes to check his/her understanding of the materials before continuing with the practice set.

10.7.2 Questions

Q10-1. How does a single-bit error differ from a burst error?

Q10-2. What is the definition of a linear block code?

Q10-3. In a block code, a dataword is 20 bits and the corresponding codeword is 25 bits. What are the values of k, r, and n according to the definitions in the text? How many redundant bits are added to each dataword?

Q10-4. In a codeword, we add two redundant bits to each 8-bit data word. Find the number of

a. valid codewords. **b.** invalid codewords.

Q10-5. What is the minimum Hamming distance?

Q10-6. If we want to be able to detect two-bit errors, what should be the minimum Hamming distance?

Q10-7. A category of error detecting (and correcting) code, called the ***Hamming code,*** is a code in which $\boldsymbol{d}_{\text{min}} = 3$. This code can detect up to two errors (or correct one single error). In this code, the values of n, k, and r are related as: $n = 2^r - 1$ and $k = n - r$. Find the number of bits in the dataword and the codewords if r is 3.

Q10-8. In CRC, if the dataword is 5 bits and the codeword is 8 bits, how many 0s need to be added to the dataword to make the dividend? What is the size of the remainder? What is the size of the divisor?

Q10-9. In CRC, which of the following generators (divisors) guarantees the detection of a single bit error?

a. 101 **b.** 100 **c.** 1

Q10-10. In CRC, which of the following generators (divisors) guarantees the detection of an odd number of errors?

a. 10111 **b.** 101101 **c.** 111

Q10-11. In CRC, we have chosen the generator 1100101. What is the probability of detecting a burst error of length

a. 5? **b.** 7? **c.** 10?

Q10-12. Assume we are sending data items of 16-bit length. If two data items are swapped during transmission, can the traditional checksum detect this error? Explain.

Q10-13. Can the value of a traditional checksum be all 0s (in binary)? Defend your answer.

Q10-14. Show how the Fletcher algorithm (Figure 10.18) attaches weights to the data items when calculating the checksum.

Q10-15. Show how the Adler algorithm (Figure 10.19) attaches weights to the data items when calculating the checksum.

10.7.3 Problems

P10-1. What is the maximum effect of a 2-ms burst of noise on data transmitted at the following rates?

a. 1500 bps **b.** 12 kbps **c.** 100 kbps **d.** 100 Mbps

P10-2. Assume that the probability that a bit in a data unit is corrupted during transmission is p. Find the probability that x number of bits are corrupted in an n-bit data unit for each of the following cases.

a. $n = 8, x = 1, p = 0.2$

b. $n = 16, x = 3, p = 0.3$

c. $n = 32, x = 10, p = 0.4$

P10-3. Exclusive-OR (XOR) is one of the most used operations in the calculation of codewords. Apply the exclusive-OR operation on the following pairs of patterns. Interpret the results.

a. (10001) $\oplus$ (10001) **b.** (11100) $\oplus$ (00000) **c.** (10011) $\oplus$ (11111)

P10-4. In Table 10.1, the sender sends dataword 10. A 3-bit burst error corrupts the codeword. Can the receiver detect the error? Defend your answer.

P10-5. Using the code in Table 10.2, what is the dataword if each of the following codewords is received?

a. 01011 **b.** 11111 **c.** 00000 **d.** 11011

P10-6. Prove that the code represented by the following codewords is not linear. You need to find only one case that violates the linearity.

{(00000), (01011), (10111), (11111)}

P10-7. What is the Hamming distance for each of the following codewords?

a. d (10000, 00000) **b.** d (10101, 10000)
c. d (00000, 11111) **d.** d (00000, 00000)

P10-8. Although it can be formally proved that the code in Table 10.3 is both linear and cyclic, use only two tests to partially prove the fact:

a. Test the cyclic property on codeword 0101100.

b. Test the linear property on codewords 0010110 and 1111111.

P10-9. Referring to the CRC-8 in Table 5.4, answer the following questions:

a. Does it detect a single error? Defend your answer.

b. Does it detect a burst error of size 6? Defend your answer.

c. What is the probability of detecting a burst error of size 9?

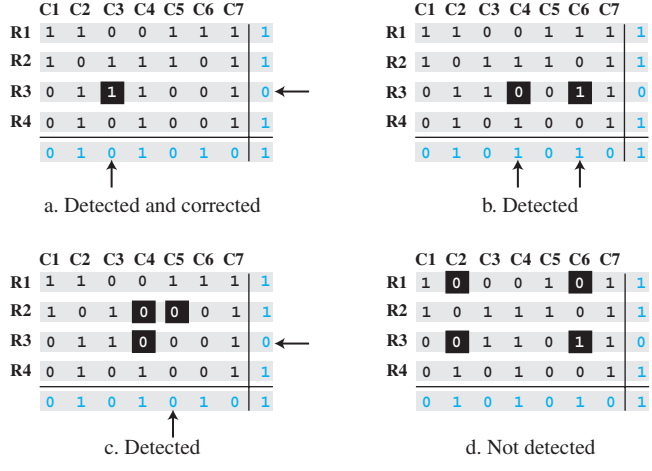
d. What is the probability of detecting a burst error of size 15?

P10-10. Assuming even parity, find the parity bit for each of the following data units.

a. 1001011 **b.** 0001100 **c.** 1000000 **d.** 1110111

P10-11. A simple parity-check bit, which is normally added at the end of the word (changing a 7-bit ASCII character to a byte), cannot detect even numbers of errors. For example, two, four, six, or eight errors cannot be detected in this way. A better solution is to organize the characters in a table and create row and column parities. The bit in the row parity is sent with the byte, the column parity is sent as an extra byte (Figure 10.23).

Figure 10.23 *P10-11*



Show how the following errors can be detected:

a. An error at (R3, C3).

b. Two errors at (R3, C4) and (R3, C6).

c. Three errors at (R2, C4), (R2, C5), and (R3, C4).

d. Four errors at (R1, C2), (R1, C6), (R3, C2), and (R3, C6).

P10-12. Given the dataword 101001111 and the divisor 10111, show the generation of the CRC codeword at the sender site (using binary division).

P10-13. Apply the following operations on the corresponding polynomials:

a. $(x^3 + x^2 + x + 1) + (x^4 + x^2 + x + 1)$

b. $(x^3 + x^2 + x + 1) - (x^4 + x^2 + x + 1)$

c. $(x^3 + x^2) \times (x^4 + x^2 + x + 1)$

d. $(x^3 + x^2 + x + 1) / (x^2 + 1)$

P10-14. Answer the following questions:

a. What is the polynomial representation of 101110?

b. What is the result of shifting 101110 three bits to the left?

c. Repeat part b using polynomials.

d. What is the result of shifting 101110 four bits to the right?

e. Repeat part d using polynomials.

P10-15. Which of the following CRC generators guarantee the detection of a single bit error?

a. $x^3 + x + 1$ **b.** $x^4 + x^2$ **c.** 1 **d.** $x^2 + 1$

P10-16. Referring to the CRC-8 polynomial in Table 10.7, answer the following questions:

a. Does it detect a single error? Defend your answer.

b. Does it detect a burst error of size 6? Defend your answer.

c. What is the probability of detecting a burst error of size 9?

d. What is the probability of detecting a burst error of size 15?

P10-17. Referring to the CRC-32 polynomial in Table 10.4, answer the following questions:

a. Does it detect a single error? Defend your answer.

b. Does it detect a burst error of size 16? Defend your answer.

c. What is the probability of detecting a burst error of size 33?

d. What is the probability of detecting a burst error of size 55?

P10-18. Assume a packet is made only of four 16-bit words $(A7A2)_{16}$, $(CABF)_{16}$, $(903A)_{16}$, and $(A123)_{16}$. Manually simulate the algorithm in Figure 10.17 to find the checksum.

P10-19. Traditional checksum calculation needs to be done in one's complement arithmetic. Computers and calculators today are designed to do calculations in two's complement arithmetic. One way to calculate the traditional checksum is to add the numbers in two's complement arithmetic, find the quotient and remainder of dividing the result by 2^{16}, and add the quotient and the remainder to get the sum in one's complement. The checksum can be found by subtracting the sum from $2^{16} - 1$. Use the above method to find the checksum of the following four numbers: 43,689, 64,463, 45,112, and 59,683.

P10-20. This problem shows a special case in checksum handling. A sender has two data items to send: $(4567)_{16}$ and $(BA98)_{16}$. What is the value of the checksum?

P10-21. Manually simulate the Fletcher algorithm (Figure 10.18) to calculate the checksum of the following bytes: $(2B)_{16}$, $(3F)_{16}$, $(6A)_{16}$, and $(AF)_{16}$. Also show that the result is a weighted checksum.

P10-22. Manually simulate the Adler algorithm (Figure 10.19) to calculate the checksum of the following words: $(FBFF)_{16}$ and $(EFAA)_{16}$. Also show that the result is a weighted checksum.

P10-23. One of the examples of a weighted checksum is the ISBN-10 code we see printed on the back cover of some books. In ISBN-10, there are 9 decimal digits that define the country, the publisher, and the book. The tenth (rightmost) digit is a checksum digit. The code, $D_1D_2D_3D_4D_5D_6D_7D_8D_9\mathbf{C}$, satisfies the following.

$$[(10 \times D_1) + (9 \times D_2) + (8 \times D_3) + \ldots + (2 \times D_9) + (1 \times \mathbf{C})] \bmod 11 = 0$$

In other words, the weights are 10, 9, . . .,1. If the calculated value for C is 10, one uses the letter X instead. By replacing each weight w with its complement in modulo 11 arithmetic $(11 - w)$, it can be shown that the check digit can be calculated as shown below.

$$\mathbf{C} = [(1 \times D_1) + (2 \times D_2) + (3 \times D_3) + \ldots + (9 \times D_9)] \bmod 11$$

Calculate the check digit for ISBN-10: **0-07-296775-C**.

P10-24. An ISBN-13 code, a new version of ISBN-10, is another example of a weighted checksum with 13 digits, in which there are 12 decimal digits defining the book and the last digit is the checksum digit. The code, $D_1D_2D_3D_4D_5D_6D_7D_8D_9D_{10}D_{11}D_{12}\mathbf{C}$, satisfies the following.

$$[(1 \times D_1) + (3 \times D_2) + (1 \times D_3) + \ldots + (3 \times D_{12}) + (1 \times \mathbf{C})] \bmod 10 = 0$$

In other words, the weights are 1 and 3 alternately. Using the above description, calculate the check digit for ISBN-13: **978-0-07-296775-C**.

P10-25. In the interleaving approach to FEC, assume each packet contains 10 samples from a sampled piece of music. Instead of loading the first packet with the first 10 samples, the second packet with the second 10 samples, and so on, the sender loads the first packet with the odd-numbered samples of the first 20 samples, the second packet with the even-numbered samples of the first 20 samples, and so on. The receiver reorders the samples and plays them. Now assume that the third packet is lost in transmission. What will be missed at the receiver site?

P10-26. Assume we want to send a dataword of two bits using FEC based on the Hamming distance. Show how the following list of datawords/codewords can automatically correct up to a one-bit error in transmission.

$$00 \rightarrow 00000 \qquad 01 \rightarrow 01011 \qquad 10 \rightarrow 10101 \qquad 11 \rightarrow 11110$$

P10-27. Assume we need to create codewords that can automatically correct a one-bit error. What should the number of redundant bits (r) be, given the number of bits in the dataword (k)? Remember that the codeword needs to be $n = k + r$ bits, called C(n, k). After finding the relationship, find the number of bits in r if k is 1, 2, 5, 50, or 1000.

P10-28. In the previous problem we tried to find the number of bits to be added to a dataword to correct a single-bit error. If we need to correct more than one bit, the number of redundant bits increases. What should the number of redundant bits (r) be to automatically correct one or two bits (not necessarily contiguous) in a dataword of size k? After finding the relationship, find the number of bits in r if k is 1, 2, 5, 50, or 1000.

P10-29. Using the ideas in the previous two problems, we can create a general formula for correcting any number of errors (m) in a codeword of size (n). Develop such a formula. Use the combination of n objects taking x objects at a time.

P10-30. In Figure 10.22, assume we have 100 packets. We have created two sets of packets with high and low resolutions. Each high-resolution packet carries on average 700 bits. Each low-resolution packet carries on average 400 bits. How many extra bits are we sending in this scheme for the sake of FEC? What is the percentage of overhead?

10.8 SIMULATION EXPERIMENTS

10.8.1 Applets

We have created some Java applets to show some of the main concepts discussed in this chapter. It is strongly recommended that the students activate these applets on the book website and carefully examine the protocols in action.

10.9 PROGRAMMING ASSIGNMENTS

For each of the following assignments, write a program in the programming language you are familiar with.

Prg10-1. A program to simulate the calculation of CRC.

Prg10-2. A program to simulate the calculation of traditional checksum.

Prg10-3. A program to simulate the calculation of Fletcher checksum.

Prg10-4. A program to simulate the calculation of Adler checksum.

CHAPTER 11

Data Link Control (DLC)

As we discussed in Chapter 9, the data-link layer is divided into two sublayers. In this chapter, we discuss the upper sublayer of the data-link layer (DLC). The lower sublayer, multiple access control (MAC) will be discussed in Chapter 12. We have already discussed error detection and correction, an issue that is encountered in several layers, in Chapter 10.

This chapter is divided into four sections.

- The first section discusses the general services provided by the DLC sublayer. It first describes framing and two types of frames used in this sublayer. The section then discusses flow and error control. Finally, the section explains that a DLC protocol can be either connectionless or connection-oriented.
- The second section discusses some simple and common data-link protocols that are implemented at the DLC sublayer. The section first describes the Simple Protocol. It then explains the Stop-and-Wait Protocol.
- The third section introduces HDLC, a protocol that is the basis of all common data-link protocols in use today such as PPP and Ethernet. The section first talks about configurations and transfer modes. It then describes framing and three different frame formats used in this protocol.
- The fourth section discusses PPP, a very common protocol for point-to-point access. It first introduces the services provided by the protocol. The section also describes the format of the frame in this protocol. It then describes the transition mode in the protocol using an FSM. The section finally explains multiplexing in PPP.

11.1 DLC SERVICES

The **data link control (DLC)** deals with procedures for communication between two adjacent nodes—node-to-node communication—no matter whether the link is dedicated or broadcast. Data link control functions include *framing* and *flow and error control*. In this section, we first discuss framing, or how to organize the bits that are carried by the physical layer. We then discuss flow and error control.

11.1.1 Framing

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

The data-link layer, on the other hand, needs to pack bits into frames, so that each frame is distinguishable from another. Our postal system practices a type of framing. The simple act of inserting a letter into an envelope separates one piece of information from another; the envelope serves as the delimiter. In addition, each envelope defines the sender and receiver addresses, which is necessary since the postal system is a many-to-many carrier facility.

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

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

Frame Size

Frames can be of fixed or variable size. In *fixed-size framing,* there is no need for defining the boundaries of the frames; the size itself can be used as a delimiter. An example of this type of framing is the ATM WAN, which uses frames of fixed size called *cells*. We discuss ATM in Chapter 14.

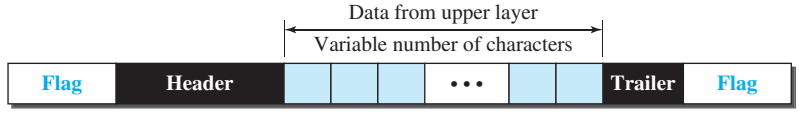
Our main discussion in this chapter concerns *variable-size framing,* prevalent in local-area networks. In variable-size framing, we need a way to define the end of one frame and the beginning of the next. Historically, two approaches were used for this purpose: a character-oriented approach and a bit-oriented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

Character-Oriented Framing

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

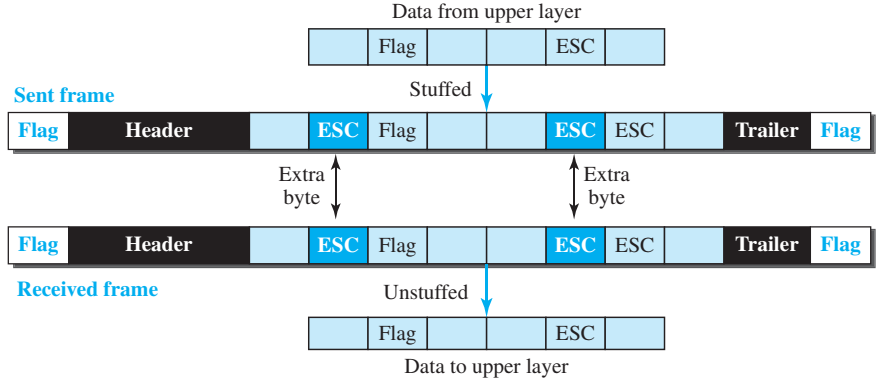
start or end of a frame. Figure 11.1 shows the format of a frame in a character-oriented protocol.

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



Character-oriented framing was popular when only text was exchanged by the data-link layers. The flag could be selected to be any character not used for text communication. Now, however, we send other types of information such as graphs, audio, and video; any character used for the flag could also be part of the information. If this happens, the receiver, when it encounters this pattern in the middle of the data, thinks it has reached the end of the frame. To fix this problem, a byte-stuffing strategy was added to character-oriented framing. In **byte stuffing** (or character stuffing), a special byte is added to the data section of the frame when there is a character with the same pattern as the flag. The data section is stuffed with an extra byte. This byte is usually called the *escape character (ESC)* and has a predefined bit pattern. Whenever the receiver encounters the ESC character, it removes it from the data section and treats the next character as data, not as a delimiting flag. Figure 11.2 shows the situation.

Figure 11.2 *Byte stuffing and unstuffing*



Byte stuffing is the process of adding one extra byte whenever there is a flag or escape character in the text.

Byte stuffing by the escape character allows the presence of the flag in the data section of the frame, but it creates another problem. What happens if the text contains one or more escape characters followed by a byte with the same pattern as the flag? The

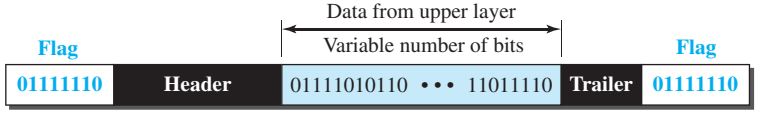
receiver removes the escape character, but keeps the next byte, which is incorrectly interpreted as the end of the frame. To solve this problem, the escape characters that are part of the text must also be marked by another escape character. In other words, if the escape character is part of the text, an extra one is added to show that the second one is part of the text.

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

Bit-Oriented Framing

In *bit-oriented framing,* the data section of a frame is a sequence of bits to be interpreted by the upper layer as text, graphic, audio, video, and so on. However, in addition to headers (and possible trailers), we still need a delimiter to separate one frame from the other. Most protocols use a special 8-bit pattern flag, 01111110, as the delimiter to define the beginning and the end of the frame, as shown in Figure 11.3.

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



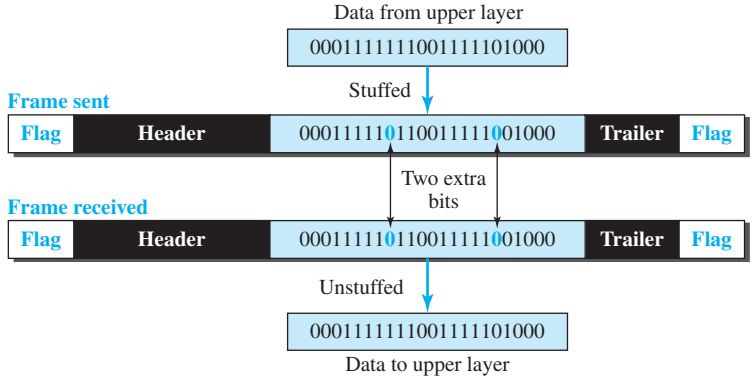
This flag can create the same type of problem we saw in the character-oriented protocols. That is, if the flag pattern appears in the data, we need to somehow inform the receiver that this is not the end of the frame. We do this by stuffing 1 single bit (instead of 1 byte) to prevent the pattern from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

Bit stuffing is the process of adding one extra 0 whenever five consecutive 1s follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag.

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

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

Figure 11.4 *Bit stuffing and unstuffing*



11.1.2 Flow and Error Control

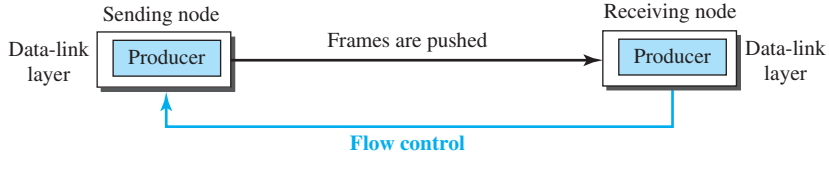
We briefly defined flow and error control in Chapter 9; we elaborate on these two issues here. One of the responsibilities of the data-link control sublayer is flow and error control at the data-link layer.

Flow Control

Whenever an entity produces items and another entity consumes them, there should be a balance between production and consumption rates. If the items are produced faster than they can be consumed, the consumer can be overwhelmed and may need to discard some items. If the items are produced more slowly than they can be consumed, the consumer must wait, and the system becomes less efficient. Flow control is related to the first issue. We need to prevent losing the data items at the consumer site.

In communication at the data-link layer, we are dealing with four entities: network and data-link layers at the sending node and network and data-link layers at the receiving node. Although we can have a complex relationship with more than one producer and consumer (as we will see in Chapter 23), we ignore the relationships between networks and data-link layers and concentrate on the relationship between two data-link layers, as shown in Figure 11.5.

Figure 11.5 *Flow control at the data-link layer*



The figure shows that the data-link layer at the sending node tries to push frames toward the data-link layer at the receiving node. If the receiving node cannot process and deliver the packet to its network at the same rate that the frames arrive, it becomes overwhelmed with frames. Flow control in this case can be feedback from the receiving node to the sending node to stop or slow down pushing frames.

Buffers

Although flow control can be implemented in several ways, one of the solutions is normally to use two *buffers*; one at the sending data-link layer and the other at the receiving data-link layer. A buffer is a set of memory locations that can hold packets at the sender and receiver. The flow control communication can occur by sending signals from the consumer to the producer. When the buffer of the receiving data-link layer is full, it informs the sending data-link layer to stop pushing frames.

Example 11.1

The above discussion requires that the consumers communicate with the producers on two occasions: when the buffer is full and when there are vacancies. If the two parties use a buffer with only one slot, the communication can be easier. Assume that each data-link layer uses one single memory slot to hold a frame. When this single slot in the receiving data-link layer is empty, it sends a note to the network layer to send the next frame.

Error Control

Since the underlying technology at the physical layer is not fully reliable, we need to implement error control at the data-link layer to prevent the receiving node from delivering corrupted packets to its network layer. Error control at the data-link layer is normally very simple and implemented using one of the following two methods. In both methods, a CRC is added to the frame header by the sender and checked by the receiver.

- In the first method, if the frame is corrupted, it is silently discarded; if it is not corrupted, the packet is delivered to the network layer. This method is used mostly in wired LANs such as Ethernet.
- In the second method, if the frame is corrupted, it is silently discarded; if it is not corrupted, an acknowledgment is sent (for the purpose of both flow and error control) to the sender.

Combination of Flow and Error Control

Flow and error control can be combined. In a simple situation, the acknowledgment that is sent for flow control can also be used for error control to tell the sender the packet has arrived uncorrupted. The lack of acknowledgment means that there is a problem in the sent frame. We show this situation when we discuss some simple protocols in the next section. A frame that carries an acknowledgment is normally called an *ACK* to distinguish it from the data frame.

11.1.3 Connectionless and Connection-Oriented

A DLC protocol can be either connectionless or connection-oriented. We will discuss this issue very briefly here, but we return to this topic in the network and transport layer.

Connectionless Protocol

In a connectionless protocol, frames are sent from one node to the next without any relationship between the frames; each frame is independent. Note that the term *connectionless* here does not mean that there is no physical connection (transmission medium) between the nodes; it means that there is no *connection* between frames. The frames are not numbered and there is no sense of ordering. Most of the data-link protocols for LANs are connectionless protocols.

Connection-Oriented Protocol

In a connection-oriented protocol, a logical connection should first be established between the two nodes (setup phase). After all frames that are somehow related to each other are transmitted (transfer phase), the logical connection is terminated (teardown phase). In this type of communication, the frames are numbered and sent in order. If they are not received in order, the receiver needs to wait until all frames belonging to the same set are received and then deliver them in order to the network layer. Connection-oriented protocols are rare in wired LANs, but we can see them in some point-to-point protocols, some wireless LANs, and some WANs.

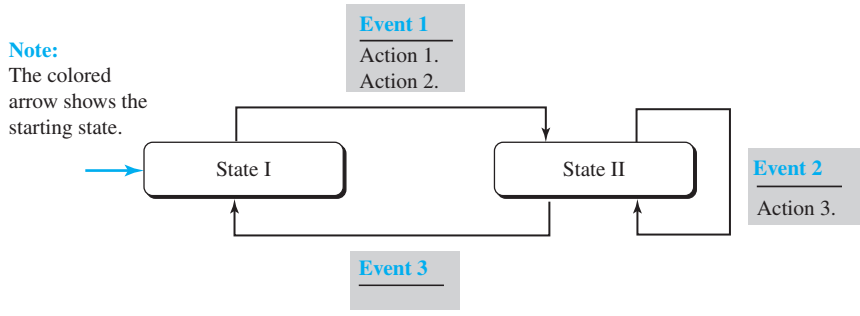
11.2 DATA-LINK LAYER PROTOCOLS

Traditionally four protocols have been defined for the data-link layer to deal with flow and error control: Simple, Stop-and-Wait, Go-Back-N, and Selective-Repeat. Although the first two protocols still are used at the data-link layer, the last two have disappeared. We therefore briefly discuss the first two protocols in this chapter, in which we need to understand some wired and wireless LANs. We postpone the discussion of all four, in full detail, to Chapter 23, where we discuss the transport layer.

The behavior of a data-link-layer protocol can be better shown as a **finite state machine (FSM).** An FSM is thought of as a machine with a finite number of states. The machine is always in one of the states until an *event* occurs. Each event is associated with two reactions: defining the list (possibly empty) of actions to be performed and determining the next state (which can be the same as the current state). One of the states must be defined as the initial state, the state in which the machine starts when it turns on. In Figure 11.6, we show an example of a machine using FSM. We have used rounded-corner rectangles to show states, colored text to show events, and regular black text to show actions. A horizontal line is used to separate the event from the actions, although later we replace the horizontal line with a slash. The arrow shows the movement to the next state.

The figure shows a machine with three states. There are only three possible events and three possible actions. The machine starts in state I. If event 1 occurs, the machine performs actions 1 and 2 and moves to state II. When the machine is in state II, two events may occur. If event 1 occurs, the machine performs action 3 and remains in the same state, state II. If event 3 occurs, the machine performs no action, but move to state I.

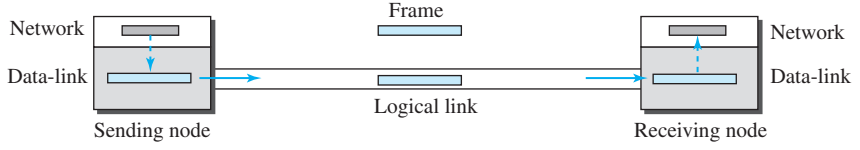
Figure 11.6 *Connectionless and connection-oriented service represented as FSMs*



11.2.1 Simple Protocol

Our first protocol is a **simple protocol** with neither flow nor error control. We assume that the receiver can immediately handle any frame it receives. In other words, the receiver can never be overwhelmed with incoming frames. Figure 11.7 shows the layout for this protocol.

Figure 11.7 *Simple protocol*

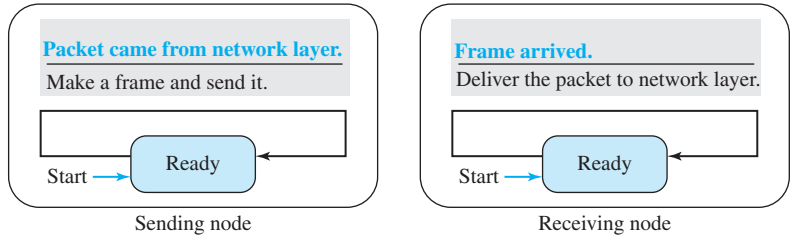


The data-link layer at the sender gets a packet from its network layer, makes a frame out of it, and sends the frame. The data-link layer at the receiver receives a frame from the link, extracts the packet from the frame, and delivers the packet to its network layer. The data-link layers of the sender and receiver provide transmission services for their network layers.

FSMs

The sender site should not send a frame until its network layer has a message to send. The receiver site cannot deliver a message to its network layer until a frame arrives. We can show these requirements using two FSMs. Each FSM has only one state, the *ready state*. The sending machine remains in the ready state until a request comes from the process in the network layer. When this event occurs, the sending machine encapsulates the message in a frame and sends it to the receiving machine. The receiving machine remains in the ready state until a frame arrives from the sending machine. When this event occurs, the receiving machine decapsulates the message out of the frame and delivers it to the process at the network layer. Figure 11.8 shows the FSMs for the simple protocol. We'll see more in Chapter 23, which uses this protocol.

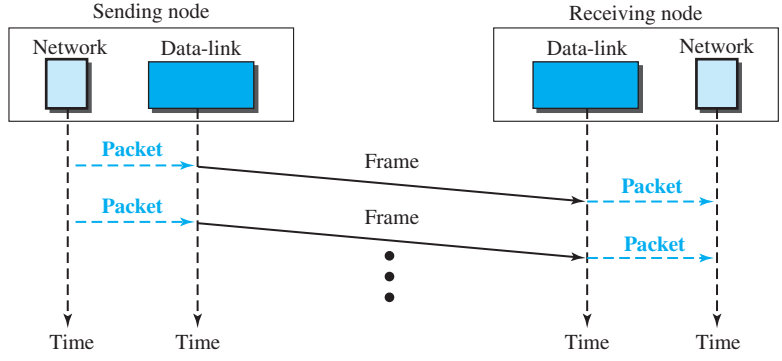
Figure 11.8 *FSMs for the simple protocol*



Example 11.2

Figure 11.9 shows an example of communication using this protocol. It is very simple. The sender sends frames one after another without even thinking about the receiver.

Figure 11.9 *Flow diagram for Example 11.2*

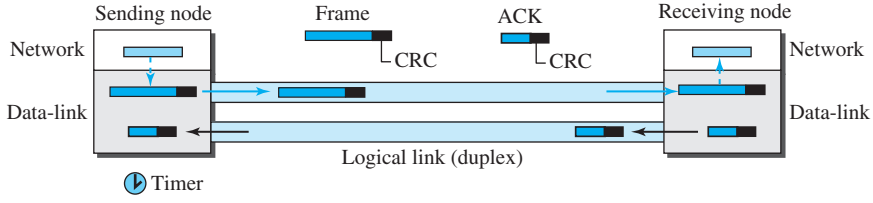


11.2.2 Stop-and-Wait Protocol

Our second protocol is called the **Stop-and-Wait protocol,** which uses both flow and error control. We show a primitive version of this protocol here, but we discuss the more sophisticated version in Chapter 23 when we have learned about sliding windows. In this protocol, the sender sends one frame at a time and waits for an acknowledgment before sending the next one. To detect corrupted frames, we need to add a CRC (see Chapter 10) to each data frame. When a frame arrives at the receiver site, it is checked. If its CRC is incorrect, the frame is corrupted and silently discarded. The silence of the receiver is a signal for the sender that a frame was either corrupted or lost. Every time the sender sends a frame, it starts a timer. If an acknowledgment arrives before the timer expires, the timer is stopped and the sender sends the next frame (if it has one to send). If the timer expires, the sender resends the previous frame, assuming that the frame was either lost or corrupted. This means that the sender needs to keep a copy of the frame until its acknowledgment arrives. When the corresponding

acknowledgment arrives, the sender discards the copy and sends the next frame if it is ready. Figure 11.10 shows the outline for the Stop-and-Wait protocol. Note that only one frame and one acknowledgment can be in the channels at any time.

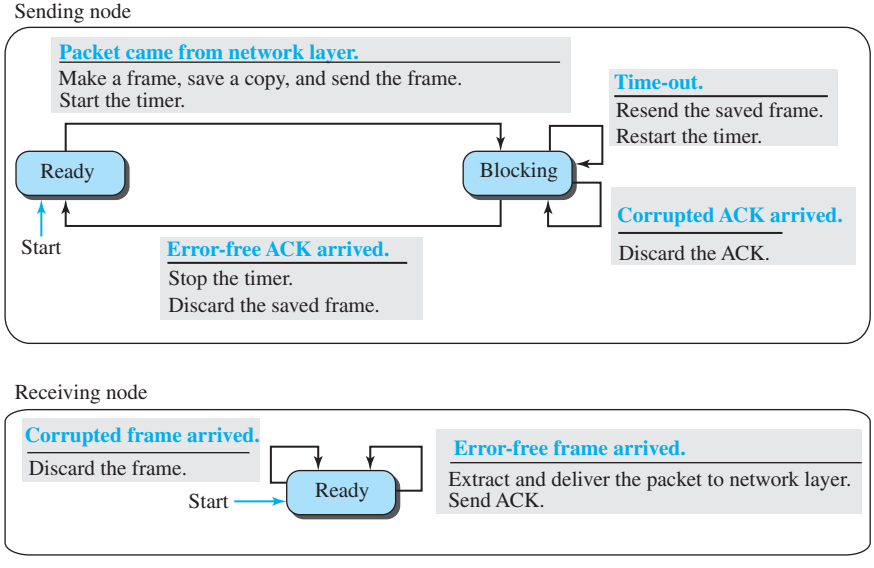
Figure 11.10 *Stop-and-Wait protocol*



FSMs

Figure 11.11 shows the FSMs for our primitive Stop-and-Wait protocol.

Figure 11.11 *FSM for the Stop-and-Wait protocol*



We describe the sender and receiver states below.

Sender States

The sender is initially in the ready state, but it can move between the ready and blocking state.

- ***Ready State.*** When the sender is in this state, it is only waiting for a packet from the network layer. If a packet comes from the network layer, the sender creates a frame, saves a copy of the frame, starts the only timer and sends the frame. The sender then moves to the blocking state.
- ***Blocking State.*** When the sender is in this state, three events can occur:
  - **a.** If a time-out occurs, the sender resends the saved copy of the frame and restarts the timer.
  - **b.** If a corrupted ACK arrives, it is discarded.
  - **c.** If an error-free ACK arrives, the sender stops the timer and discards the saved copy of the frame. It then moves to the ready state.

Receiver

The receiver is always in the *ready* state. Two events may occur:

- **a.** If an error-free frame arrives, the message in the frame is delivered to the network layer and an ACK is sent.
- **b.** If a corrupted frame arrives, the frame is discarded.

Example 11.3

Figure 11.12 shows an example. The first frame is sent and acknowledged. The second frame is sent, but lost. After time-out, it is resent. The third frame is sent and acknowledged, but the acknowledgment is lost. The frame is resent. However, there is a problem with this scheme. The network layer at the receiver site receives two copies of the third packet, which is not right. In the next section, we will see how we can correct this problem using sequence numbers and acknowledgment numbers.

Sequence and Acknowledgment Numbers

We saw a problem in Example 11.3 that needs to be addressed and corrected. Duplicate packets, as much as corrupted packets, need to be avoided. As an example, assume we are ordering some item online. If each packet defines the specification of an item to be ordered, duplicate packets mean ordering an item more than once. To correct the problem in Example 11.3, we need to add **sequence numbers** to the data frames and **acknowledgment numbers** to the ACK frames. However, numbering in this case is very simple. Sequence numbers are 0, 1, 0, 1, 0, 1, . . . ; the acknowledgment numbers can also be 1, 0, 1, 0, 1, 0, … In other words, the sequence numbers start with 0, the acknowledgment numbers start with 1. An acknowledgment number always defines the sequence number of the next frame to receive.

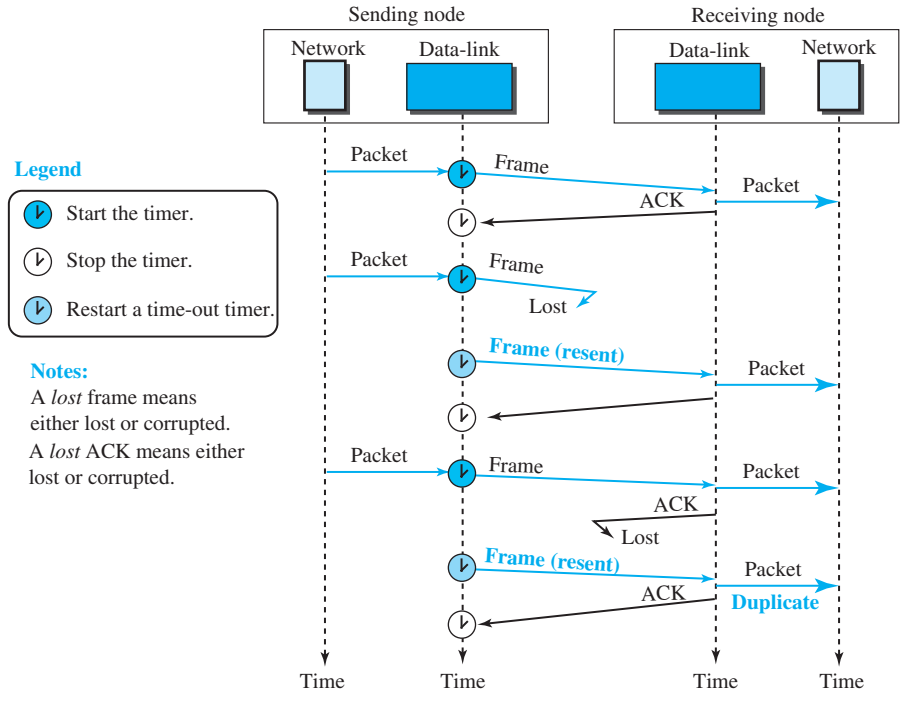
Example 11.4

Figure 11.13 shows how adding sequence numbers and acknowledgment numbers can prevent duplicates. The first frame is sent and acknowledged. The second frame is sent, but lost. After time-out, it is resent. The third frame is sent and acknowledged, but the acknowledgment is lost. The frame is resent.

FSMs with Sequence and Acknowledgment Numbers

We can change the FSM in Figure 11.11 to include the sequence and acknowledgment numbers, but we leave this as a problem at the end of the chapter.

Figure 11.12 *Flow diagram for Example 11.3*



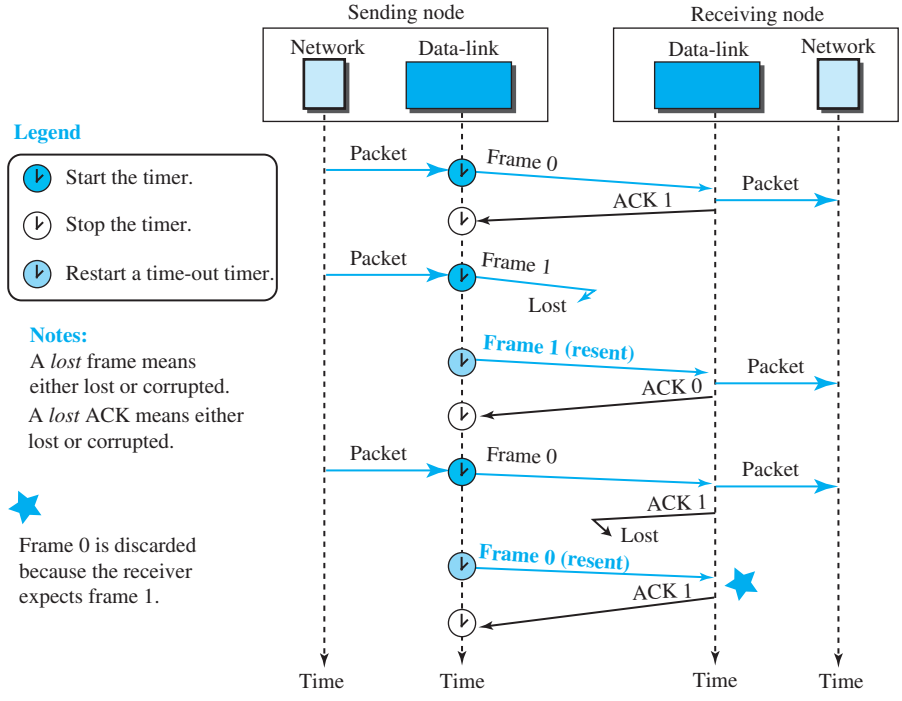
11.2.3 Piggybacking

The two protocols we discussed in this section are designed for unidirectional communication, in which data is flowing only in one direction although the acknowledgment may travel in the other direction. Protocols have been designed in the past to allow data to flow in both directions. However, to make the communication more efficient, the data in one direction is piggybacked with the acknowledgment in the other direction. In other words, when node A is sending data to node B, Node A also acknowledges the data received from node B. Because piggybacking makes communication at the data-link layer more complicated, it is not a common practice. We discuss two-way communication and piggybacking in more detail in Chapter 23.

11.3 HDLC

High-level Data Link Control (HDLC) is a bit-oriented protocol for communication over point-to-point and multipoint links. It implements the Stop-and-Wait protocol we discussed earlier. Although this protocol is more a theoretical issue than practical, most of the concept defined in this protocol is the basis for other practical protocols such as PPP, which we discuss next, or the Ethernet protocol, which we discuss in wired LANs (Chapter 13), or in wireless LANs (Chapter 15).

Figure 11.13 *Flow diagram for Example 11.4*



11.3.1 Configurations and Transfer Modes

HDLC provides two common transfer modes that can be used in different configurations: *normal response mode (NRM)* and *asynchronous balanced mode (ABM)*. In *normal response mode (NRM)*, the station configuration is unbalanced. We have one primary station and multiple secondary stations. A *primary station* can send commands; a *secondary station* can only respond. The NRM is used for both point-to-point and multipoint links, as shown in Figure 11.14.

In ABM, the configuration is balanced. The link is point-to-point, and each station can function as a primary and a secondary (acting as peers), as shown in Figure 11.15. This is the common mode today.

11.3.2 Framing

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

Figure 11.14 *Normal response mode*

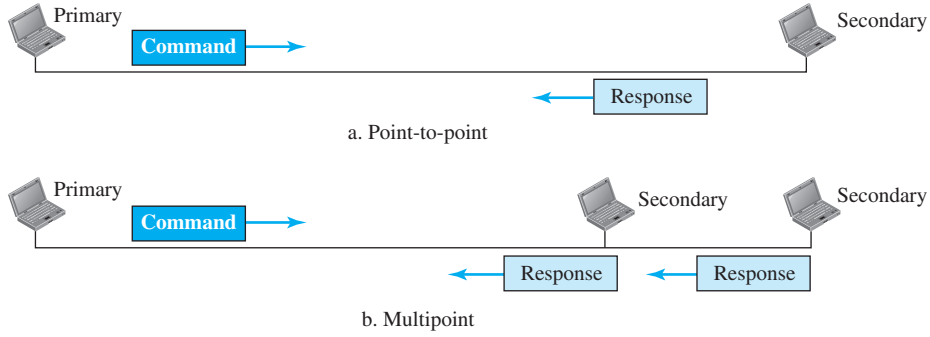
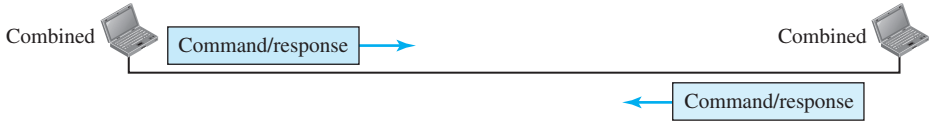
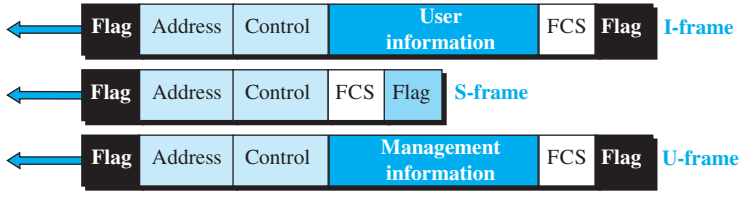


Figure 11.15 *Asynchronous balanced mode*



link itself. Each frame in HDLC may contain up to six fields, as shown in Figure 11.16: a beginning flag field, an address field, a control field, an information field, a frame check sequence (FCS) field, and an ending flag field. In multiple-frame transmissions, the ending flag of one frame can serve as the beginning flag of the next frame.

Figure 11.16 *HDLC frames*



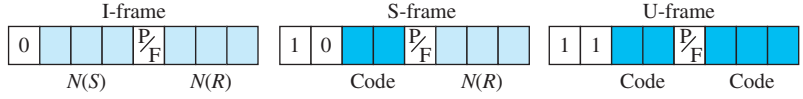
Let us now discuss the fields and their use in different frame types.

- ***Flag field.*** This field contains synchronization pattern 01111110, which identifies both the beginning and the end of a frame.
- ***Address field.*** This field contains the address of the secondary station. If a primary station created the frame, it contains a *to* address. If a secondary station creates the frame, it contains a *from* address. The address field can be one byte or several bytes long, depending on the needs of the network.

- ***Control field.*** The control field is one or two bytes used for flow and error control. The interpretation of bits are discussed later.
- ***Information field.*** The information field contains the user's data from the network layer or management information. Its length can vary from one network to another.
- ***FCS field.*** The frame check sequence (FCS) is the HDLC error detection field. It can contain either a 2- or 4-byte CRC.

The control field determines the type of frame and defines its functionality. So let us discuss the format of this field in detail. The format is specific for the type of frame, as shown in Figure 11.17.

Figure 11.17 *Control field format for the different frame types*



Control Field for I-Frames

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

Control Field for S-Frames

Supervisory frames are used for flow and error control whenever piggybacking is either impossible or inappropriate. S-frames do not have information fields. If the first 2 bits of the control field are 10, this means the frame is an S-frame. The last 3 bits, called $N(R)$, correspond to the acknowledgment number (ACK) or negative acknowledgment number (NAK), depending on the type of S-frame. The 2 bits called *code* are used to define the type of S-frame itself. With 2 bits, we can have four types of S-frames, as described below:

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

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

Control Field for U-Frames

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

Control Field for U-Frames

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

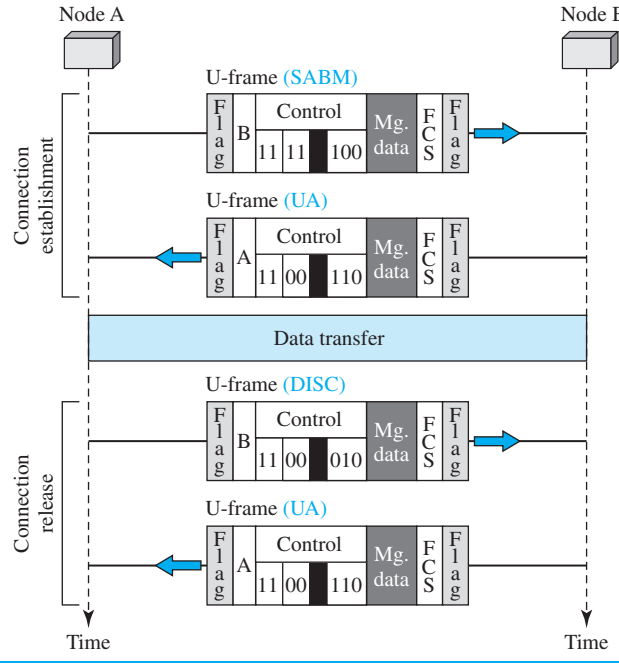
Example 11.5

Figure 11.18 shows how U-frames can be used for connection establishment and connection release. Node A asks for a connection with a set asynchronous balanced mode (SABM) frame; node B gives a positive response with an unnumbered acknowledgment (UA) frame. After these two exchanges, data can be transferred between the two nodes (not shown in the figure). After data transfer, node A sends a DISC (disconnect) frame to release the connection; it is confirmed by node B responding with a UA (unnumbered acknowledgment).

Example 11.6

Figure 11.19 shows two exchanges using piggybacking. The first is the case where no error has occurred; the second is the case where an error has occurred and some frames are discarded.

Figure 11.18 *Example of connection and disconnection*



11.4 POINT-TO-POINT PROTOCOL (PPP)

One of the most common protocols for point-to-point access is the **Point-to-Point Protocol (PPP).** Today, millions of Internet users who need to connect their home computers to the server of an Internet service provider use PPP. The majority of these users have a traditional modem; they are connected to the Internet through a telephone line, which provides the services of the physical layer. But to control and manage the transfer of data, there is a need for a point-to-point protocol at the data-link layer. PPP is by far the most common.

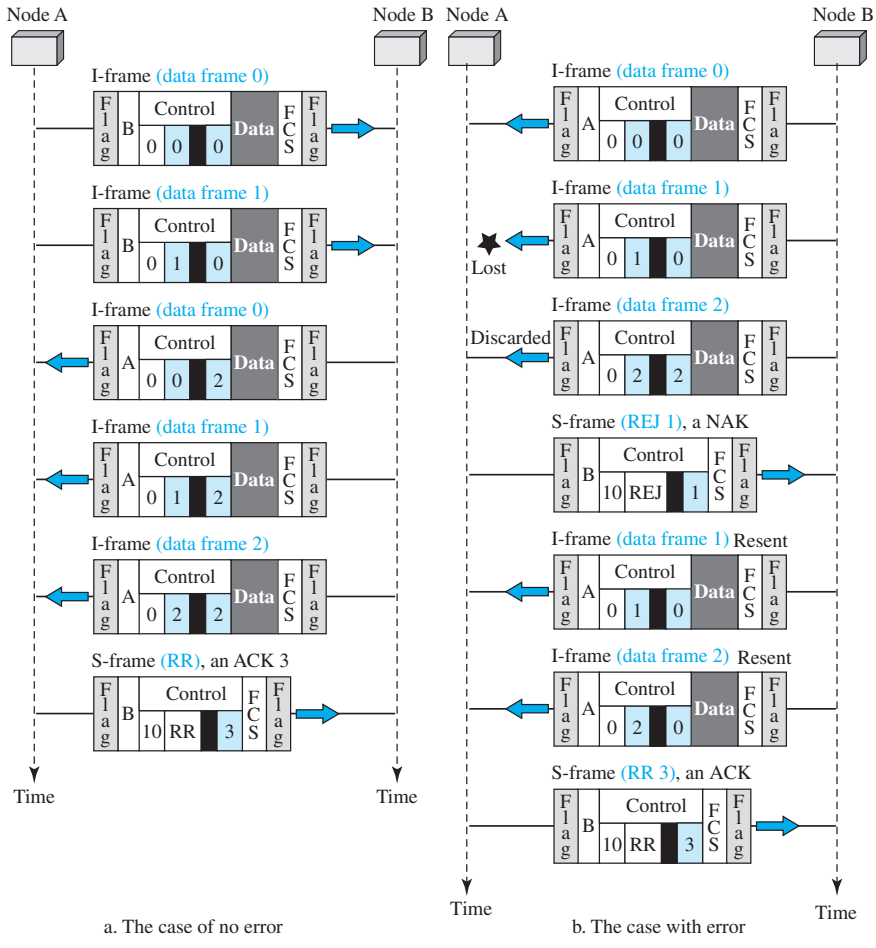
11.4.1 Services

The designers of PPP have included several services to make it suitable for a point-to-point protocol, but have ignored some traditional services to make it simple.

Services Provided by PPP

PPP defines the format of the frame to be exchanged between devices. It also defines how two devices can negotiate the establishment of the link and the exchange of data. PPP is designed to accept payloads from several network layers (not only IP). Authentication is also provided in the protocol, but it is optional. The new version of PPP, called *Multilink PPP,* provides connections over multiple links. One interesting feature of PPP is that it provides network address configuration. This is particularly useful when a home user needs a temporary network address to connect to the Internet.

Figure 11.19 *Example of piggybacking with and without error*



Services Not Provided by PPP

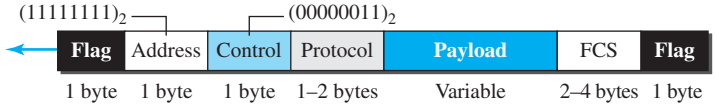
PPP does not provide flow control. A sender can send several frames one after another with no concern about overwhelming the receiver. PPP has a very simple mechanism for error control. A CRC field is used to detect errors. If the frame is corrupted, it is silently discarded; the upper-layer protocol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

11.4.2 Framing

PPP uses a character-oriented (or byte-oriented) frame. Figure 11.20 shows the format of a PPP frame. The description of each field follows:

- ***Flag.*** A PPP frame starts and ends with a 1-byte flag with the bit pattern 01111110.

Figure 11.20 *PPP frame format*



- ***Address.*** The address field in this protocol is a constant value and set to 11111111 (broadcast address).
- ***Control.*** This field is set to the constant value 00000011 (imitating unnumbered frames in HDLC). As we will discuss later, PPP does not provide any flow control. Error control is also limited to error detection.
- ***Protocol.*** The protocol field defines what is being carried in the data field: either user data or other information. This field is by default 2 bytes long, but the two parties can agree to use only 1 byte.
- ***Payload field.*** This field carries either the user data or other information that we will discuss shortly. The data field is a sequence of bytes with the default of a maximum of 1500 bytes; but this can be changed during negotiation. The data field is byte-stuffed if the flag byte pattern appears in this field. Because there is no field defining the size of the data field, padding is needed if the size is less than the maximum default value or the maximum negotiated value.
- ***FCS.*** The frame check sequence (FCS) is simply a 2-byte or 4-byte standard CRC.

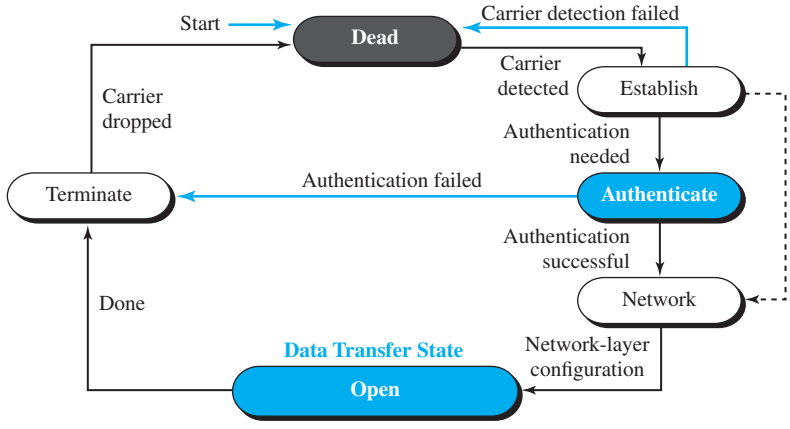
Byte Stuffing

Since PPP is a byte-oriented protocol, the flag in PPP is a byte that needs to be escaped whenever it appears in the data section of the frame. The escape byte is 01111101, which means that every time the flaglike pattern appears in the data, this extra byte is stuffed to tell the receiver that the next byte is not a flag. Obviously, the escape byte itself should be stuffed with another escape byte.

11.4.3 Transition Phases

A PPP connection goes through phases which can be shown in a *transition phase* diagram (see Figure 11.21). The transition diagram, which is an FSM, starts with the *dead* state. In this state, there is no active carrier (at the physical layer) and the line is quiet. When one of the two nodes starts the communication, the connection goes into the *establish* state. In this state, options are negotiated between the two parties. If the two parties agree that they need authentication (for example, if they do not know each other), then the system needs to do authentication (an extra step); otherwise, the parties can simply start communication. The link-control protocol packets, discussed shortly, are used for this purpose. Several packets may be exchanged here. Data transfer takes place in the *open* state. When a connection reaches this state, the exchange of data packets can be started. The connection remains in this state until one of the endpoints wants to terminate the connection. In this case, the system goes to the *terminate* state. The system remains in this state until the carrier (physical-layer signal) is dropped, which moves the system to the *dead* state again.

Figure 11.21 *Transition phases*



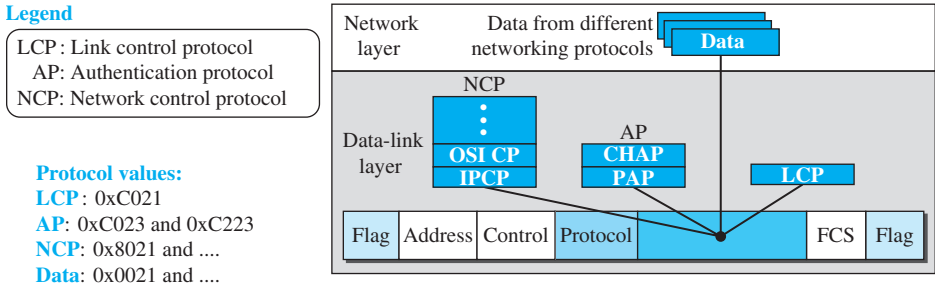
11.4.4 Multiplexing

Although PPP is a link-layer protocol, it uses another set of protocols to establish the link, authenticate the parties involved, and carry the network-layer data. Three sets of protocols are defined to make PPP powerful: the Link Control Protocol (LCP), two Authentication Protocols (APs), and several Network Control Protocols (NCPs). At any moment, a PPP packet can carry data from one of these protocols in its data field, as shown in Figure 11.22. Note that there are one LCP, two APs, and several NCPs. Data may also come from several different network layers.

Link Control Protocol

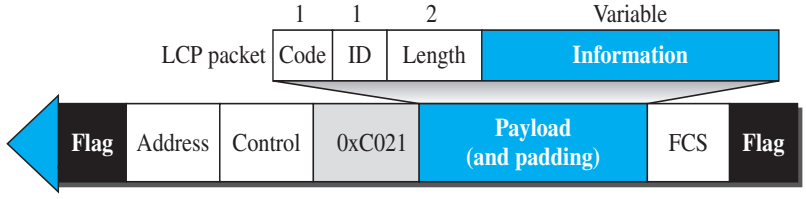
The **Link Control Protocol (LCP)** is responsible for establishing, maintaining, configuring, and terminating links. It also provides negotiation mechanisms to set options between the two endpoints. Both endpoints of the link must reach an agreement about the options before the link can be established. See Figure 11.21.

Figure 11.22 *Multiplexing in PPP*



All LCP packets are carried in the payload field of the PPP frame with the protocol field set to C021 in hexadecimal (see Figure 11.23).

Figure 11.23 *LCP packet encapsulated in a frame*



The code field defines the type of LCP packet. There are 11 types of packets, as shown in Table 11.1.

Table 11.1 *LCP packets*

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

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

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

There are many options that can be negotiated between the two endpoints. Options are inserted in the information field of the configuration packets. In this case, the

information field is divided into three fields: option type, option length, and option data. We list some of the most common options in Table 11.2.

Table 11.2 *Common options*

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

Authentication Protocols

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

PAP

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

a. The user who wants to access a system sends an authentication identification (usually the user name) and a password.

b. The system checks the validity of the identification and password and either accepts or denies connection.

Figure 11.24 shows the three types of packets used by PAP and how they are actually exchanged. When a PPP frame is carrying any PAP packets, the value of the protocol field is 0xC023. The three PAP packets are authenticate-request, authenticate-ack, and authenticate-nak. The first packet is used by the user to send the user name and password. The second is used by the system to allow access. The third is used by the system to deny access.

CHAP

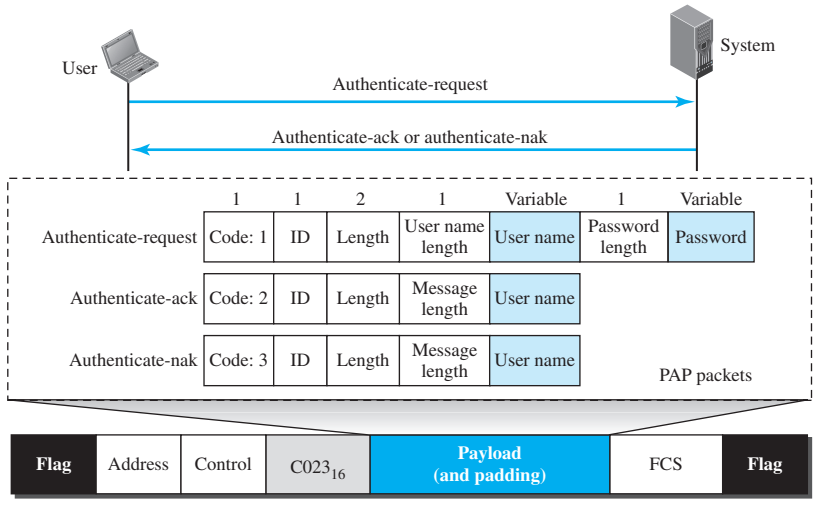
The **Challenge Handshake Authentication Protocol (CHAP)** is a three-way handshaking authentication protocol that provides greater security than PAP. In this method, the password is kept secret; it is never sent online.

a. The system sends the user a challenge packet containing a challenge value, usually a few bytes.

b. The user applies a predefined function that takes the challenge value and the user's own password and creates a result. The user sends the result in the response packet to the system.

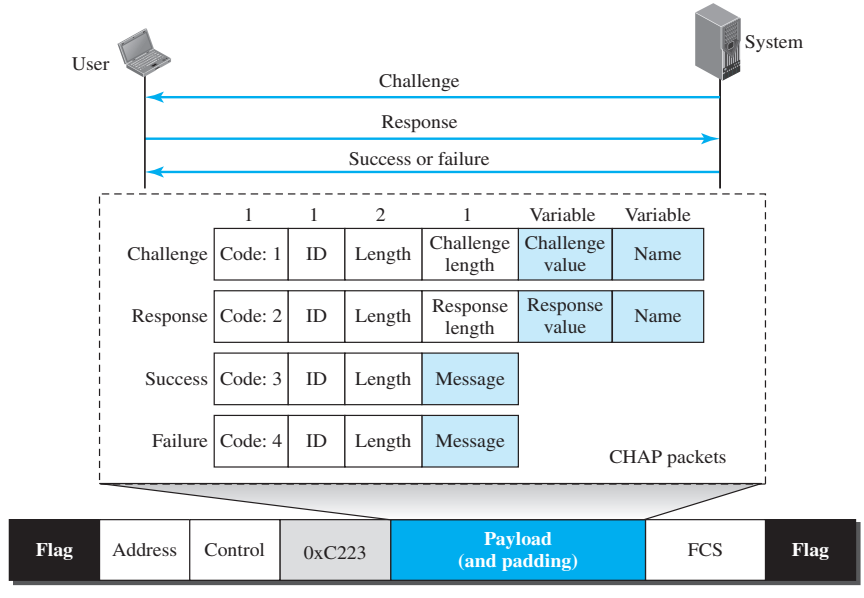
c. The system does the same. It applies the same function to the password of the user (known to the system) and the challenge value to create a result. If the

Figure 11.24 *PAP packets encapsulated in a PPP frame*



result created is the same as the result sent in the response packet, access is granted; otherwise, it is denied. CHAP is more secure than PAP, especially if the system continuously changes the challenge value. Even if the intruder learns the challenge value and the result, the password is still secret. Figure 11.25 shows the packets and how they are used.

Figure 11.25 *CHAP packets encapsulated in a PPP frame*



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

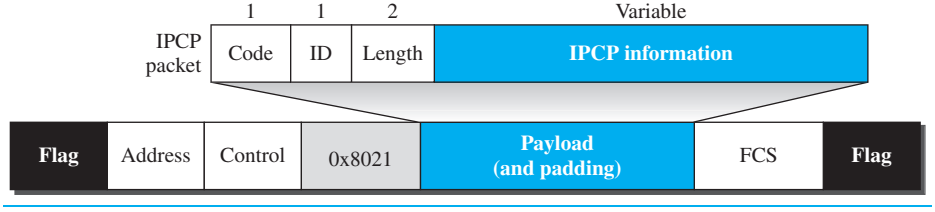
Network Control Protocols

PPP is a multiple-network-layer protocol. It can carry a network-layer data packet from protocols defined by the Internet, OSI, Xerox, DECnet, AppleTalk, Novel, and so on. To do this, PPP has defined a specific Network Control Protocol for each network protocol. For example, IPCP (Internet Protocol Control Protocol) configures the link for carrying IP data packets. Xerox CP does the same for the Xerox protocol data packets, and so on. Note that none of the NCP packets carry network-layer data; they just configure the link at the network layer for the incoming data.

IPCP

One NCP protocol is the **Internet Protocol Control Protocol (IPCP).** This protocol configures the link used to carry IP packets in the Internet. IPCP is especially of interest to us. The format of an IPCP packet is shown in Figure 11.26. Note that the value of the protocol field in hexadecimal is 8021.

Figure 11.26 *IPCP packet encapsulated in PPP frame*



IPCP defines seven packets, distinguished by their code values, as shown in Table 11.3.

Table 11.3 *Code value for IPCP packets*

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

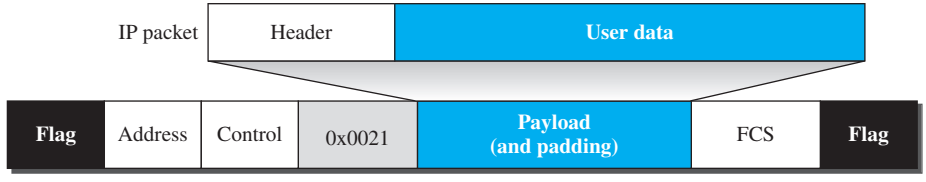
Other Protocols

There are other NCP protocols for other network-layer protocols. The OSI Network Layer Control Protocol has a protocol field value of 8023; the Xerox NS IDP Control Protocol has a protocol field value of 8025; and so on.

Data from the Network Layer

After the network-layer configuration is completed by one of the NCP protocols, the users can exchange data packets from the network layer. Here again, there are different protocol fields for different network layers. For example, if PPP is carrying data from the IP network layer, the field value is 0021 (note that the three rightmost digits are the same as for IPCP). If PPP is carrying data from the OSI network layer, the value of the protocol field is 0023, and so on. Figure 11.27 shows the frame for IP.

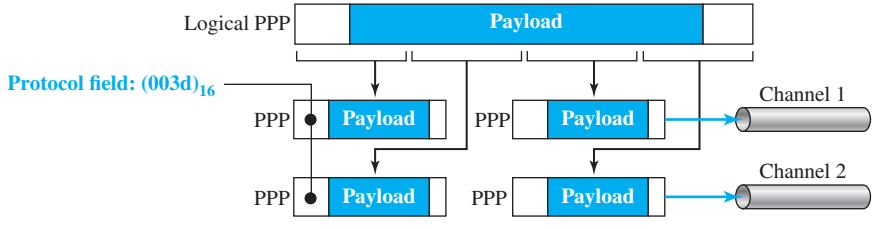
Figure 11.27 *IP datagram encapsulated in a PPP frame*



Multilink PPP

PPP was originally designed for a single-channel point-to-point physical link. The availability of multiple channels in a single point-to-point link motivated the development of Multilink PPP. In this case, a logical PPP frame is divided into several actual PPP frames. A segment of the logical frame is carried in the payload of an actual PPP frame, as shown in Figure 11.28. To show that the actual PPP frame is carrying a fragment of a logical PPP frame, the protocol field is set to $(003d)_{16}$. This new development adds complexity. For example, a sequence number needs to be added to the actual PPP frame to show a fragment's position in the logical frame.

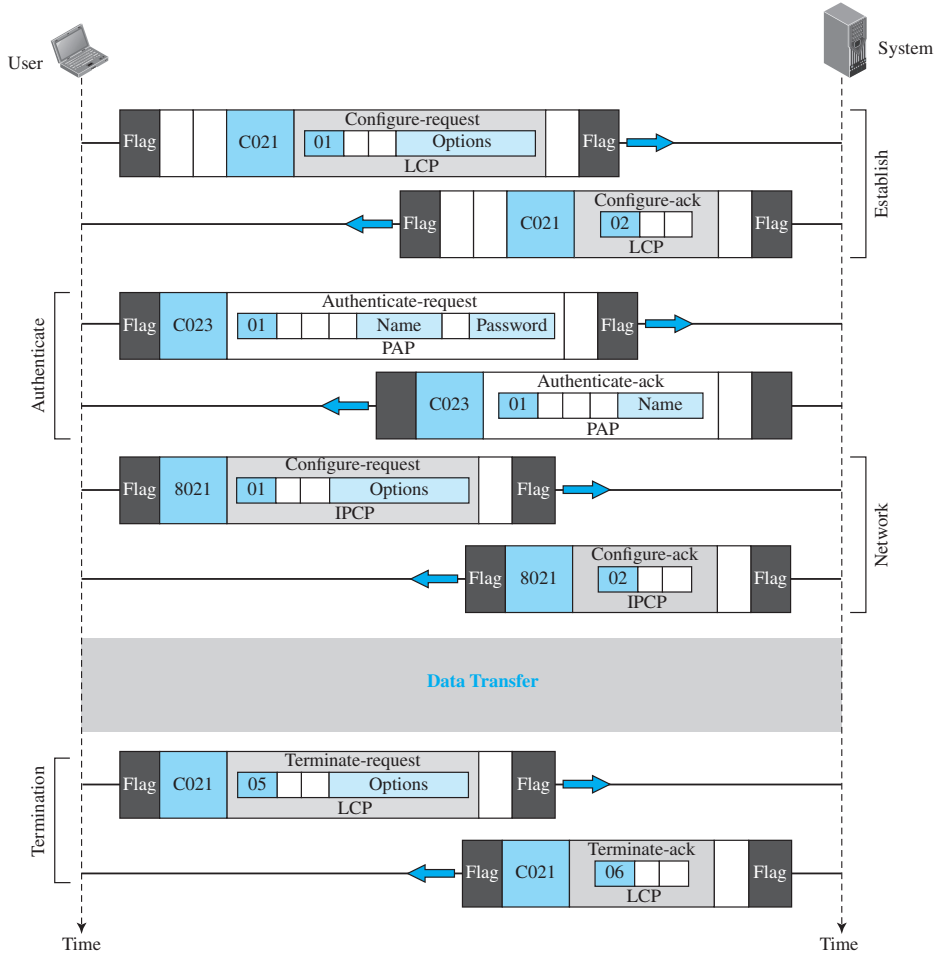
Figure 11.28 *Multilink PPP*



Example 11.7

Let us go through the phases followed by a network layer packet as it is transmitted through a PPP connection. Figure 11.29 shows the steps. For simplicity, we assume unidirectional movement of data from the user site to the system site (such as sending an e-mail through an ISP).

Figure 11.29 *An example*



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

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

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

11.5 END-CHAPTER MATERIALS

11.5.1 Recommended Reading

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

Books

Several books discuss link-layer issues. Among them we recommend [Ham 80], [Zar 02], [Ror 96], [Tan 03], [GW 04], [For 03], [KMK 04], [Sta 04], [Kes 02], [PD 03], [Kei 02], [Spu 00], [KCK 98], [Sau 98], [Izz 00], [Per 00], and [WV 00].

11.5.2 Key Terms

acknowledgment number
bit stuffing
byte stuffing
Challenge Handshake Authentication Protocol (CHAP)
data link control (DLC)
finite state machine (FSM)
flag
High-level Data Link Control (HDLC)
Internet Protocol Control Protocol (IPCP)
Link Control Protocol (LCP)
Password Authentication Protocol (PAP)
piggybacking
Point-to-Point Protocol (PPP)
sequence number
Simple Protocol
Stop-and-Wait Protocol

11.5.3 Summary

Data link control deals with the design and procedures for communication between two adjacent nodes: node-to-node communication. Framing in the data-link layer separates one packet from another. In fixed-size framing, there is no need for defining the boundaries of frames; in variable-size framing, we need a delimiter (flag) to define the boundary of two frames. Variable-size framing uses two categories of protocols: byte-oriented (or character-oriented) and bit-oriented. In a byte-oriented protocol, the data section of a frame is a sequence of bytes; in a bit-oriented protocol, the data section of a frame is a sequence of bits. In byte-oriented protocols, we use byte stuffing; in bit-oriented protocols, we use bit stuffing.

Another duty of DLC is flow and error control. At the data-link layer, flow control means creating a balance between the frames sent by a node and the frames that can be handled by the next node. Error control at the data-link layer is normally implemented very simply. Corrupted frames are silently discarded; uncorrupted frames are accepted with or without sending acknowledgments to the sender.

A DLC protocol can be either connectionless or connection-oriented. In a connectionless protocol, frames are sent from one node to the next without any relationship between the frames; each frame is independent. In a connection-oriented protocol, a logical connection should first be established between the two nodes before sending the data frames. After all related frames are transmitted, the logical connection is terminated.

Data-link protocols have been designed to handle communication between two nodes. We discussed two protocols in this chapter. In the Simple Protocol, there is no flow and error control. In the Stop-and-Wait Protocol, there are both flow and error controls, but communication is a frame at a time.

High-level Data Link Control (HDLC) is a bit-oriented protocol for communication over point-to-point and multipoint links. It implements the Stop-and-Wait protocol. It is the basis of many protocols in practice today. HDLC defines three types of frames: information frames, supervisory frames, and unnumbered frames. The informational frames are used to carry data frames. Supervisory frames are used only to transport control information for flow and error control. Unnumbered frames are reserved for system management and provide connection-oriented service.

One of the most common protocols for point-to-point access is the Point-to-Point Protocol (PPP). PPP uses only one type of frame, but allows multiplexing of different payloads to achieve a kind of connection-oriented service authentication. Encapsulating different packets in a frame allows PPP to move to different states to provide necessary services.

11.6 PRACTICE SET

11.6.1 Quizzes

A set of interactive quizzes for this chapter can be found on the book website. It is strongly recommended that the student take the quizzes to check his/her understanding of the materials before continuing with the practice set.

11.6.2 Questions

Q11-1. Define *framing* and give the reason it is needed.

Q11-2. Explain why flags are needed when we use variable-size frames.

Q11-3. Assume a new character-oriented protocol is using the 16-bit Unicode as the character set. What should the size of the flag be in this protocol?

Q11-4. Compare and contrast byte-oriented and bit-oriented protocols.

Q11-5. Compare and contrast byte-stuffing and bit-stuffing.

Q11-6. In a byte-oriented protocol, should we first unstuff the extra bytes and then remove the flags or reverse the process?

Q11-7. In a bit-oriented protocol, should we first unstuff the extra bits and then remove the flags or reverse the process?

Q11-8. Compare and contrast flow control and error control.

Q11-9. In the Stop-and-Wait Protocol, assume that the sender has only one slot in which to keep the frame to send or the copy of the sent frame. What happens if the network layer delivers a packet to the data-link layer at this moment?

Q11-10. In Example 11.3 (Figure 11.12) how many frames are in transit at the same time?

Q11-11. In Example 11.4 (Figure 11.13) how many frames are in transit at the same time?

Q11-12. In the traditional Ethernet protocol (Chapter 13), the frames are sent with the CRC. If the frame is corrupted, the receiving node just discards it. Is this an example of a Simple Protocol or the Stop-and-Wait Protocol? Explain.

Q11-13. In Figure 11.11, do the ready and blocking states use the same timer? Explain.

Q11-14. Explain why there is no need for CRC in the Simple Protocol.

Q11-15. In Figure 11.9, we show the packet path as a horizontal line, but the frame path as a diagonal line. Can you explain the reason?

Q11-16. In Figure 11.12, explain why we need a timer at the sending site, but none at the receiving site.

Q11-17. Does the duplex communication in Figure 11.10 necessarily mean we need two separate media between the two nodes? Explain.

Q11-18. Define *piggybacking* and its benefit.

Q11-19. In Figure 11.16, which frame type can be used for acknowledgment?

Q11-20. Compare Figure 11.6 and Figure 11.21. If both are FSMs, why are there no event/action pairs in the second?

Q11-21. In PPP, we normally talk about *user* and *system* instead of *sending and receiving nodes;* explain the reason.

Q11-22. Compare and contrast HDLC with PPP.

Q11-23. Compare the flag byte and the escape byte in PPP. Are they are the same? Explain.

Q11-24. In Figure 11.20, explain why we need only one address field. Explain why the address is set to the predefined value of $(11111111)_2$.

11.6.3 Problems

P11-1. Byte-stuff the following frame payload in which E is the escape byte, F is the flag byte, and D is a data byte other than an escape or a flag character.

D	**E**	D	D	**F**	D	D	**E**	**E**	D	**F**	D

P11-2. Unstuff the following frame payload in which E is the escape byte, F is the flag byte, and D is a data byte other than an escape or a flag character.

E	**E**	D	**E**	**F**	D	D	**E**	**F**	**E**	**E**	D	D	D	

P11-3. Bit-stuff the following frame payload:

```
000111111100111110100011111111110000111
```

P11-4. Unstuff the following frame payload:

```
0001111100000111110111010011101111000001111
```

P11-5. Assume we change the Stop-and-Wait Protocol to include a NAK (negative feedback), which is used only when a corrupted frame arrives and is discarded. Redraw Figure 11.9 to show this change.

P11-6. In Example 11.4 (Figure 11.13), assume the round trip time for a frame is 40 milliseconds. Explain what will happen if we set the time-out in each of the following cases.

a. 35 milliseconds **b.** 45 milliseconds **c.** 40 milliseconds

P11-7. Redraw Figure 11.12 using the following scenario:

a. The first frame is sent and acknowledged.

b. The second frame is sent and acknowledged, but the acknowledgment is lost.

c. The second frame is resent, but it is timed-out.

d. The second frame is resent and acknowledged.

P11-8. Redraw Figure 11.2 using the following scenario:

a. Frame 0 is sent, but lost.

b. Frame 0 is resent and acknowledged.

c. Frame 1 is sent and acknowledged, but the acknowledgment is lost.

d. Frame 1 is resent and acknowledged.

P11-9. In Figure 11.11, show what happens in each of the following cases:

a. The sender is at the ready state and an error-free ACK arrives.

b. The sender is at the blocking state and a time-out occurs.

c. The sender is at the ready state and a time-out occurs.

P11-10. In Figure 11.11, show what happens in each of the following cases:

a. The receiver is in the ready state and a packet comes from the network layer.

b. The receiver is in the ready state and a corrupted frame arrives.

c. The receiver is in the ready state and an acknowledgment arrives.

P11-11. Using the following specifications, draw a finite state machine with three states (I, II, and III), five events, and six actions:

a. If the machine is in state I, two events can occur. If event 1 occurs, the machine moves to state II. If event 2 occurs, the machine performs actions 1 and 2 and moves to state III.

b. If the machine is in state II, two events can occur. If event 3 occurs, the machine remains in state II. If event 4 occurs, the machine moves to state III.

c. If the machine is in state III, three events can occur. If event 2 occurs, the machine remains in state III. If event 3 occurs, the machine performs actions 1, 2, 4, and 5 moves to state II. If event 5 occurs, the machine performs actions 1, 2, and 6 and moves to state I.

P11-12. Using the following specifications, draw a finite state machine with three states (I, II, and III), six events, and four actions:

a. If the machine is in state I, two events can occur. If event 1 occurs, the machine moves to state III. If event 3 occurs, the machine performs actions 2 and 4 and moves to state II.

b. If the machine is in state II, two events can occur. If event 4 occurs, the machine remains in state II. If event 6 occurs, the machine performs actions 1 and 2 and moves to state III.

c. If the machine is in state III, three events can occur. If event 2 occurs, the machine remains in state III. If event 6 occurs, the machine performs actions 2, 3, 4, and 5 moves to state I. If event 4 occurs, the machine performs actions 1 and 2 and moves to state I.

P11-13. Redraw Figure 11.11 using a variable to hold the one-bit sequence number and a variable to hold the one-bit acknowledgment number.

P11-14. Redraw Figure 11.10 using piggybacking.

P11-15. Assume PPP is in the established phase; show payload encapsulated in the frame.

P11-16. Redraw Figure 11.21 with the system not using authentication.

P11-17. Assume PPP is in the authentication phase, show payload exchanged between the nodes if PPP is using

a. PAP

b. CHAP

P11-18. Assume the only computer in the residence uses PPP to communicate with the ISP. If the user sends 10 network-layer packets to ISP, how many frames are exchanged in each of the following cases:

a. Using no authentication?

b. Using PAP for authentication?

c. Using CHAP for authentication?

11.7 SIMULATION EXPERIMENTS

11.7.1 Applets

We have created some Java applets to show some of the main concepts discussed in this chapter. It is strongly recommended that the students activate these applets on the book website and carefully examine the protocols in action.

11.8 PROGRAMMING ASSIGNMENTS

For each of the following assignments, write a program in the programming language you are familiar with.

Prg11-1. Write and test a program that simulates the byte stuffing and byte unstuffing as shown in Figure 11.2.

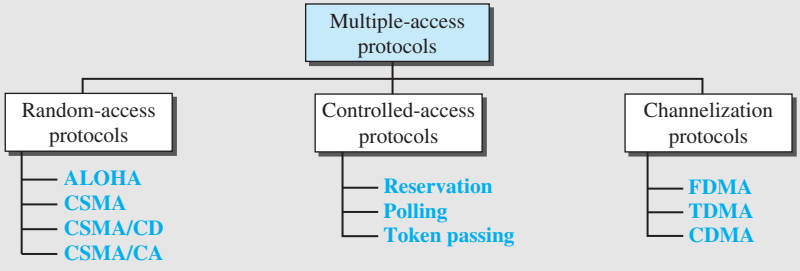
Prg11-2. Write and test a program that simulates the bit stuffing and bit unstuffing as shown in Figure 11.4.

CHAPTER 12

Media Access Control (MAC)

When nodes or stations are connected and use a common link, called a *multipoint* or *broadcast link,* we need a multiple-access protocol to coordinate access to the link. The problem of controlling the access to the medium is similar to the rules of speaking in an assembly. The procedures guarantee that the right to speak is upheld and ensure that two people do not speak at the same time, do not interrupt each other, do not monopolize the discussion, and so on. Many protocols have been devised to handle access to a shared link. All of these protocols belong to a sublayer in the data-link layer called *media access control (MAC).* We categorize them into three groups, as shown in Figure 12.1.

Figure 12.1 *Taxonomy of multiple-access protocols*



This chapter is divided into three sections:

- The first section discusses random-access protocols. Four protocols, ALOHA, CSMA, CSMA/CD, and CSMA/CA, are described in this section. These protocols are mostly used in LANs and WANs, which we discuss in future chapters.
- The second section discusses controlled-access protocols. Three protocols, reservation, polling, and token-passing, are described in this section. Some of these protocols are used in LANs, but others have some historical value.
- The third section discusses channelization protocols. Three protocols, FDMA, TDMA, and CDMA are described in this section. These protocols are used in cellular telephony, which we discuss in Chapter 16.

12.1 RANDOM ACCESS

In **random-access** or **contention** methods, no station is superior to another station and none is assigned control over another. At each instance, a station that has data to send uses a procedure defined by the protocol to make a decision on whether or not to send. This decision depends on the state of the medium (idle or busy). In other words, each station can transmit when it desires on the condition that it follows the predefined procedure, including testing the state of the medium.

Two features give this method its name. First, there is no scheduled time for a station to transmit. Transmission is random among the stations. That is why these methods are called *random access*. Second, no rules specify which station should send next. Stations compete with one another to access the medium. That is why these methods are also called *contention* methods.

In a random-access method, each station has the right to the medium without being controlled by any other station. However, if more than one station tries to send, there is an access conflict—***collision***—and the frames will be either destroyed or modified. To avoid access conflict or to resolve it when it happens, each station follows a procedure that answers the following questions:

- When can the station access the medium?
- What can the station do if the medium is busy?
- How can the station determine the success or failure of the transmission?
- What can the station do if there is an access conflict?

The random-access methods we study in this chapter have evolved from a very interesting protocol known as *ALOHA*, which used a very simple procedure called ***multiple access (MA).*** The method was improved with the addition of a procedure that forces the station to sense the medium before transmitting. This was called *carrier sense multiple access* (*CSMA*). This method later evolved into two parallel methods: *carrier sense multiple access with collision detection (CSMA/CD),* which tells the station what to do when a collision is detected, and *carrier sense multiple access with collision avoidance (CSMA/CA),* which tries to avoid the collision.

12.1.1 ALOHA

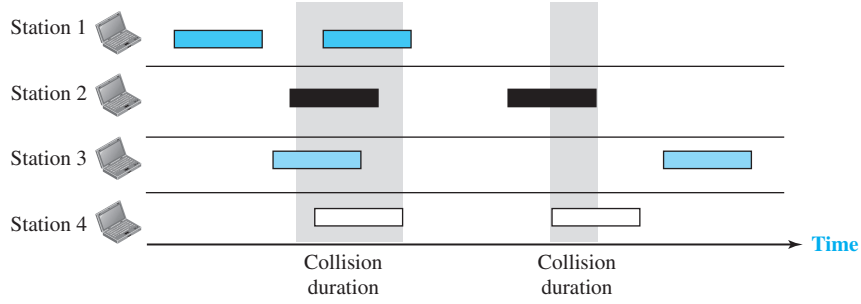
ALOHA, the earliest random access method, was developed at the University of Hawaii in early 1970. It was designed for a radio (wireless) LAN, but it can be used on any shared medium.

It is obvious that there are potential collisions in this arrangement. The medium is shared between the stations. When a station sends data, another station may attempt to do so at the same time. The data from the two stations collide and become garbled.

Pure ALOHA

The original ALOHA protocol is called ***pure ALOHA.*** This is a simple but elegant protocol. The idea is that each station sends a frame whenever it has a frame to send (multiple access). However, since there is only one channel to share, there is the possibility of collision between frames from different stations. Figure 12.2 shows an example of frame collisions in pure ALOHA.

Figure 12.2 *Frames in a pure ALOHA network*



There are four stations (unrealistic assumption) that contend with one another for access to the shared channel. The figure shows that each station sends two frames; there are a total of eight frames on the shared medium. Some of these frames collide because multiple frames are in contention for the shared channel. Figure 12.2 shows that only two frames survive: one frame from station 1 and one frame from station 3. We need to mention that even if one bit of a frame coexists on the channel with one bit from another frame, there is a collision and both will be destroyed. It is obvious that we need to resend the frames that have been destroyed during transmission.

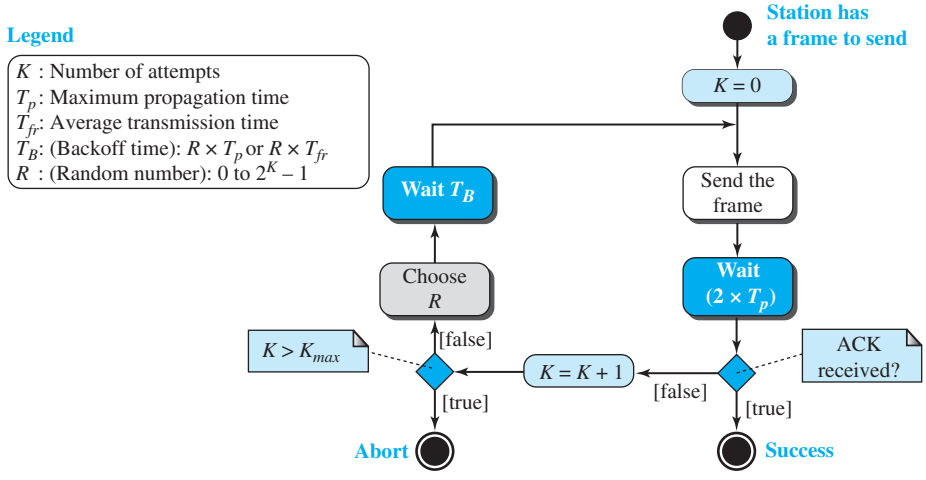
The pure ALOHA protocol relies on acknowledgments from the receiver. When a station sends a frame, it expects the receiver to send an acknowledgment. If the acknowledgment does not arrive after a time-out period, the station assumes that the frame (or the acknowledgment) has been destroyed and resends the frame.

A collision involves two or more stations. If all these stations try to resend their frames after the time-out, the frames will collide again. Pure ALOHA dictates that when the time-out period passes, each station waits a random amount of time before resending its frame. The randomness will help avoid more collisions. We call this time the *backoff time* T_B.

Pure ALOHA has a second method to prevent congesting the channel with retransmitted frames. After a maximum number of retransmission attempts K_{max}, a station must give up and try later. Figure 12.3 shows the procedure for pure ALOHA based on the above strategy.

The time-out period is equal to the maximum possible round-trip propagation delay, which is twice the amount of time required to send a frame between the two most widely separated stations ($2 \times T_p$). The backoff time T_B is a random value that normally depends on K (the number of attempted unsuccessful transmissions). The formula for T_B depends on the implementation. One common formula is the ***binary exponential backoff.*** In this method, for each retransmission, a multiplier $R = 0$ to $2^K - 1$ is randomly chosen and multiplied by T_p (maximum propagation time) or T_{fr} (the average time required to send out a frame) to find T_B. Note that in this procedure, the range of the random numbers increases after each collision. The value of K_{max} is usually chosen as 15.

Figure 12.3 *Procedure for pure ALOHA protocol*



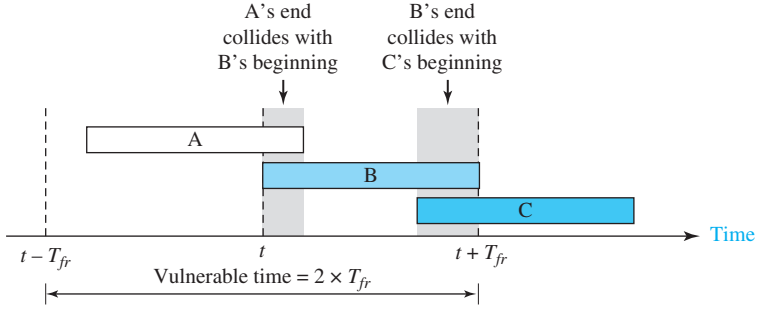
Example 12.1

The stations on a wireless ALOHA network are a maximum of 600 km apart. If we assume that signals propagate at 3×10^8 m/s, we find $T_p = (600 \times 10^3) / (3 \times 10^8) = 2$ ms. For $K = 2$, the range of R is {0, 1, 2, 3}. This means that T_B can be 0, 2, 4, or 6 ms, based on the outcome of the random variable R.

Vulnerable time

Let us find the ***vulnerable time,*** the length of time in which there is a possibility of collision. We assume that the stations send fixed-length frames with each frame taking T_{fr} seconds to send. Figure 12.4 shows the vulnerable time for station B.

Figure 12.4 *Vulnerable time for pure ALOHA protocol*



Station B starts to send a frame at time t. Now imagine station A has started to send its frame after $t - T_{fr}$. This leads to a collision between the frames from station B and

station A. On the other hand, suppose that station C starts to send a frame before time $t + T_{fr}$. Here, there is also a collision between frames from station B and station C.

Looking at Figure 12.4, we see that the vulnerable time during which a collision may occur in pure ALOHA is 2 times the frame transmission time.

$$\textbf{Pure ALOHA vulnerable time} = 2 \times T_{fr}$$

Example 12.2

A pure ALOHA network transmits 200-bit frames on a shared channel of 200 kbps. What is the requirement to make this frame collision-free?

Solution

Average frame transmission time T_{fr} is 200 bits/200 kbps or 1 ms. The vulnerable time is 2×1 ms = 2 ms. This means no station should send later than 1 ms before this station starts transmission and no station should start sending during the period (1 ms) that this station is sending.

Throughput

Let us call G the average number of frames generated by the system during one frame transmission time. Then it can be proven that the average number of successfully transmitted frames for pure ALOHA is $S = G \times e^{-2G}$. The maximum throughput S_{max} is 0.184, for $G = 1/2$. (We can find it by setting the derivative of S with respect to G to 0; see Exercises.) In other words, if one-half a frame is generated during one frame transmission time (one frame during two frame transmission times), then 18.4 percent of these frames reach their destination successfully. We expect $G = 1/2$ to produce the maximum throughput because the vulnerable time is 2 times the frame transmission time. Therefore, if a station generates only one frame in this vulnerable time (and no other stations generate a frame during this time), the frame will reach its destination successfully.

The throughput for pure ALOHA is $S = G \times e^{-2G}$.
The maximum throughput $S_{max} = 1/(2e) = 0.184$ when $G = (1/2)$.

Example 12.3

A pure ALOHA network transmits 200-bit frames on a shared channel of 200 kbps. What is the throughput if the system (all stations together) produces

a. 1000 frames per second?

b. 500 frames per second?

c. 250 frames per second?

Solution

The frame transmission time is 200/200 kbps or 1 ms.

a. If the system creates 1000 frames per second, or 1 frame per millisecond, then $G = 1$. In this case $S = G \times e^{-2G} = 0.135$ (13.5 percent). This means that the throughput is $1000 \times 0.135 = 135$ frames. Only 135 frames out of 1000 will probably survive.

b. If the system creates 500 frames per second, or 1/2 frames per millisecond, then $G = 1/2$. In this case $S = G \times e^{-2G} = 0.184$ (18.4 percent). This means that the throughput is $500 \times 0.184 = 92$ and that only 92 frames out of 500 will probably survive. Note that this is the *maximum* throughput case, percentagewise.

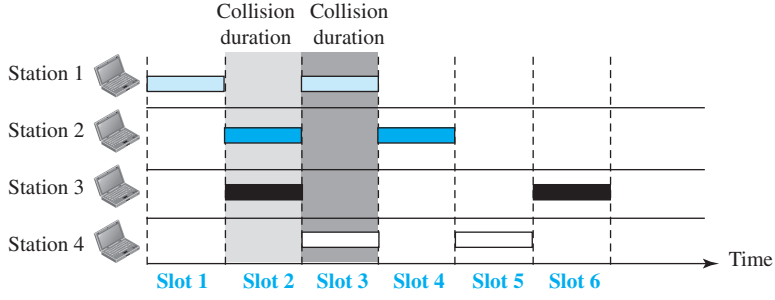
c. If the system creates 250 frames per second, or 1/4 frames per millisecond, then $G = 1/4$. In this case $S = G \times e^{-2G} = 0.152$ (15.2 percent). This means that the throughput is $250 \times 0.152 = 38$. Only 38 frames out of 250 will probably survive.

Slotted ALOHA

Pure ALOHA has a vulnerable time of $2 \times T_{fr}$. This is so because there is no rule that defines when the station can send. A station may send soon after another station has started or just before another station has finished. Slotted ALOHA was invented to improve the efficiency of pure ALOHA.

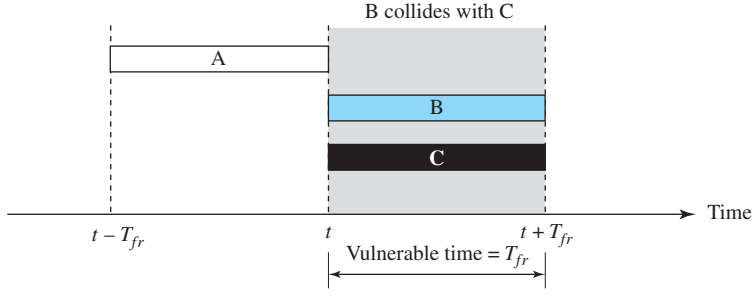
In **slotted ALOHA** we divide the time into slots of T_{fr} seconds and force the station to send only at the beginning of the time slot. Figure 12.5 shows an example of frame collisions in slotted ALOHA.

Figure 12.5 *Frames in a slotted ALOHA network*



Because a station is allowed to send only at the beginning of the synchronized time slot, if a station misses this moment, it must wait until the beginning of the next time slot. This means that the station which started at the beginning of this slot has already finished sending its frame. Of course, there is still the possibility of collision if two stations try to send at the beginning of the same time slot. However, the vulnerable time is now reduced to one-half, equal to T_{fr}. Figure 12.6 shows the situation.

Figure 12.6 *Vulnerable time for slotted ALOHA protocol*



Slotted ALOHA vulnerable time = T_{fr}

Throughput

It can be proven that the average number of successful transmissions for slotted ALOHA is $S = G \times e^{-G}$. The maximum throughput S_{max} is 0.368, when $G = 1$. In other words, if one frame is generated during one frame transmission time, then 36.8 percent of these frames reach their destination successfully. We expect $G = 1$ to produce maximum throughput because the vulnerable time is equal to the frame transmission time. Therefore, if a station generates only one frame in this vulnerable time (and no other station generates a frame during this time), the frame will reach its destination successfully.

The throughput for slotted ALOHA is $S = G \times e^{-G}$.
The maximum throughput $S_{max} = 0.368$ when $G = 1$.

Example 12.4

A slotted ALOHA network transmits 200-bit frames using a shared channel with a 200-kbps bandwidth. Find the throughput if the system (all stations together) produces

a. 1000 frames per second.

b. 500 frames per second.

c. 250 frames per second.

Solution

This situation is similar to the previous exercise except that the network is using slotted ALOHA instead of pure ALOHA. The frame transmission time is 200/200 kbps or 1 ms.

a. In this case G is 1. So $S = G \times e^{-G} = 0.368$ (36.8 percent). This means that the throughput is $1000 \times 0.0368 = 368$ frames. Only 368 out of 1000 frames will probably survive. Note that this is the maximum throughput case, percentagewise.

b. Here G is 1/2. In this case $S = G \times e^{-G} = 0.303$ (30.3 percent). This means that the throughput is $500 \times 0.0303 = 151$. Only 151 frames out of 500 will probably survive.

c. Now G is 1/4. In this case $S = G \times e^{-G} = 0.195$ (19.5 percent). This means that the throughput is $250 \times 0.195 = 49$. Only 49 frames out of 250 will probably survive.

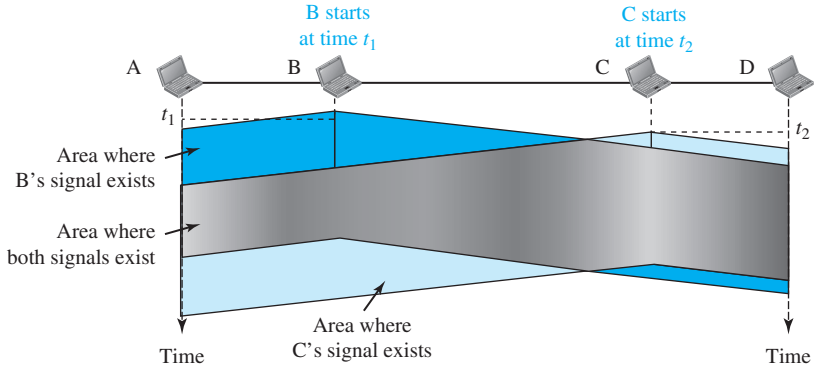
12.1.2 CSMA

To minimize the chance of collision and, therefore, increase the performance, the CSMA method was developed. The chance of collision can be reduced if a station senses the medium before trying to use it. **Carrier sense multiple access (CSMA)** requires that each station first listen to the medium (or check the state of the medium) before sending. In other words, CSMA is based on the principle "sense before transmit" or "listen before talk."

CSMA can reduce the possibility of collision, but it cannot eliminate it. The reason for this is shown in Figure 12.7, a space and time model of a CSMA network. Stations are connected to a shared channel (usually a dedicated medium).

The possibility of collision still exists because of propagation delay; when a station sends a frame, it still takes time (although very short) for the first bit to reach every station and for every station to sense it. In other words, a station may sense the medium and find it idle, only because the first bit sent by another station has not yet been received.

Figure 12.7 *Space/time model of a collision in CSMA*

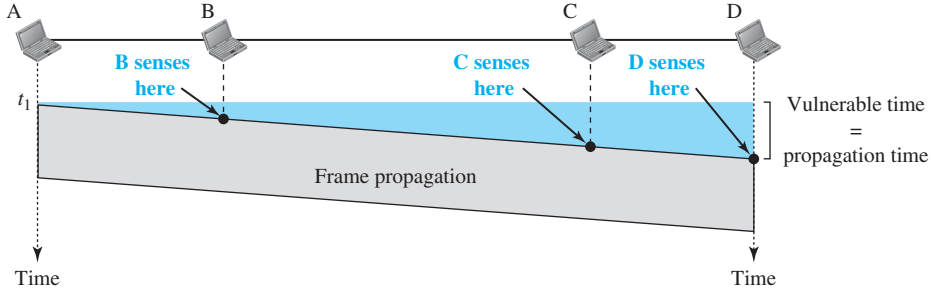


At time t_1, station B senses the medium and finds it idle, so it sends a frame. At time t_2 ($t_2 > t_1$), station C senses the medium and finds it idle because, at this time, the first bits from station B have not reached station C. Station C also sends a frame. The two signals collide and both frames are destroyed.

Vulnerable Time

The vulnerable time for CSMA is the ***propagation time*** T_p. This is the time needed for a signal to propagate from one end of the medium to the other. When a station sends a frame and any other station tries to send a frame during this time, a collision will result. But if the first bit of the frame reaches the end of the medium, every station will already have heard the bit and will refrain from sending. Figure 12.8 shows the worst case. The leftmost station, A, sends a frame at time t_1, which reaches the rightmost station, D, at time $t_1 + T_p$. The gray area shows the vulnerable area in time and space.

Figure 12.8 *Vulnerable time in CSMA*



Persistence Methods

What should a station do if the channel is busy? What should a station do if the channel is idle? Three methods have been devised to answer these questions: the **1-persistent method,** the **nonpersistent method,** and the ***p*-persistent method.** Figure 12.9 shows the behavior of three persistence methods when a station finds a channel busy.

Figure 12.9 *Behavior of three persistence methods*

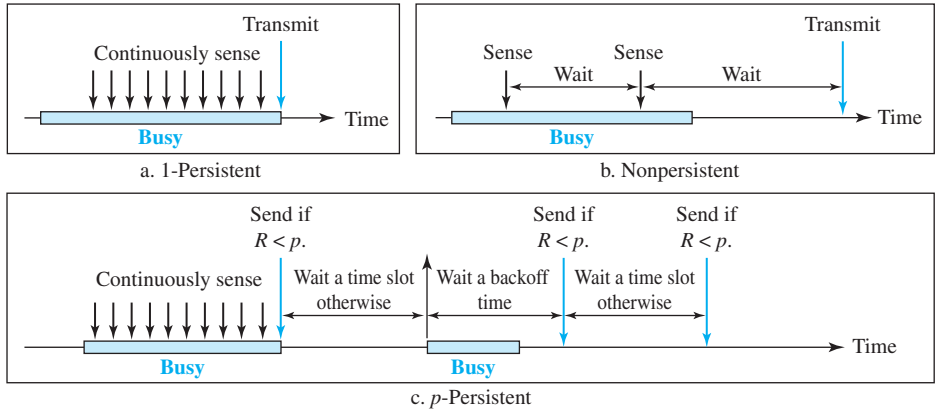


Figure 12.10 shows the flow diagrams for these methods.

1-Persistent

The *1-persistent method* is simple and straightforward. In this method, after the station finds the line idle, it sends its frame immediately (with probability 1). This method has the highest chance of collision because two or more stations may find the line idle and send their frames immediately. We will see later that Ethernet uses this method.

Nonpersistent

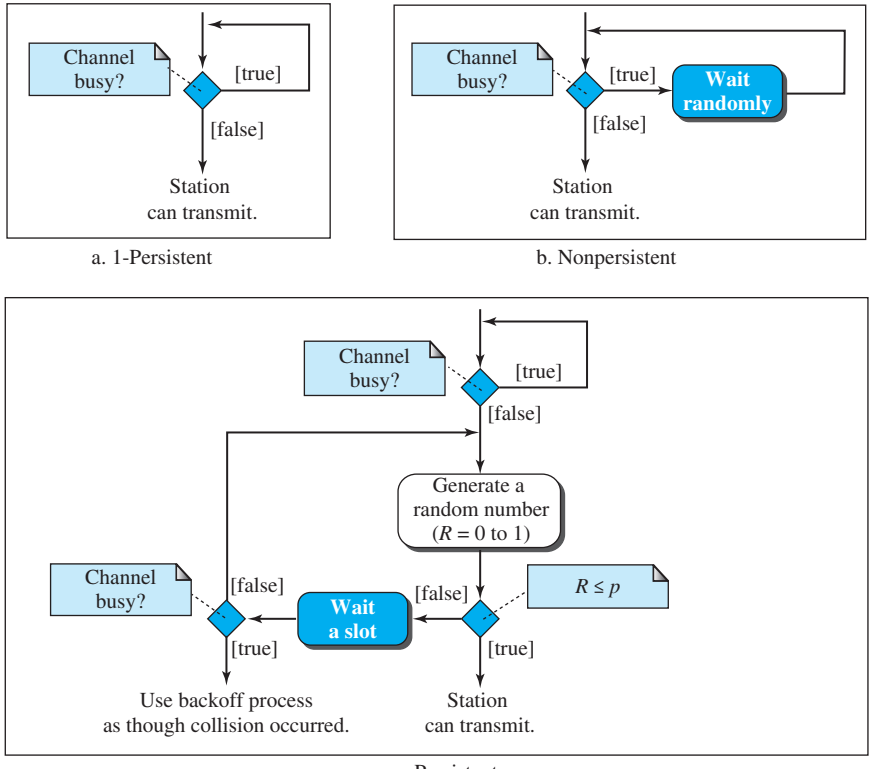
In the *nonpersistent method,* a station that has a frame to send senses the line. If the line is idle, it sends immediately. If the line is not idle, it waits a random amount of time and then senses the line again. The nonpersistent approach reduces the chance of collision because it is unlikely that two or more stations will wait the same amount of time and retry to send simultaneously. However, this method reduces the efficiency of the network because the medium remains idle when there may be stations with frames to send.

p-*Persistent*

The p-*persistent method* is used if the channel has time slots with a slot duration equal to or greater than the maximum propagation time. The *p*-persistent approach combines the advantages of the other two strategies. It reduces the chance of collision and improves efficiency. In this method, after the station finds the line idle it follows these steps:

1. With probability *p*, the station sends its frame.

Figure 12.10 *Flow diagram for three persistence methods*



2. With probability $q = 1 - p$, the station waits for the beginning of the next time slot and checks the line again.
   a. If the line is idle, it goes to step 1.
   b. If the line is busy, it acts as though a collision has occurred and uses the back-off procedure.

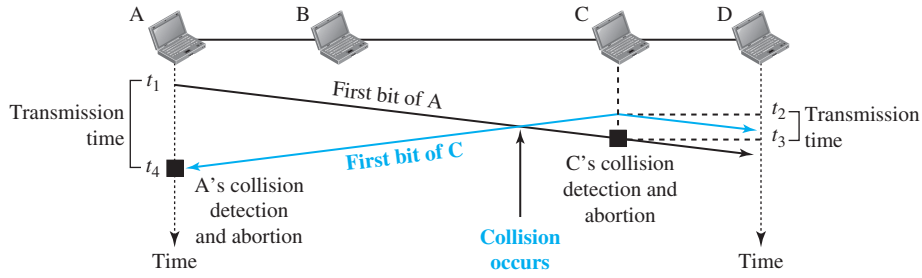
12.1.3 CSMA/CD

The CSMA method does not specify the procedure following a collision. **Carrier sense multiple access with collision detection (CSMA/CD)** augments the algorithm to handle the collision.

In this method, a station monitors the medium after it sends a frame to see if the transmission was successful. If so, the station is finished. If, however, there is a collision, the frame is sent again.

To better understand CSMA/CD, let us look at the first bits transmitted by the two stations involved in the collision. Although each station continues to send bits in the frame until it detects the collision, we show what happens as the first bits collide. In Figure 12.11, stations A and C are involved in the collision.

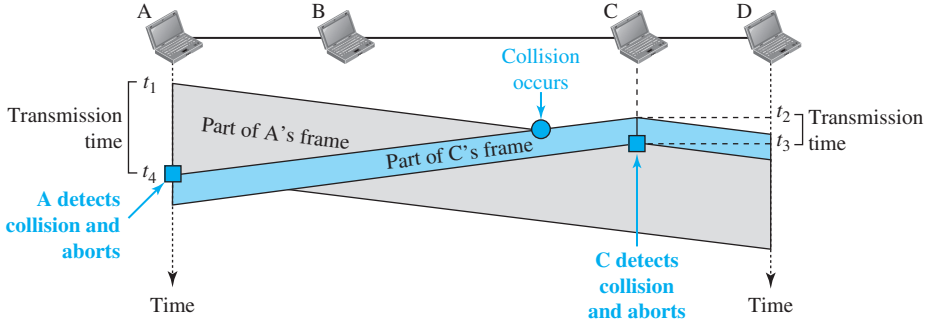
Figure 12.11 *Collision of the first bits in CSMA/CD*



At time t_1, station A has executed its persistence procedure and starts sending the bits of its frame. At time t_2, station C has not yet sensed the first bit sent by A. Station C executes its persistence procedure and starts sending the bits in its frame, which propagate both to the left and to the right. The collision occurs sometime after time t_2. Station C detects a collision at time t_3 when it receives the first bit of A's frame. Station C immediately (or after a short time, but we assume immediately) aborts transmission. Station A detects collision at time t_4 when it receives the first bit of C's frame; it also immediately aborts transmission. Looking at the figure, we see that A transmits for the duration $t_4 - t_1$; C transmits for the duration $t_3 - t_2$.

Now that we know the time durations for the two transmissions, we can show a more complete graph in Figure 12.12.

Figure 12.12 *Collision and abortion in CSMA/CD*



Minimum Frame Size

For CSMA/CD to work, we need a restriction on the frame size. Before sending the last bit of the frame, the sending station must detect a collision, if any, and abort the transmission. This is so because the station, once the entire frame is sent, does not keep a copy of

the frame and does not monitor the line for collision detection. Therefore, the frame transmission time T_{fr} must be at least two times the maximum propagation time T_p. To understand the reason, let us think about the worst-case scenario. If the two stations involved in a collision are the maximum distance apart, the signal from the first takes time T_p to reach the second, and the effect of the collision takes another time T_P to reach the first. So the requirement is that the first station must still be transmitting after $2T_p$.

Example 12.5

A network using CSMA/CD has a bandwidth of 10 Mbps. If the maximum propagation time (including the delays in the devices and ignoring the time needed to send a jamming signal, as we see later) is 25.6 μs, what is the minimum size of the frame?

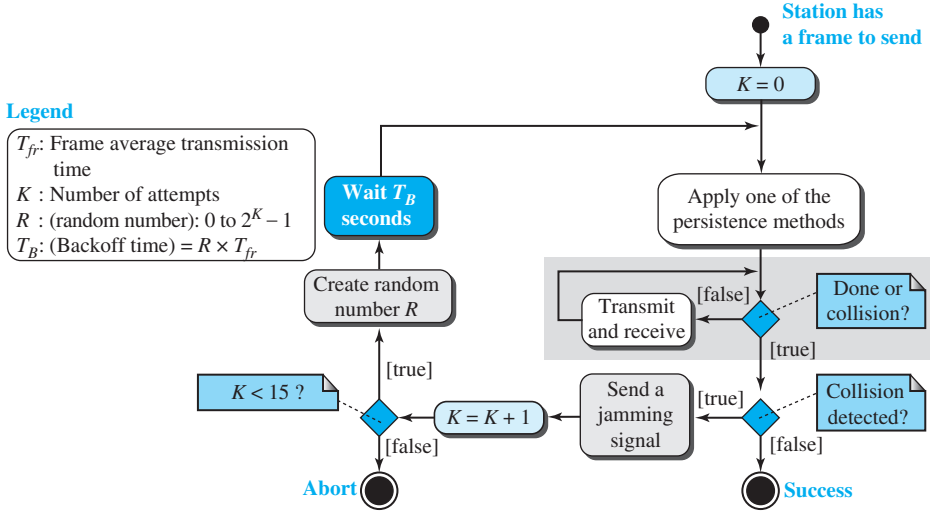
Solution

The minimum frame transmission time is $T_{fr} = 2 \times T_p = 51.2$ μs. This means, in the worst case, a station needs to transmit for a period of 51.2 μs to detect the collision. The minimum size of the frame is 10 Mbps × 51.2 μs = 512 bits or 64 bytes. This is actually the minimum size of the frame for Standard Ethernet, as we will see later in the chapter.

Procedure

Now let us look at the flow diagram for CSMA/CD in Figure 12.13. It is similar to the one for the ALOHA protocol, but there are differences.

Figure 12.13 *Flow diagram for the CSMA/CD*



The first difference is the addition of the persistence process. We need to sense the channel before we start sending the frame by using one of the persistence processes we discussed previously (nonpersistent, 1-persistent, or *p*-persistent). The corresponding box can be replaced by one of the persistence processes shown in Figure 12.10.

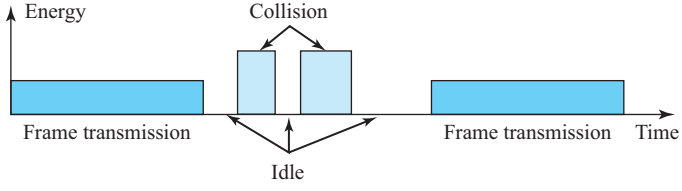
The second difference is the frame transmission. In ALOHA, we first transmit the entire frame and then wait for an acknowledgment. In CSMA/CD, transmission and collision detection are continuous processes. We do not send the entire frame and then look for a collision. The station transmits and receives continuously and simultaneously (using two different ports or a bidirectional port). We use a loop to show that transmission is a continuous process. We constantly monitor in order to detect one of two conditions: either transmission is finished or a collision is detected. Either event stops transmission. When we come out of the loop, if a collision has not been detected, it means that transmission is complete; the entire frame is transmitted. Otherwise, a collision has occurred.

The third difference is the sending of a short **jamming signal** to make sure that all other stations become aware of the collision.

Energy Level

We can say that the level of energy in a channel can have three values: zero, normal, and abnormal. At the zero level, the channel is idle. At the normal level, a station has successfully captured the channel and is sending its frame. At the abnormal level, there is a collision and the level of the energy is twice the normal level. A station that has a frame to send or is sending a frame needs to monitor the energy level to determine if the channel is idle, busy, or in collision mode. Figure 12.14 shows the situation.

Figure 12.14 *Energy level during transmission, idleness, or collision*



Throughput

The throughput of CSMA/CD is greater than that of pure or slotted ALOHA. The maximum throughput occurs at a different value of G and is based on the persistence method and the value of p in the p-persistent approach. For the 1-persistent method, the maximum throughput is around 50 percent when $G = 1$. For the nonpersistent method, the maximum throughput can go up to 90 percent when G is between 3 and 8.

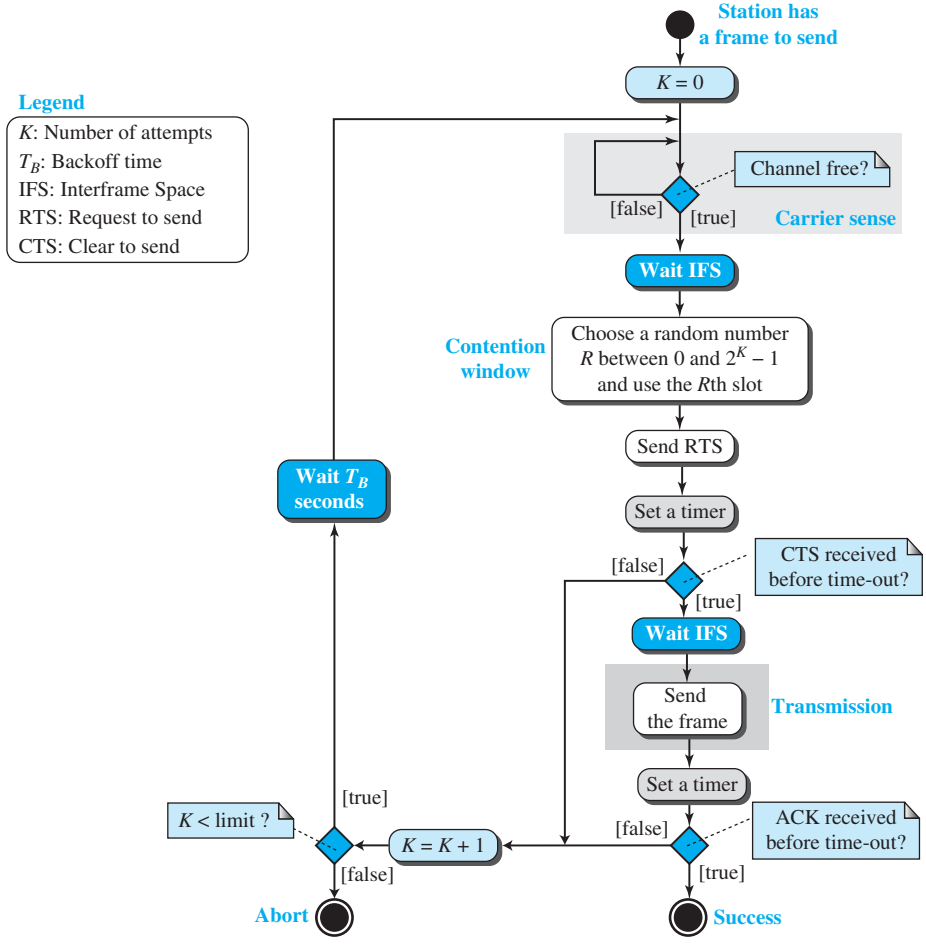
Traditional Ethernet

One of the LAN protocols that used CSMA/CD is the traditional Ethernet with the data rate of 10 Mbps. We discuss the Ethernet LANs in Chapter 13, but it is good to know that the traditional Ethernet was a broadcast LAN that used the 1-persistence method to control access to the common media. Later versions of Ethernet try to move from CSMA/CD access methods for the reason that we discuss in Chapter 13.

12.1.4 CSMA/CA

Carrier sense multiple access with collision avoidance (CSMA/CA) was invented for wireless networks. Collisions are avoided through the use of CSMA/CA's three strategies: the interframe space, the contention window, and acknowledgments, as shown in Figure 12.15. We discuss RTS and CTS frames later.

Figure 12.15 *Flow diagram of CSMA/CA*

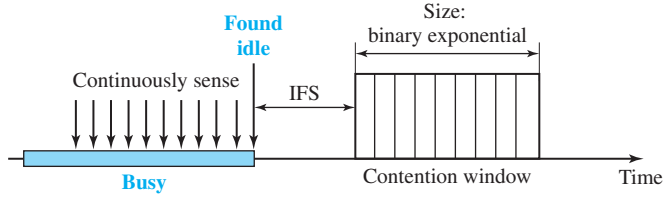


- ***Interframe Space (IFS).*** First, collisions are avoided by deferring transmission even if the channel is found idle. When an idle channel is found, the station does not send immediately. It waits for a period of time called the ***interframe space*** or ***IFS.*** Even though the channel may appear idle when it is sensed, a distant station may have already started transmitting. The distant station's signal has not yet reached this

station. The IFS time allows the front of the transmitted signal by the distant station to reach this station. After waiting an IFS time, if the channel is still idle, the station can send, but it still needs to wait a time equal to the contention window (described next). The IFS variable can also be used to prioritize stations or frame types. For example, a station that is assigned a shorter IFS has a higher priority.

- ***Contention Window.*** The **contention window** is an amount of time divided into slots. A station that is ready to send chooses a random number of slots as its wait time. The number of slots in the window changes according to the binary exponential backoff strategy. This means that it is set to one slot the first time and then doubles each time the station cannot detect an idle channel after the IFS time. This is very similar to the *p*-persistent method except that a random outcome defines the number of slots taken by the waiting station. One interesting point about the contention window is that the station needs to sense the channel after each time slot. However, if the station finds the channel busy, it does not restart the process; it just stops the timer and restarts it when the channel is sensed as idle. This gives priority to the station with the longest waiting time. See Figure 12.16.

Figure 12.16 *Contention window*



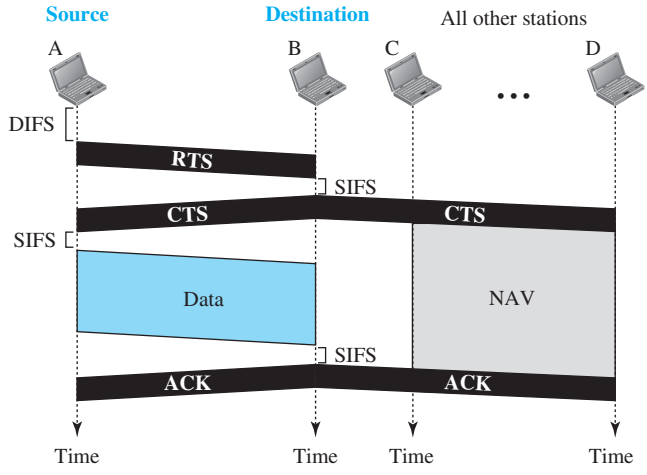
- ***Acknowledgment.*** With all these precautions, there still may be a collision resulting in destroyed data. In addition, the data may be corrupted during the transmission. The positive acknowledgment and the time-out timer can help guarantee that the receiver has received the frame.

Frame Exchange Time Line

Figure 12.17 shows the exchange of data and control frames in time.

1. Before sending a frame, the source station senses the medium by checking the energy level at the carrier frequency.
   a. The channel uses a persistence strategy with backoff until the channel is idle.
   b. After the station is found to be idle, the station waits for a period of time called the ***DCF interframe space (DIFS);*** then the station sends a control frame called the *request to send (RTS)*.
2. After receiving the RTS and waiting a period of time called the ***short interframe space (SIFS),*** the destination station sends a control frame, called the *clear to send (CTS),* to the source station. This control frame indicates that the destination station is ready to receive data.

Figure 12.17 *CSMA/CA and NAV*



3. The source station sends data after waiting an amount of time equal to SIFS.
4. The destination station, after waiting an amount of time equal to SIFS, sends an acknowledgment to show that the frame has been received. Acknowledgment is needed in this protocol because the station does not have any means to check for the successful arrival of its data at the destination. On the other hand, the lack of collision in CSMA/CD is a kind of indication to the source that data have arrived.

Network Allocation Vector

How do other stations defer sending their data if one station acquires access? In other words, how is the *collision avoidance* aspect of this protocol accomplished*?* The key is a feature called *NAV.*

When a station sends an RTS frame, it includes the duration of time that it needs to occupy the channel. The stations that are affected by this transmission create a timer called a **network allocation vector (NAV)** that shows how much time must pass before these stations are allowed to check the channel for idleness. Each time a station accesses the system and sends an RTS frame, other stations start their NAV. In other words, each station, before sensing the physical medium to see if it is idle, first checks its NAV to see if it has expired. Figure 12.17 shows the idea of NAV.

Collision During Handshaking

What happens if there is a collision during the time when RTS or CTS control frames are in transition, often called the *handshaking period?* Two or more stations may try to send RTS frames at the same time. These control frames may collide. However, because there is no mechanism for collision detection, the sender assumes there has been a collision if it has not received a CTS frame from the receiver. The backoff strategy is employed, and the sender tries again.

Hidden-Station Problem

The solution to the hidden station problem is the use of the handshake frames (RTS and CTS). Figure 12.17 also shows that the RTS message from B reaches A, but not C. However, because both B and C are within the range of A, the CTS message, which contains the duration of data transmission from B to A, reaches C. Station C knows that some hidden station is using the channel and refrains from transmitting until that duration is over.

CSMA/CA and Wireless Networks

CSMA/CA was mostly intended for use in wireless networks. The procedure described above, however, is not sophisticated enough to handle some particular issues related to wireless networks, such as hidden terminals or exposed terminals. We will see how these issues are solved by augmenting the above protocol with handshaking features. The use of CSMA/CA in wireless networks will be discussed in Chapter 15.

12.2 CONTROLLED ACCESS

In **controlled access,** the stations consult one another to find which station has the right to send. A station cannot send unless it has been authorized by other stations. We discuss three controlled-access methods.

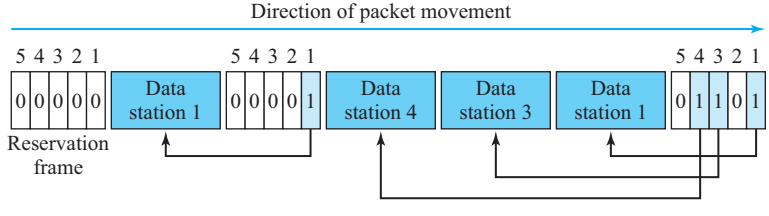
12.2.1 Reservation

In the **reservation** method, a station needs to make a reservation before sending data. Time is divided into intervals. In each interval, a reservation frame precedes the data frames sent in that interval.

If there are *N* stations in the system, there are exactly *N* reservation minislots in the reservation frame. Each minislot belongs to a station. When a station needs to send a data frame, it makes a reservation in its own minislot. The stations that have made reservations can send their data frames after the reservation frame.

Figure 12.18 shows a situation with five stations and a five-minislot reservation frame. In the first interval, only stations 1, 3, and 4 have made reservations. In the second interval, only station 1 has made a reservation.

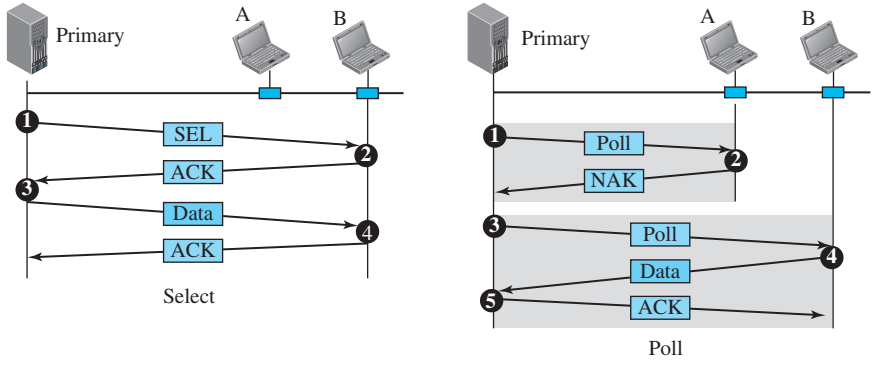
Figure 12.18 *Reservation access method*



12.2.2 Polling

Polling works with topologies in which one device is designated as a ***primary station*** and the other devices are ***secondary stations.*** All data exchanges must be made through the primary device even when the ultimate destination is a secondary device. The primary device controls the link; the secondary devices follow its instructions. It is up to the primary device to determine which device is allowed to use the channel at a given time. The primary device, therefore, is always the initiator of a session (see Figure 12.19). This method uses poll and select functions to prevent collisions. However, the drawback is if the primary station fails, the system goes down.

Figure 12.19 *Select and poll functions in polling-access method*



Select

The *select* function is used whenever the primary device has something to send. Remember that the primary controls the link. If the primary is neither sending nor receiving data, it knows the link is available. If it has something to send, the primary device sends it. What it does not know, however, is whether the target device is prepared to receive. So the primary must alert the secondary to the upcoming transmission and wait for an acknowledgment of the secondary's ready status. Before sending data, the primary creates and transmits a select (SEL) frame, one field of which includes the address of the intended secondary.

Poll

The *poll* function is used by the primary device to solicit transmissions from the secondary devices. When the primary is ready to receive data, it must ask (poll) each device in turn if it has anything to send. When the first secondary is approached, it responds either with a NAK frame if it has nothing to send or with data (in the form of a data frame) if it does. If the response is negative (a NAK frame), then the primary polls the next secondary in the same manner until it finds one with data to send. When the response is positive (a data frame), the primary reads the frame and returns an acknowledgment (ACK frame), verifying its receipt.

12.2.3 Token Passing

In the **token-passing** method, the stations in a network are organized in a logical ring. In other words, for each station, there is a *predecessor* and a *successor*. The predecessor is the station which is logically before the station in the ring; the successor is the station which is after the station in the ring. The current station is the one that is accessing the channel now. The right to this access has been passed from the predecessor to the current station. The right will be passed to the successor when the current station has no more data to send.

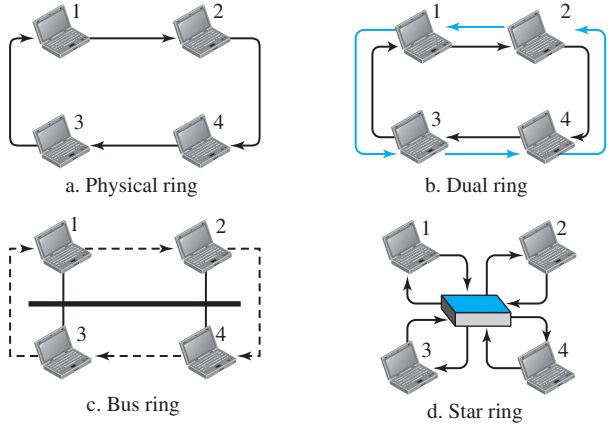
But how is the right to access the channel passed from one station to another? In this method, a special packet called a ***token*** circulates through the ring. The possession of the token gives the station the right to access the channel and send its data. When a station has some data to send, it waits until it receives the token from its predecessor. It then holds the token and sends its data. When the station has no more data to send, it releases the token, passing it to the next logical station in the ring. The station cannot send data until it receives the token again in the next round. In this process, when a station receives the token and has no data to send, it just passes the data to the next station.

Token management is needed for this access method. Stations must be limited in the time they can have possession of the token. The token must be monitored to ensure it has not been lost or destroyed. For example, if a station that is holding the token fails, the token will disappear from the network. Another function of token management is to assign priorities to the stations and to the types of data being transmitted. And finally, token management is needed to make low-priority stations release the token to high-priority stations.

Logical Ring

In a token-passing network, stations do not have to be physically connected in a ring; the ring can be a logical one. Figure 12.20 shows four different physical topologies that can create a logical ring.

Figure 12.20 *Logical ring and physical topology in token-passing access method*



In the physical ring topology, when a station sends the token to its successor, the token cannot be seen by other stations; the successor is the next one in line. This means that the token does not have to have the address of the next successor. The problem with this topology is that if one of the links—the medium between two adjacent stations—fails, the whole system fails.

The dual ring topology uses a second (auxiliary) ring which operates in the reverse direction compared with the main ring. The second ring is for emergencies only (such as a spare tire for a car). If one of the links in the main ring fails, the system automatically combines the two rings to form a temporary ring. After the failed link is restored, the auxiliary ring becomes idle again. Note that for this topology to work, each station needs to have two transmitter ports and two receiver ports. The high-speed Token Ring networks called *FDDI (Fiber Distributed Data Interface)* and *CDDI (Copper Distributed Data Interface)* use this topology.

In the bus ring topology, also called a token bus, the stations are connected to a single cable called a *bus.* They, however, make a logical ring, because each station knows the address of its successor (and also predecessor for token management purposes). When a station has finished sending its data, it releases the token and inserts the address of its successor in the token. Only the station with the address matching the destination address of the token gets the token to access the shared media. The Token Bus LAN, standardized by IEEE, uses this topology.

In a star ring topology, the physical topology is a star. There is a hub, however, that acts as the connector. The wiring inside the hub makes the ring; the stations are connected to this ring through the two wire connections. This topology makes the network less prone to failure because if a link goes down, it will be bypassed by the hub and the rest of the stations can operate. Also adding and removing stations from the ring is easier. This topology is still used in the Token Ring LAN designed by IBM.

12.3 CHANNELIZATION

Channelization (or *channel partition,* as it is sometimes called) is a multiple-access method in which the available bandwidth of a link is shared in time, frequency, or through code, among different stations. In this section, we discuss three channelization protocols: FDMA, TDMA, and CDMA.

We see the application of all these methods in Chapter 16 when we discuss cellular phone systems.

12.3.1 FDMA

In **frequency-division multiple access (FDMA),** the available bandwidth is divided into frequency bands. Each station is allocated a band to send its data. In other words, each band is reserved for a specific station, and it belongs to the station all the time. Each station also uses a bandpass filter to confine the transmitter frequencies. To prevent

CHAPTER 15

Wireless LANs

We discussed wired LANs and wired WANs in the two previous chapters. We concentrate on wireless LANs in this chapter and wireless WANs in the next.

In this chapter, we cover two types of wireless LANs. The first is the wireless LAN defined by the IEEE 802.11 project (sometimes called *wireless Ethernet*); the second is a personal wireless LAN, Bluetooth, that is sometimes called *personal area network* or *PAN*.

This chapter is divided into three sections:

- The first section introduces the general issues behind wireless LANs and compares wired and wireless networks. The section describes the characteristics of the wireless networks and the way access is controlled in these types of networks.
- The second section discusses a wireless LAN defined by the IEEE 802.11 Project, which is sometimes called *wireless Ethernet*. This section defines the architecture of this type of LAN and describes the MAC sublayer, which uses the CSMA/CA access method discussed in Chapter 12. The section then shows the addressing mechanism used in this network and gives the format of different packets used at the data-link layer. Finally, the section discusses different physical-layer protocols that are used by this type of network.
- The third section discusses the Bluetooth technology as a personal area network (PAN). The section describes the architecture of the network, the addressing mechanism, and the packet format. Different layers used in this protocol are also briefly described and compared with the ones in the other wired and wireless LANs.

15.1 INTRODUCTION

Wireless communication is one of the fastest-growing technologies. The demand for connecting devices without the use of cables is increasing everywhere. Wireless LANs can be found on college campuses, in office buildings, and in many public areas. Before we discuss a specific protocol related to wireless LANs, let us talk about them in general.

15.1.1 Architectural Comparison

Let us first compare the architecture of wired and wireless LANs to give some idea of what we need to look for when we study wireless LANs.

Medium

The first difference we can see between a wired and a wireless LAN is the medium. In a wired LAN, we use wires to connect hosts. In Chapter 7, we saw that we moved from multiple access to point-to-point access through the generation of the Ethernet. In a switched LAN, with a link-layer switch, the communication between the hosts is point-to-point and full-duplex (bidirectional). In a wireless LAN, the medium is air, the signal is generally broadcast. When hosts in a wireless LAN communicate with each other, they are sharing the same medium (multiple access). In a very rare situation, we may be able to create a point-to-point communication between two wireless hosts by using a very limited bandwidth and two-directional antennas. Our discussion in this chapter, however, is about the multiple-access medium, which means we need to use MAC protocols.

Hosts

In a wired LAN, a host is always connected to its network at a point with a fixed link-layer address related to its network interface card (NIC). Of course, a host can move from one point in the Internet to another point. In this case, its link-layer address remains the same, but its network-layer address will change, as we see later in Chapter 19, Section 19.3 (Mobile IP section). However, before the host can use the services of the Internet, it needs to be physically connected to the Internet. In a wireless LAN, a host is not physically connected to the network; it can move freely (as we'll see) and can use the services provided by the network. Therefore, mobility in a wired network and wireless network are totally different issues, which we try to clarify in this chapter.

Isolated LANs

The concept of a wired isolated LAN also differs from that of a wireless isolated LAN. A wired isolated LAN is a set of hosts connected via a link-layer switch (in the recent generation of Ethernet). A wireless isolated LAN, called an ***ad hoc network*** in wireless LAN terminology, is a set of hosts that communicate freely with each other. The concept of a link-layer switch does not exist in wireless LANs. Figure 15.1 shows two isolated LANs, one wired and one wireless.

Connection to Other Networks

A wired LAN can be connected to another network or an internetwork such as the Internet using a router. A wireless LAN may be connected to a wired infrastructure network,

to a wireless infrastructure network, or to another wireless LAN. The first situation is the one that we discuss in this section: connection of a wireless LAN to a wired infrastructure network. Figure 15.2 shows the two environments.

Figure 15.1 *Isolated LANs: wired versus wireless*

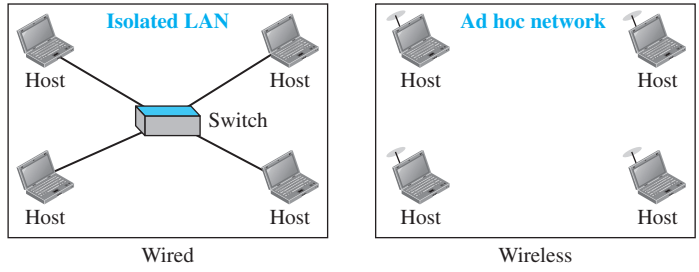
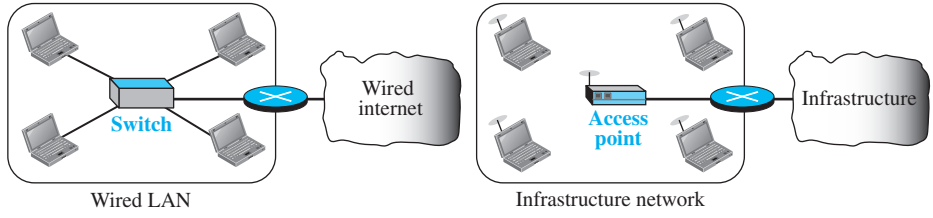


Figure 15.2 *Connection of a wired LAN and a wireless LAN to other networks*



In this case, the wireless LAN is referred to as an *infrastructure network,* and the connection to the wired infrastructure, such as the Internet, is done via a device called an ***access point (AP)***. Note that the role of the access point is completely different from the role of a link-layer switch in the wired environment. An access point is gluing two different environments together: one wired and one wireless. Communication between the AP and the wireless host occurs in a wireless environment; communication between the AP and the infrastructure occurs in a wired environment.

Moving between Environments

The discussion above confirms what we learned in Chapters 2 and 9: a wired LAN or a wireless LAN operates only in the lower two layers of the TCP/IP protocol suite. This means that if we have a wired LAN in a building that is connected via a router or a modem to the Internet, all we need in order to move from the wired environment to a wireless environment is to change the network interface cards designed for wired environments to the ones designed for wireless environments and replace the link-layer switch with an access point. In this change, the link-layer addresses will change (because of changing NICs), but the network-layer addresses (IP addresses) will remain the same; we are moving from wired links to wireless links.

15.1.2 Characteristics

There are several characteristics of wireless LANs that either do not apply to wired LANs or the existence of which is negligible and can be ignored. We discuss some of these characteristics here to pave the way for discussing wireless LAN protocols.

Attenuation

The strength of electromagnetic signals decreases rapidly because the signal disperses in all directions; only a small portion of it reaches the receiver. The situation becomes worse with mobile senders that operate on batteries and normally have small power supplies.

Interference

Another issue is that a receiver may receive signals not only from the intended sender, but also from other senders if they are using the same frequency band.

Multipath Propagation

A receiver may receive more than one signal from the same sender because electromagnetic waves can be reflected back from obstacles such as walls, the ground, or objects. The result is that the receiver receives some signals at different phases (because they travel different paths). This makes the signal less recognizable.

Error

With the above characteristics of a wireless network, we can expect that errors and error detection are more serious issues in a wireless network than in a wired network. If we think about the error level as the measurement of **signal-to-noise ratio (SNR),** we can better understand why error detection and error correction and retransmission are more important in a wireless network. We discussed SNR in more detail in Chapter 3, but it is enough to say that it measures the ratio of good stuff to bad stuff (signal to noise). If SNR is high, it means that the signal is stronger than the noise (unwanted signal), so we may be able to convert the signal to actual data. On the other hand, when SNR is low, it means that the signal is corrupted by the noise and the data cannot be recovered.

15.1.3 Access Control

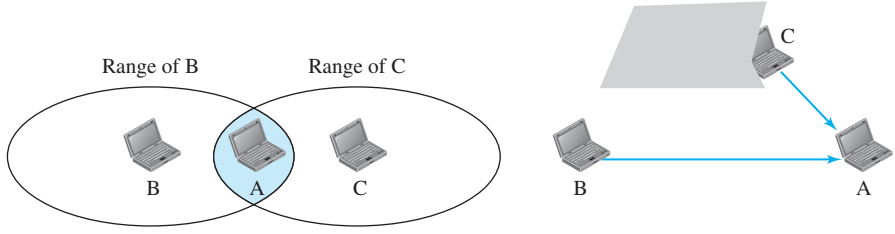
Maybe the most important issue we need to discuss in a wireless LAN is access control—how a wireless host can get access to the shared medium (air). We discussed in Chapter 12 that the Standard Ethernet uses the CSMA/CD algorithm. In this method, each host contends to access the medium and sends its frame if it finds the medium idle. If a collision occurs, it is detected and the frame is sent again. Collision detection in CSMA/CD serves two purposes. If a collision is detected, it means that the frame has not been received and needs to be resent. If a collision is not detected, it is a kind of acknowledgment that the frame was received. The CSMA/CD algorithm does not work in wireless LANs for three reasons:

1. To detect a collision, a host needs to send and receive at the same time (sending the frame and receiving the collision signal), which means the host needs to work in a

duplex mode. Wireless hosts do not have enough power to do so (the power is supplied by batteries). They can only send or receive at one time.

2. Because of the hidden station problem, in which a station may not be aware of another station's transmission due to some obstacles or range problems, collision may occur but not be detected. Figure 15.3 shows an example of the hidden station problem. Station B has a transmission range shown by the left oval (sphere in space); every station in this range can hear any signal transmitted by station B. Station C has a transmission range shown by the right oval (sphere in space); every station located in this range can hear any signal transmitted by C. Station C is

Figure 15.3 *Hidden station problem*



outside the transmission range of B; likewise, station B is outside the transmission range of C. Station A, however, is in the area covered by both B and C; it can hear any signal transmitted by B or C. The figure also shows that the hidden station problem may also occur due to an obstacle.

Assume that station B is sending data to station A. In the middle of this transmission, station C also has data to send to station A. However, station C is out of B's range and transmissions from B cannot reach C. Therefore C thinks the medium is free. Station C sends its data to A, which results in a collision at A because this station is receiving data from both B and C. In this case, we say that stations B and C are hidden from each other with respect to A. Hidden stations can reduce the capacity of the network because of the possibility of collision.

3. The distance between stations can be great. Signal fading could prevent a station at one end from hearing a collision at the other end.

To overcome the above three problems, Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) was invented for wireless LANs, which we discussed in Chapter 12.

15.2 IEEE 802.11 PROJECT

IEEE has defined the specifications for a wireless LAN, called IEEE 802.11, which covers the physical and data-link layers. It is sometimes called *wireless Ethernet*. In

some countries, including the United States, the public uses the term *WiFi* (short for wireless fidelity) as a synonym for *wireless LAN*. WiFi, however, is a wireless LAN that is certified by the WiFi Alliance, a global, nonprofit industry association of more than 300 member companies devoted to promoting the growth of wireless LANs.

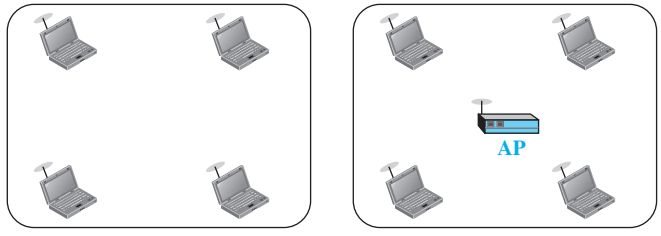
15.2.1 Architecture

The standard defines two kinds of services: the basic service set (BSS) and the extended service set (ESS).

Basic Service Set

IEEE 802.11 defines the **basic service set (BSS)** as the building blocks of a wireless LAN. A basic service set is made of stationary or mobile wireless stations and an optional central base station, known as the *access point (AP)*. Figure 15.4 shows two sets in this standard.

Figure 15.4 *Basic service sets (BSSs)*



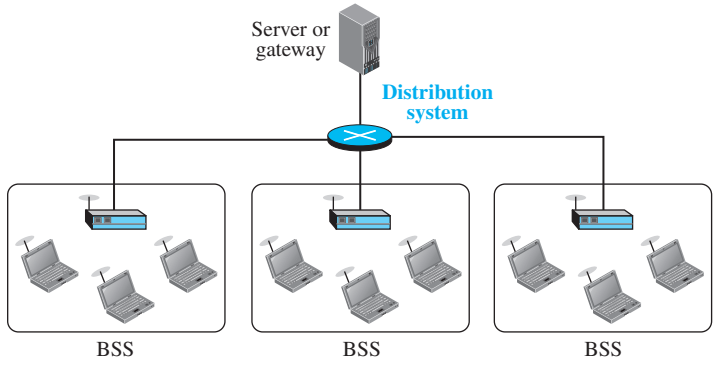
The BSS without an AP is a stand-alone network and cannot send data to other BSSs. It is called an *ad hoc architecture*. In this architecture, stations can form a network without the need of an AP; they can locate one another and agree to be part of a BSS. A BSS with an AP is sometimes referred to as an *infrastructure BSS*.

Extended Service Set

An **extended service set (ESS)** is made up of two or more BSSs with APs. In this case, the BSSs are connected through a *distribution system,* which is a wired or a wireless network. The distribution system connects the APs in the BSSs. IEEE 802.11 does not restrict the distribution system; it can be any IEEE LAN such as an Ethernet. Note that the extended service set uses two types of stations: mobile and stationary. The mobile stations are normal stations inside a BSS. The stationary stations are AP stations that are part of a wired LAN. Figure 15.5 shows an ESS.

When BSSs are connected, the stations within reach of one another can communicate without the use of an AP. However, communication between a station in a BSS and the outside BSS occurs via the AP. The idea is similar to communication in a cellular network (discussed in Chapter 16) if we consider each BSS to be a cell and each AP to be a base station. Note that a mobile station can belong to more than one BSS at the same time.

Figure 15.5 *Extended service set (ESS)*



Station Types

IEEE 802.11 defines three types of stations based on their mobility in a wireless LAN: no-transition, BSS-transition, and ESS-transition mobility. A station with **no-transition mobility** is either stationary (not moving) or moving only inside a BSS. A station with **BSS-transition mobility** can move from one BSS to another, but the movement is confined inside one ESS. A station with **ESS-transition mobility** can move from one ESS to another. However, IEEE 802.11 does not guarantee that communication is continuous during the move.

15.2.2 MAC Sublayer

IEEE 802.11 defines two MAC sublayers: the distributed coordination function (DCF) and point coordination function (PCF). Figure 15.6 shows the relationship between the two MAC sublayers, the LLC sublayer, and the physical layer. We discuss the physical layer implementations later in the chapter and will now concentrate on the MAC sublayer.

Distributed Coordination Function

One of the two protocols defined by IEEE at the MAC sublayer is called the ***distributed coordination function (DCF)***. DCF uses CSMA/CA as the access method (see Chapter 12).

Frame Exchange Time Line

Figure 15.7 shows the exchange of data and control frames in time.

1. Before sending a frame, the source station senses the medium by checking the energy level at the carrier frequency.
   a. The channel uses a persistence strategy with backoff until the channel is idle.
   b. After the station is found to be idle, the station waits for a period of time called the ***distributed interframe space (DIFS);*** then the station sends a control frame called the *request to send (RTS)*.

Figure 15.6 *MAC layers in IEEE 802.11 standard*

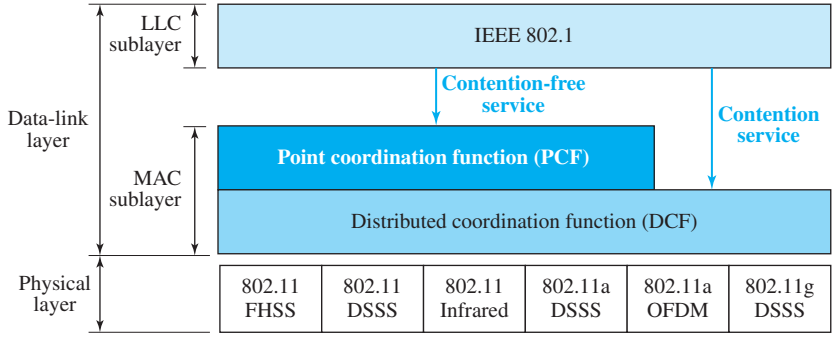
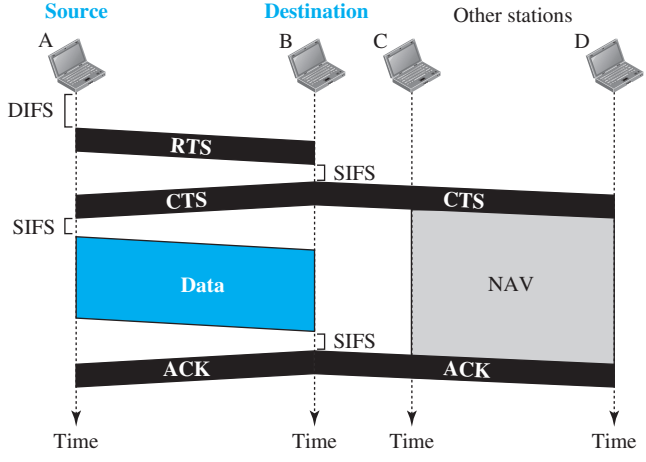


Figure 15.7 *CSMA/CA and NAV*



2. After receiving the RTS and waiting a period of time called the ***short interframe space (SIFS),*** the destination station sends a control frame, called the *clear to send (CTS),* to the source station. This control frame indicates that the destination station is ready to receive data.
3. The source station sends data after waiting an amount of time equal to SIFS.
4. The destination station, after waiting an amount of time equal to SIFS, sends an acknowledgment to show that the frame has been received. Acknowledgment is needed in this protocol because the station does not have any means to check for the successful arrival of its data at the destination. On the other hand, the lack of collision in CSMA/CD is a kind of indication to the source that data have arrived.

Network Allocation Vector

How do other stations defer sending their data if one station acquires access? In other words, how is the *collision avoidance* aspect of this protocol accomplished? The key is a feature called NAV.

When a station sends an RTS frame, it includes the duration of time that it needs to occupy the channel. The stations that are affected by this transmission create a timer called a ***network allocation vector (NAV)*** that shows how much time must pass before these stations are allowed to check the channel for idleness. Each time a station accesses the system and sends an RTS frame, other stations start their NAV. In other words, each station, before sensing the physical medium to see if it is idle, first checks its NAV to see if it has expired. Figure 15.7 shows the idea of NAV.

Collision During Handshaking

What happens if there is a collision during the time when RTS or CTS control frames are in transition, often called the *handshaking period*? Two or more stations may try to send RTS frames at the same time. These control frames may collide. However, because there is no mechanism for collision detection, the sender assumes there has been a collision if it has not received a CTS frame from the receiver. The backoff strategy is employed, and the sender tries again.

Hidden-Station Problem

The solution to the hidden station problem is the use of the handshake frames (RTS and CTS). Figure 15.7 also shows that the RTS message from B reaches A, but not C. However, because both B and C are within the range of A, the CTS message, which contains the duration of data transmission from B to A, reaches C. Station C knows that some hidden station is using the channel and refrains from transmitting until that duration is over.

Point Coordination Function (PCF)

The **point coordination function (PCF)** is an optional access method that can be implemented in an infrastructure network (not in an ad hoc network). It is implemented on top of the DCF and is used mostly for time-sensitive transmission.

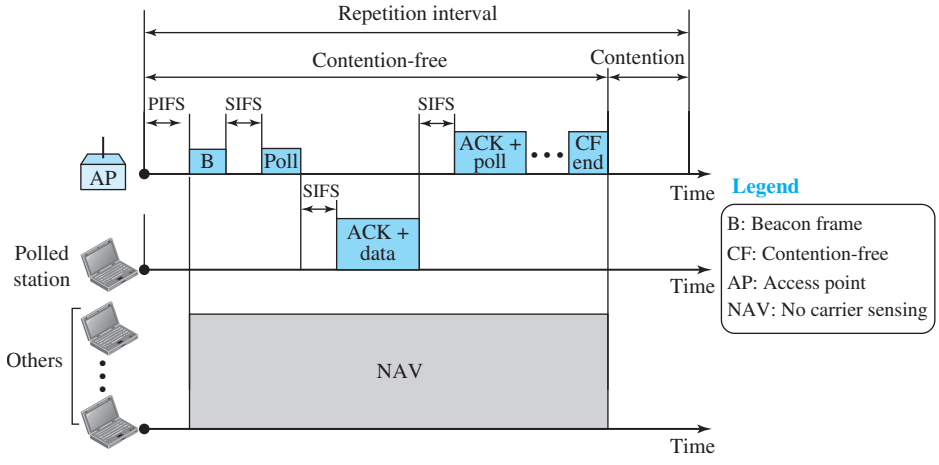
PCF has a centralized, contention-free polling access method, which we discussed in Chapter 12. The AP performs polling for stations that are capable of being polled. The stations are polled one after another, sending any data they have to the AP.

To give priority to PCF over DCF, another interframe space, PIFS, has been defined. PIFS (PCF IFS) is shorter than DIFS. This means that if, at the same time, a station wants to use only DCF and an AP wants to use PCF, the AP has priority.

Due to the priority of PCF over DCF, stations that only use DCF may not gain access to the medium. To prevent this, a repetition interval has been designed to cover both contention-free PCF and contention-based DCF traffic. The *repetition interval,* which is repeated continuously, starts with a special control frame, called a ***beacon frame*.** When the stations hear the beacon frame, they start their NAV for the duration of the contention-free period of the repetition interval. Figure 15.8 shows an example of a repetition interval.

During the repetition interval, the PC (point controller) can send a poll frame, receive data, send an ACK, receive an ACK, or do any combination of these (802.11

Figure 15.8 *Example of repetition interval*



uses piggybacking). At the end of the contention-free period, the PC sends a CF end (contention-free end) frame to allow the contention-based stations to use the medium.

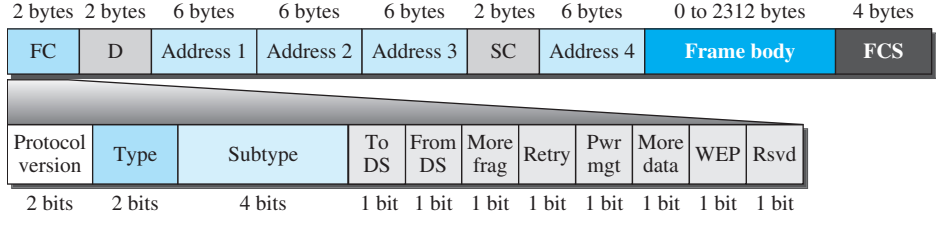
Fragmentation

The wireless environment is very noisy, so frames are often corrupted. A corrupt frame has to be retransmitted. The protocol, therefore, recommends fragmentation—the division of a large frame into smaller ones. It is more efficient to resend a small frame than a large one.

Frame Format

The MAC layer frame consists of nine fields, as shown in Figure 15.9.

Figure 15.9 *Frame format*



- ***Frame control (FC).*** The FC field is 2 bytes long and defines the type of frame and some control information. Table 15.1 describes the subfields. We will discuss each frame type later in this chapter.

Table 15.1 *Subfields in FC field*

Field	*Explanation*
Version	Current version is 0
Type	Type of information: management (00), control (01), or data (10)
Subtype	Subtype of each type (see Table 15.2)
To DS	Defined later
From DS	Defined later
More frag	When set to 1, means more fragments
Retry	When set to 1, means retransmitted frame
Pwr mgt	When set to 1, means station is in power management mode
More data	When set to 1, means station has more data to send
WEP	Wired equivalent privacy (encryption implemented)
Rsvd	Reserved

- ***D.*** This field defines the duration of the transmission that is used to set the value of NAV. In one control frame, it defines the ID of the frame.
- ***Addresses.*** There are four address fields, each 6 bytes long. The meaning of each address field depends on the value of the *To DS* and *From DS* subfields and will be discussed later.
- ***Sequence control.*** This field, often called the *SC* field, defines a 16-bit value. The first four bits define the fragment number; the last 12 bits define the sequence number, which is the same in all fragments.
- ***Frame body.*** This field, which can be between 0 and 2312 bytes, contains information based on the type and the subtype defined in the FC field.
- ***FCS.*** The FCS field is 4 bytes long and contains a CRC-32 error-detection sequence.

Frame Types

A wireless LAN defined by IEEE 802.11 has three categories of frames: management frames, control frames, and data frames.

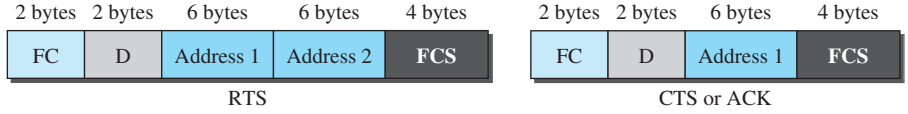
Management Frames

Management frames are used for the initial communication between stations and access points.

Control Frames

Control frames are used for accessing the channel and acknowledging frames. Figure 15.10 shows the format.

Figure 15.10 *Control frames*



For control frames the value of the type field is 01; the values of the subtype fields for frames we have discussed are shown in Table 15.2.

Table 15.2 *Values of subtype fields in control frames*

Subtype	*Meaning*
1011	Request to send (RTS)
1100	Clear to send (CTS)
1101	Acknowledgment (ACK)

Data Frames

Data frames are used for carrying data and control information.

15.2.3 Addressing Mechanism

The IEEE 802.11 addressing mechanism specifies four cases, defined by the value of the two flags in the FC field, *To DS* and *From DS*. Each flag can be either 0 or 1, resulting in four different situations. The interpretation of the four addresses (address 1 to address 4) in the MAC frame depends on the value of these flags, as shown in Table 15.3.

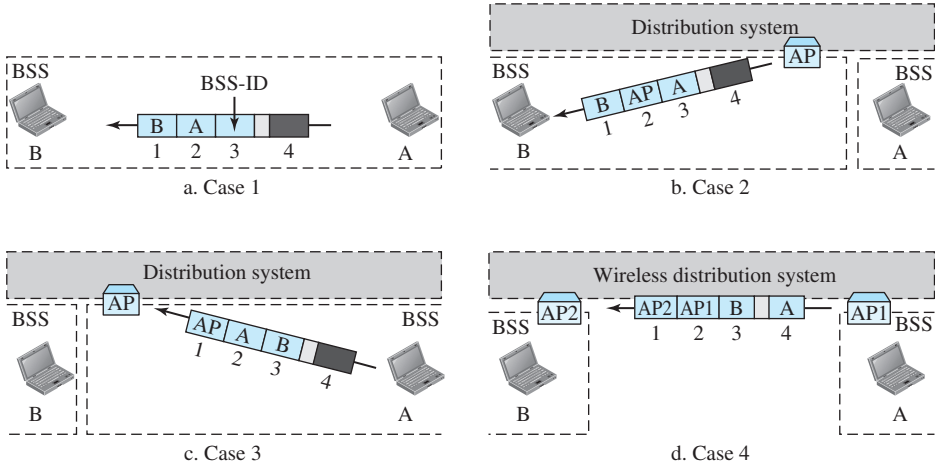
Table 15.3 *Addresses*

To DS	*From DS*	*Address 1*	*Address 2*	*Address 3*	*Address 4*
0	0	Destination	Source	BSS ID	N/A
0	1	Destination	Sending AP	Source	N/A
1	0	Receiving AP	Source	Destination	N/A
1	1	Receiving AP	Sending AP	Destination	Source

Note that address 1 is always the address of the next device that the frame will visit. Address 2 is always the address of the previous device that the frame has left. Address 3 is the address of the final destination station if it is not defined by address 1 or the original source station if it is not defined by address 2. Address 4 is the original source when the distribution system is also wireless.

- **Case 1: 00** In this case, *To DS* = 0 and *From DS* = 0. This means that the frame is not going to a distribution system (*To DS* = 0) and is not coming from a distribution system (*From DS* = 0). The frame is going from one station in a BSS to another without passing through the distribution system. The addresses are shown in Figure 15.11.
- **Case 2: 01** In this case, *To DS* = 0 and *From DS* = 1. This means that the frame is coming from a distribution system (*From DS* = 1). The frame is coming from an

Figure 15.11 *Addressing mechanisms*



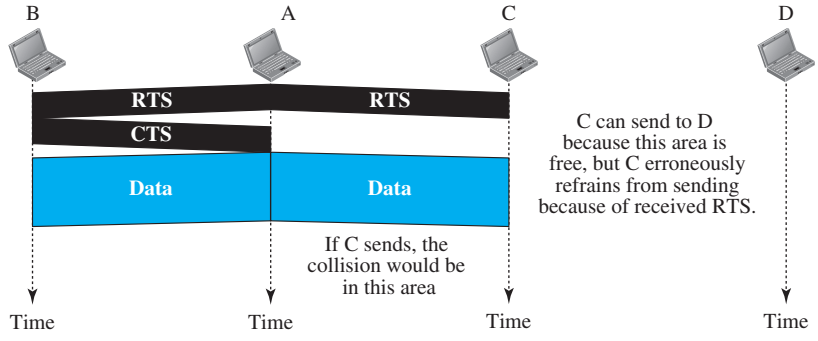
AP and going to a station. The addresses are as shown in Figure 15.11. Note that address 3 contains the original sender of the frame (in another BSS).

- **Case 3: 10** In this case, *To DS* = 1 and *From DS* = 0. This means that the frame is going to a distribution system (*To DS* = 1). The frame is going from a station to an AP. The ACK is sent to the original station. The addresses are as shown in Figure 15.11. Note that address 3 contains the final destination of the frame in the distribution system.
- **Case 4: 11** In this case, *To DS* = 1 and *From DS* = 1. This is the case in which the distribution system is also wireless. The frame is going from one AP to another AP in a wireless distribution system. Here, we need four addresses to define the original sender, the final destination, and two intermediate APs. Figure 15.11 shows the situation.

Exposed Station Problem

We discussed how to solve the hidden station problem. A similar problem is called the *exposed station problem*. In this problem a station refrains from using a channel when it is, in fact, available. In Figure 15.12, station A is transmitting to station B. Station C has some data to send to station D, which can be sent without interfering with the transmission from A to B. However, station C is exposed to transmission from A; it hears what A is sending and thus refrains from sending. In other words, C is too conservative and wastes the capacity of the channel. The handshaking messages RTS and CTS cannot help in this case. Station C hears the RTS from A and refrains from sending, even though the communication between C and D cannot cause a collision in the zone between A and C; station C cannot know that station A's transmission does not affect the zone between C and D.

Figure 15.12 *Exposed station problem*



15.2.4 Physical Layer

We discuss six specifications, as shown in Table 15.4. All implementations, except the infrared, operate in the *industrial, scientific,* and *medical (ISM)* band, which defines three unlicensed bands in the three ranges 902–928 MHz, 2.400–4.835 GHz, and 5.725–5.850 GHz.

Table 15.4 *Specifications*

IEEE	*Technique*	*Band*	*Modulation*	*Rate (Mbps)*
802.11	FHSS	2.400–4.835 GHz	FSK	1 and 2
	DSSS	2.400–4.835 GHz	PSK	1 and 2
	None	Infrared	PPM	1 and 2
802.11a	OFDM	5.725–5.850 GHz	PSK or QAM	6 to 54
802.11b	DSSS	2.400–4.835 GHz	PSK	5.5 and 11
802.11g	OFDM	2.400–4.835 GHz	Different	22 and 54
802.11n	OFDM	5.725–5.850 GHz	Different	600

IEEE 802.11 FHSS

IEEE 802.11 FHSS uses the **frequency-hopping spread spectrum (FHSS)** method, as discussed in Chapter 6. FHSS uses the 2.400–4.835 GHz ISM band. The band is divided into 79 subbands of 1 MHz (and some guard bands). A pseudorandom number generator selects the hopping sequence. The modulation technique in this specification is either two-level FSK or four-level FSK with 1 or 2 bits/baud, which results in a data rate of 1 or 2 Mbps, as shown in Figure 15.13.

IEEE 802.11 DSSS

IEEE 802.11 DSSS uses the **direct-sequence spread spectrum (DSSS)** method, as discussed in Chapter 6. DSSS uses the 2.400–4.835 GHz ISM band. The modulation technique in this specification is PSK at 1 Mbaud/s. The system allows 1 or 2 bits/baud (BPSK or QPSK), which results in a data rate of 1 or 2 Mbps, as shown in Figure 15.14.

Figure 15.13 *Physical layer of IEEE 802.11 FHSS*

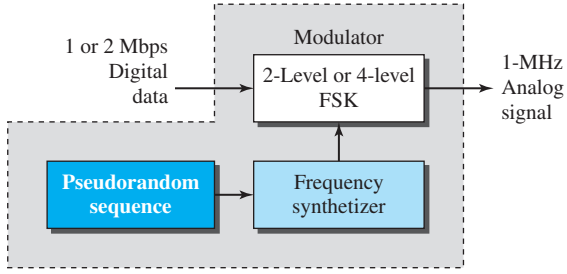
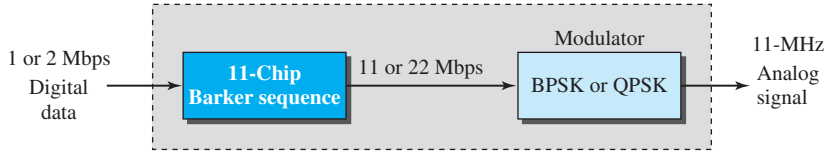


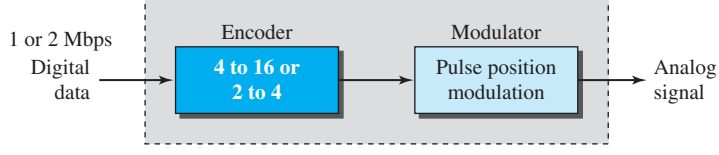
Figure 15.14 *Physical layer of IEEE 802.11 DSSS*



IEEE 802.11 Infrared

IEEE 802.11 infrared uses infrared light in the range of 800 to 950 nm. The modulation technique is called ***pulse position modulation (PPM).*** For a 1-Mbps data rate, a 4-bit sequence is first mapped into a 16-bit sequence in which only one bit is set to 1 and the rest are set to 0. For a 2-Mbps data rate, a 2-bit sequence is first mapped into a 4-bit sequence in which only one bit is set to 1 and the rest are set to 0. The mapped sequences are then converted to optical signals; the presence of light specifies 1, the absence of light specifies 0. See Figure 15.15.

Figure 15.15 *Physical layer of IEEE 802.11 infrared*



IEEE 802.11a OFDM

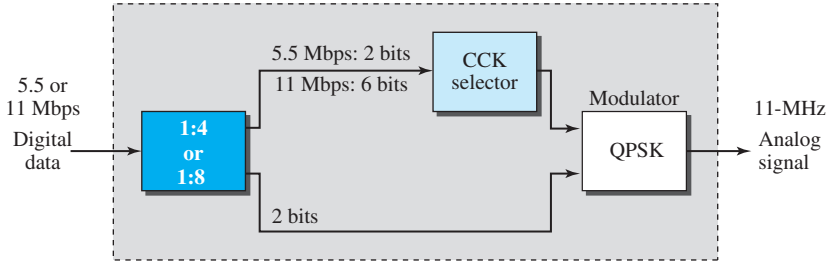
IEEE 802.11a OFDM describes the **orthogonal frequency-division multiplexing (OFDM)** method for signal generation in a 5.725–5.850 GHz ISM band. OFDM is similar to FDM, as discussed in Chapter 6, with one major difference: All the

subbands are used by one source at a given time. Sources contend with one another at the data-link layer for access. The band is divided into 52 subbands, with 48 subbands for sending 48 groups of bits at a time and 4 subbands for control information. Dividing the band into subbands diminishes the effects of interference. If the subbands are used randomly, security can also be increased. OFDM uses PSK and QAM for modulation. The common data rates are 18 Mbps (PSK) and 54 Mbps (QAM).

IEEE 802.11b DSSS

IEEE 802.11b DSSS describes the **high-rate direct-sequence spread spectrum (HR-DSSS)** method for signal generation in the 2.400–4.835 GHz ISM band. HR-DSSS is similar to DSSS except for the encoding method, which is called ***complementary code keying (CCK).*** CCK encodes 4 or 8 bits to one CCK symbol. To be backward compatible with DSSS, HR-DSSS defines four data rates: 1, 2, 5.5, and 11 Mbps. The first two use the same modulation techniques as DSSS. The 5.5-Mbps version uses BPSK and transmits at 1.375 Mbaud/s with 4-bit CCK encoding. The 11-Mbps version uses QPSK and transmits at 1.375 Mbps with 8-bit CCK encoding. Figure 15.16 shows the modulation technique for this standard.

Figure 15.16 *Physical layer of IEEE 802.11b*



IEEE 802.11g

This new specification defines forward error correction and OFDM using the 2.400–4.835 GHz ISM band. The modulation technique achieves a 22- or 54-Mbps data rate. It is backward-compatible with 802.11b, but the modulation technique is OFDM.

IEEE 802.11n

An upgrade to the 802.11 project is called 802.11n (the next generation of wireless LAN). The goal is to increase the throughput of 802.11 wireless LANs. The new standard emphasizes not only the higher bit rate but also eliminating some unnecessary overhead. The standard uses what is called ***MIMO (multiple-input multiple-output antenna)*** to overcome the noise problem in wireless LANs. The idea is that if we can send multiple output signals and receive multiple input signals, we are in a better

position to eliminate noise. Some implementations of this project have reached up to 600 Mbps data rate.

15.3 BLUETOOTH

Bluetooth is a wireless LAN technology designed to connect devices of different functions such as telephones, notebooks, computers (desktop and laptop), cameras, printers, and even coffee makers when they are at a short distance from each other. A Bluetooth LAN is an ad hoc network, which means that the network is formed spontaneously; the devices, sometimes called gadgets, find each other and make a network called a piconet. A Bluetooth LAN can even be connected to the Internet if one of the gadgets has this capability. A Bluetooth LAN, by nature, cannot be large. If there are many gadgets that try to connect, there is chaos.

Bluetooth technology has several applications. Peripheral devices such as a wireless mouse or keyboard can communicate with the computer through this technology. Monitoring devices can communicate with sensor devices in a small health care center. Home security devices can use this technology to connect different sensors to the main security controller. Conference attendees can synchronize their laptop computers at a conference.

Bluetooth was originally started as a project by the Ericsson Company. It is named for Harald Blaatand, the king of Denmark (940-981) who united Denmark and Norway. *Blaatand* translates to *Bluetooth* in English.

Today, Bluetooth technology is the implementation of a protocol defined by the IEEE 802.15 standard. The standard defines a wireless personal-area network (PAN) operable in an area the size of a room or a hall.

15.3.1 Architecture

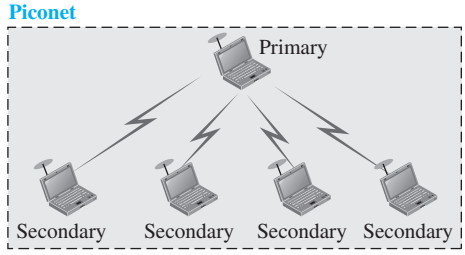
Bluetooth defines two types of networks: piconet and scatternet.

Piconets

A Bluetooth network is called a ***piconet,*** or a small net. A piconet can have up to eight stations, one of which is called the *primary;* the rest are called *secondaries.* All the secondary stations synchronize their clocks and hopping sequence with the primary. Note that a piconet can have only one primary station. The communication between the primary and secondary stations can be one-to-one or one-to-many. Figure 15.17 shows a piconet.

Although a piconet can have a maximum of seven secondaries, additional secondaries can be in the *parked state*. A secondary in a parked state is synchronized with the primary, but cannot take part in communication until it is moved from the parked state to the active state. Because only eight stations can be active in a piconet, activating a station from the parked state means that an active station must go to the parked state.

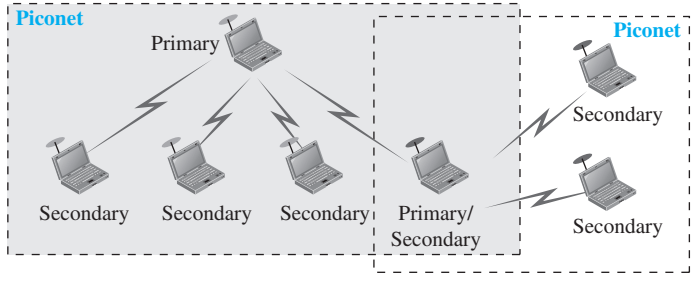
Figure 15.17 *Piconet*



Scatternet

Piconets can be combined to form what is called a ***scatternet.*** A secondary station in one piconet can be the primary in another piconet. This station can receive messages from the primary in the first piconet (as a secondary) and, acting as a primary, deliver them to secondaries in the second piconet. A station can be a member of two piconets. Figure 15.18 illustrates a scatternet.

Figure 15.18 *Scatternet*



Bluetooth Devices

A Bluetooth device has a built-in short-range radio transmitter. The current data rate is 1 Mbps with a 2.4-GHz bandwidth. This means that there is a possibility of interference between the IEEE 802.11b wireless LANs and Bluetooth LANs.

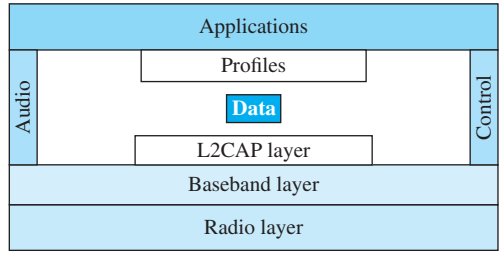
15.3.2 Bluetooth Layers

Bluetooth uses several layers that do not exactly match those of the Internet model we have defined in this book. Figure 15.19 shows these layers.

L2CAP

The **Logical Link Control and Adaptation Protocol,** or **L2CAP** (L2 here means LL), is roughly equivalent to the LLC sublayer in LANs. It is used for data exchange on an

Figure 15.19 *Bluetooth layers*



ACL link; SCO channels do not use L2CAP. Figure 15.20 shows the format of the data packet at this level.

Figure 15.20 *L2CAP data packet format*

The 16-bit length field defines the size of the data, in bytes, coming from the upper layers. Data can be up to 65,535 bytes. The channel ID (CID) defines a unique identifier for the virtual channel created at this level (see below).

The L2CAP has specific duties: multiplexing, segmentation and reassembly, quality of service (QoS), and group management.

Multiplexing

The L2CAP can do multiplexing. At the sender site, it accepts data from one of the upper-layer protocols, frames them, and delivers them to the baseband layer. At the receiver site, it accepts a frame from the baseband layer, extracts the data, and delivers them to the appropriate protocol layer. It creates a kind of virtual channel that we will discuss in later chapters on higher-level protocols.

Segmentation and Reassembly

The maximum size of the payload field in the baseband layer is 2774 bits, or 343 bytes. This includes 4 bytes to define the packet and packet length. Therefore, the size of the packet that can arrive from an upper layer can only be 339 bytes. However, application layers sometimes need to send a data packet that can be up to 65,535 bytes (an Internet packet, for example). The L2CAP divides these large packets into segments and adds extra information to define the location of the segments in the original packet. The L2CAP segments the packets at the source and reassembles them at the destination.

QoS

Bluetooth allows the stations to define a quality-of-service level. We discuss quality of service in Chapter 30. For the moment, it is sufficient to know that if no quality-of-service

level is defined, Bluetooth defaults to what is called *best-effort* service; it will do its best under the circumstances.

Group Management

Another functionality of L2CAP is to allow devices to create a type of logical addressing between themselves. This is similar to multicasting. For example, two or three secondary devices can be part of a multicast group to receive data from the primary.

Baseband Layer

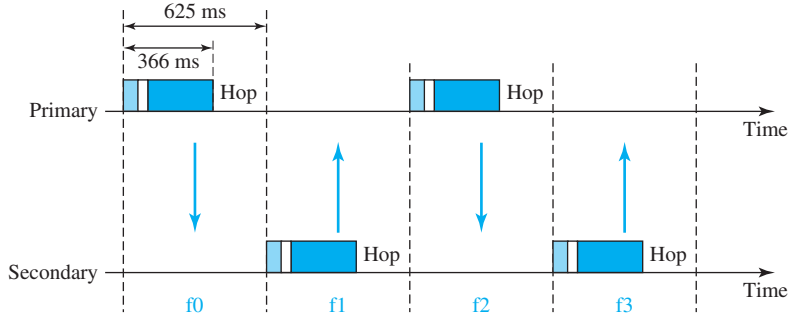
The baseband layer is roughly equivalent to the MAC sublayer in LANs. The access method is TDMA (discussed later). The primary and secondary stations communicate with each other using time slots. The length of a time slot is exactly the same as the dwell time, 625 μs. This means that during the time that one frequency is used, a primary sends a frame to a secondary, or a secondary sends a frame to the primary. Note that the communication is only between the primary and a secondary; secondaries cannot communicate directly with one another.

TDMA

Bluetooth uses a form of TDMA that is called ***TDD-TDMA (time-division duplex TDMA).*** TDD-TDMA is a kind of half-duplex communication in which the sender and receiver send and receive data, but not at the same time (half-duplex); however, the communication for each direction uses different hops. This is similar to walkie-talkies using different carrier frequencies.

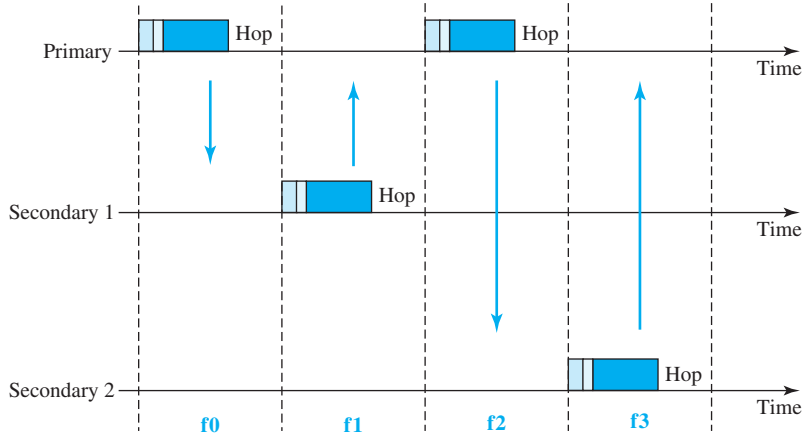
- ***Single-Secondary Communication*** If the piconet has only one secondary, the TDMA operation is very simple. The time is divided into slots of 625 μs. The primary uses even-numbered slots (0, 2, 4, . . .); the secondary uses odd-numbered slots (1, 3, 5, . . .). TDD-TDMA allows the primary and the secondary to communicate in half-duplex mode. In slot 0, the primary sends and the secondary receives; in slot 1, the secondary sends and the primary receives. The cycle is repeated. Figure 15.21 shows the concept.

Figure 15.21 *Single-secondary communication*



❑ ***Multiple-Secondary Communication*** The process is a little more involved if there is more than one secondary in the piconet. Again, the primary uses the even-numbered slots, but a secondary sends in the next odd-numbered slot if the packet in the previous slot was addressed to it. All secondaries listen on even-numbered slots, but only one secondary sends in any odd-numbered slot. Figure 15.22 shows a scenario.

Figure 15.22 *Multiple-secondary communication*



Let us elaborate on the figure.

1. In slot 0, the primary sends a frame to secondary 1.
2. In slot 1, only secondary 1 sends a frame to the primary because the previous frame was addressed to secondary 1; other secondaries are silent.
3. In slot 2, the primary sends a frame to secondary 2.
4. In slot 3, only secondary 2 sends a frame to the primary because the previous frame was addressed to secondary 2; other secondaries are silent.
5. The cycle continues.

We can say that this access method is similar to a poll/select operation with reservations. When the primary selects a secondary, it also polls it. The next time slot is reserved for the polled station to send its frame. If the polled secondary has no frame to send, the channel is silent.

Links

Two types of links can be created between a primary and a secondary: SCO links and ACL links.

❑ ***SCO*** A **synchronous connection-oriented (SCO) link** is used when avoiding latency (delay in data delivery) is more important than integrity (error-free delivery). In an SCO link, a physical link is created between the primary and a secondary by reserving specific slots at regular intervals. The basic unit of connection is two slots, one for each direction. If a packet is damaged, it is never retransmitted.

SCO is used for real-time audio where avoiding delay is all-important. A secondary can create up to three SCO links with the primary, sending digitized audio (PCM) at 64 kbps in each link.

- ***ACL*** An **asynchronous connectionless link (ACL)** is used when data integrity is more important than avoiding latency. In this type of link, if a payload encapsulated in the frame is corrupted, it is retransmitted. A secondary returns an ACL frame in the available odd-numbered slot if the previous slot has been addressed to it. ACL can use one, three, or more slots and can achieve a maximum data rate of 721 kbps.

Frame Format

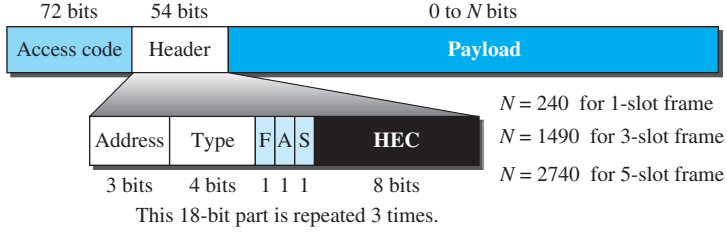
A frame in the baseband layer can be one of three types: one-slot, three-slot, or five-slot. A slot, as we said before, is 625 μs. However, in a one-slot frame exchange, 259 μs is needed for hopping and control mechanisms. This means that a one-slot frame can last only 625 – 259, or 366 μs. With a 1-MHz bandwidth and 1 bit/Hz, the size of a one-slot frame is 366 bits.

A three-slot frame occupies three slots. However, since 259 μs is used for hopping, the length of the frame is $3 \times 625 - 259 = 1616$ μs or 1616 bits. A device that uses a three-slot frame remains at the same hop (at the same carrier frequency) for three slots. Even though only one hop number is used, three hop numbers are consumed. That means the hop number for each frame is equal to the first slot of the frame.

A five-slot frame also uses 259 bits for hopping, which means that the length of the frame is $5 \times 625 - 259 = 2866$ bits.

Figure 15.23 shows the format of the three frame types.

Figure 15.23 *Frame format types*



The following describes each field:

- ***Access code.*** This 72-bit field normally contains synchronization bits and the identifier of the primary to distinguish the frame of one piconet from that of another.
- ***Header.*** This 54-bit field is a repeated 18-bit pattern. Each pattern has the following subfields:

a. ***Address.*** The 3-bit address subfield can define up to seven secondaries (1 to 7). If the address is zero, it is used for broadcast communication from the primary to all secondaries.
b. ***Type.*** The 4-bit type subfield defines the type of data coming from the upper layers. We discuss these types later.
c. ***F.*** This 1-bit subfield is for flow control. When set (1), it indicates that the device is unable to receive more frames (buffer is full).
d. ***A.*** This 1-bit subfield is for acknowledgment. Bluetooth uses Stop-and-Wait ARQ; 1 bit is sufficient for acknowledgment.
e. ***S.*** This 1-bit subfield holds a sequence number. Bluetooth uses Stop-and-Wait ARQ; 1 bit is sufficient for sequence numbering.
f. ***HEC.*** The 8-bit header error correction subfield is a checksum to detect errors in each 18-bit header section. The header has three identical 18-bit sections. The receiver compares these three sections, bit by bit. If each of the corresponding bits is the same, the bit is accepted; if not, the majority opinion rules. This is a form of forward error correction (for the header only). This double error control is needed because the nature of the communication, via air, is very noisy. Note that there is no retransmission in this sublayer.

- ***Payload.*** This subfield can be 0 to 2740 bits long. It contains data or control information coming from the upper layers.

Radio Layer

The radio layer is roughly equivalent to the physical layer of the Internet model. Bluetooth devices are low-power and have a range of 10 m.

Band

Bluetooth uses a 2.4-GHz ISM band divided into 79 channels of 1 MHz each.

FHSS

Bluetooth uses the **frequency-hopping spread spectrum (FHSS)** method in the physical layer to avoid interference from other devices or other networks. Bluetooth hops 1600 times per second, which means that each device changes its modulation frequency 1600 times per second. A device uses a frequency for only 625 μs (1/1600 s) before it hops to another frequency; the dwell time is 625 μs.

Modulation

To transform bits to a signal, Bluetooth uses a sophisticated version of FSK, called GFSK (FSK with Gaussian bandwidth filtering; a discussion of this topic is beyond the scope of this book). GFSK has a carrier frequency. Bit 1 is represented by a frequency deviation above the carrier; bit 0 is represented by a frequency deviation below the carrier. The frequencies, in megahertz, are defined according to the following formula for each channel.

$$f_c = 2402 + n \text{ MHz} \qquad n = 0, 1, 2, 3, \ldots, 78$$

For example, the first channel uses carrier frequency 2402 MHz (2.402 GHz), and the second channel uses carrier frequency 2403 MHz (2.403 GHz).